Tomus 8. Fasciculus 1.

# ACTA CYBERNETICA

FORUM CENTRALE PUBLICATIONUM
CYBERNETICARUM HUNGARICUM

Szeged, 1987

Curat: Universitas Szegediensis de Attila József nominata

8. kötet 1. füzet

# ACTA CYBERNETICA

A HAZAI KIBERNETIKAI KUTATÁSOK
KÖZPONTI PUBLIKÁCIÓS FÓRUMA

Szeged, 1987. április
A Szegedi József Attila Tudományegyetem gondozásában

# A note on zero-congruences

Stephen L. Bloom, Ralph Tindell

In this note, a theorem is proved about zero-congruences in iteration theories. If $T$ is an iteration theory (see [BEW1], [BEW2], [Es] and $\theta$ is a theory congruence (not necessarily a dagger congruence), we call $\theta$ a **zero congruence** if $f\theta g$ for all morphisms $f, g\colon 1\to 0$ in $T$. Recall that a theory congruence $\theta$ on $T$ is a family of equivalence relations $\theta_{n,p}$ on $T(n,p)$, $n,p\geqq 0$, which are preserved by composition and source pairing. A theory congruence $\theta$ is a dagger congruence if $f^\dagger\theta g^\dagger$ whenever $f\theta g$.

**1. Theorem.** *For any iteration theory $T$, the least zero congruence is a dagger congruence.*

The proof is constructive in the sense that the least zero congruence is described explicitly. The theorem is of interest for the following reason. For any set $A$, let Pfn $(A)$ denote the iteration theory whose morphisms $n\to p$ are the partial functions $A\times[n]\to A\times[p]$. Let PFN denote the variety of all iteration theories generated by those theories of the form Pfn $(A)$. In a forthcoming paper by Bloom and Ésik, it is shown that for any ranked set $\Gamma$, there is an iteration theory freely generated by $\Gamma$ in the variety PFN. This theory may be described as the quotient of the theory $\Gamma$tr freely generated by $\Gamma$ in the variety of all iteration theories by the least zero congruence. In the course of the study of that argument, it was discovered that the least zero congruence on $\Gamma$tr automatically preserved dagger. We wondered if this was a general phenomenon. The theorem shows that it is.

**2. Definition.** Let $\varrho=\varrho_{n,p}$ be the family of binary relations on $T(n,p)$ defined as follows: for any morphisms $f,f'\colon n\to p$, $f\varrho_{n,p}f'$ if there are morphisms $g\colon n\to k+p$ and $b,b'\colon k\to 0$ in $T$ such that $f=g\cdot(b+1_p)$ and $f'=g\cdot(b'+1_p)$.

Note that $\varrho$ is symmetric. Let $\varrho^*$ denote the reflexive, transitive closure of $\varrho$, which is to say that $f\varrho^*f'$ iff there is a finite sequence $f_1,f_2,\ldots,f_n$ with $f=f_1, f'=f_n$, and $f_i\varrho f_{i+1}$ for $1\leqq i\leqq n-1$, $n\geqq 1$.

**3. Lemma.** *If $f, f', g, g'$ are morphisms in $T$ with the appropriate sources and targets, and if $f\varrho f'$, $g\varrho g'$, then*

*a)* $$\langle f, g\rangle\varrho\langle f', g'\rangle,$$

*b)* $$f\cdot g\,\varrho f'\cdot g',$$

*and*

*c)* $$f^\dagger \varrho g^\dagger$$

*Proof* of *a)*. Suppose that

$$f = F\cdot(b+1_p),\quad f' = F\cdot(b'+1_p),$$

where $F\colon n\to k+p$ and $b, b'\colon k\to 0$. Suppose further that

$$g = G\cdot(c+1_p),\quad g' = G\cdot(c'+1_p),$$

where $G\colon m\to r+p$ and $c, c'\colon r\to 0$. Then

$$\langle f, g\rangle = H\cdot(b+c+1_p) \quad\text{and}\quad \langle f', g'\rangle = H\cdot(b'+c'+1_p),$$

where $H=\langle F\cdot(\varkappa+1_p), G\cdot(\lambda+1_p)\rangle$, $\varkappa=1_k+0_r$, and $\lambda=0_k+1_r$.

*Proof* of *b)*. We assume

$$f = F\cdot(b+1_p),\quad f' = F\cdot(b'+1_p) \quad\text{and}\quad g = G\cdot(c+1_s),\quad g' = G\cdot(c'+1_s),$$

where $F\colon n\to k+p$ and $G\colon p\to r+s$. Then

$$f\cdot g = F\cdot(b+1_p)\cdot G\cdot(c+1_s) = F\cdot(1_k+G)\cdot(b+c+1_s)$$

and

$$f'\cdot g' = F\cdot(1_k+G)\cdot(b'+c'+1_s).$$

*Proof* of *c)*. Suppose that $f=F\cdot(b+1_{p+n})$ and $g=F\cdot(b'+1_{p+n})$, where $F\colon n\to k+p+n$ and $b, b'\colon k\to 0$ in $T$. Then $f^\dagger=F^\dagger\cdot(b+1_p)$ and $g^\dagger=F^\dagger(b'+1_p)$.

It follows immediately from Lemma 3 that $\varrho^*$ is a theory congruence.

**4. Lemma.** *$\varrho^*$ is the least zero congruence.*

*Proof.* Let $\theta$ be any zero congruence. It is clear that if $f\varrho g$, then $f\theta g$. Thus $f\varrho^* g$ implies $f\theta g$. If $\theta$ is the least zero congruence, the converse also holds (i.e. if $f\theta g$, then $f\varrho^* g$).

The proof of the theorem follows from the preceding two facts.

$$\begin{tikzcd}
 & a \arrow[dl] \arrow[d] \arrow[dr] & \\
1 & a \arrow[dl] \arrow[d] \arrow[dr] & \perp \\
1 & a \arrow[dl] \arrow[d] \arrow[dr] & b\cdot\perp \\
1 & a & b^2\cdot\perp \\
 & \vdots & \\
 & a \arrow[dl] \arrow[d] \arrow[dr] & \\
1 & a & b^n\cdot\perp \\
 & \vdots &
\end{tikzcd}$$

*Figure 1*

**5. Remark.** Aside from the fact that iteration theories are algebraic theories, the only property of iteration theories used in the above proof is the validity of the identity

$$[F\cdot(b+1_{p+n})]^{\dagger} = F^{\dagger}\cdot(b+1_p),$$

for any $F\colon n\to k+p+n$, $b\colon k\to 0$. Since this identity is also valid in all iterative theories, and in all (ordered) rational theories [ADJ], the theorem holds for these theories as well.

**6. Example.** Let $T$ be the iteration theory of all $\Gamma$-trees (not just those of finite index [EBT]). Let $a\colon 1\to 3$, $b\colon 1\to 1$, $\perp\colon 1\to 0$ be atomic. Let $f$ be the infinite tree indicated in Figure 1. Note that $f$ has infinitely many subtrees $1\to 0$. If $g$ is the tree indicated in Figure 2, then $g$ is $h^{\dagger}$, where $h=a\cdot(1_2+\perp)\colon 1\to 2$. Note that $f$ and $g$ are not congruent by the least zero congruence, since clearly $f$ is not related by $\varrho^*$ to $g$. Among the trees related by $\varrho^*$ to $f$ are those indicated in Figure 3.

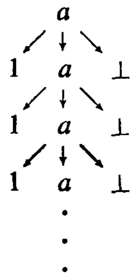


Figure 2

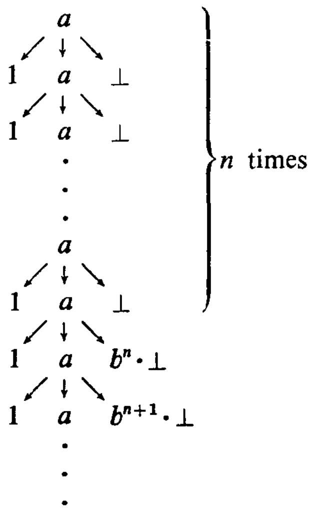


Figure 3

DEPARTMENT OF COMPUTER SCIENCE
STEVENS INSTITUTE OF TECHNOLOGY
HOBOKEN, NJ

## References

[ADJ] J. B. Wright, J. W. Thatcher, E. G. Wagner, "Rational algebraic theories and fixed-point solutions", Proceedings 17th IEEE Symposium, Foundations of Computing, Houston (1976).

[BEW1] S. L. Bloom, C. C. Elgot, J. B. Wright, "Solutions of the iteration equation and extensions of the scalar iteration operation", SIAM J. Computing, 9 (1980), 25—45.

[BEW2] S. L. Bloom, C. C. Elgot, J. B. Wright, "Vector iteration in pointed iterative theories", SIAM J. Computing, 9 (1980), 525—540.

[EBT] C. Elgot, S. L. Bloom, R. Tindell, "On the algebraic structure of rooted trees", Journal of Computer and System Sciences, 16 (1978), 362—399.

[Es] Z. Ésik, "Identities in iterative and rational theories", Comput. Linguistics and Comput. Languages XIV (1980) 183—207.

*(Received Febr. 26, 1986)*

# On the expected behaviour of the *NF* algorithm for a dual bin-packing problem

J. Csirik, G. Galambos

## Introduction

The following version of dual bin-packing problems was first studied by Assmann et al [1]: there is given a list $L=\{a_1, a_2, \dots, a_n\}$ of items (elements) and a size $s(a_i)$ for each item. Let us denote $C$ as a positive constant, $C \geqq \max_{1 \leqq i \leqq n} s(a_i)$. The aim is to pack the elements into a *maximum* number of bins so that the sum of the sizes in any given bin is *at least* $C$. (The name "dual" originates from the "classical" bin-packing problem, where the elements have to be packed into the *minimum* number of bins in such a way that the sum of the sizes of the elements in any given bin is *at most* $C$). This problem is *NP-hard* and hence the investigation of the performance of approximation algorithms is important. Without loss of generality, we may assume that $C=1$ and $0<s(a_i)\leqq 1$; if $a_i$ is real, then $s(a_i)=a_i$.

The worst-case behaviour of the well-known heuristic algorithms Next-Fit (*NF*) and Next-Fit Decreasing (*NFD*) was analysed in [1]. The *NF* algorithm places $a_1$ into the first bin ($B_1$). Let us suppose that $a_i$, $i>1$, is to be packed, and let $B_j (j \geqq 1)$ be the highest indexed non-empty bin. The algorithm places $a_i$ into $B_j$ if the sum of elements in this bin (so far) is smaller than 1; otherwise it closes the bin $B_j$, opens a new bin ($B_{j+1}$) and places the element $a_i$ into this newly-opened bin. The *NFD* algorithm differs from *NF* only in preordering the elements. The worst-case behaviour of an approximation algorithm may be characterized by means of the asymptotic worst-case ratio. To define this, let $OPT(L)$ be the maximum possible number of bins for a given instance $L$. For a given approximation algorithm $A$, let $A(L)$ denote the number of bins used by $A$ to pack $L$. Let

$R_A^N = \min \{A(L)/OPT(L)\colon L$ is an instance with $OPT(L)=N\}$.

The asymptotic worst-case ratio for $A$ is then defined as

$$R_A^\infty = \liminf_{N\to\infty} R_A^N$$

Assmann et al proved that

$$R_{NF}^\infty = R_{NFD}^\infty = 1/2$$

In the last part of that paper the average-case behaviour of these (and other) algorithms was investigated. The behaviours of the algorithms were compared in randomly generated instances, where the elements of $L$ were drawn from different distributions. In the conclusion of the paper it was suggested that the expected performance of these algorithms should be examined analytically as well. In [3] this analysis has been carried out for both algorithms. Csirik et al [3] showed that if the elements of $L=(a_1, a_2, \ldots, a_n)$ are identically distributed and drawn independently from a uniform distribution on $(0, 1/k]$ ($k$ is a positive integer), then

| | $k=1$ | $k=2$ | $k=3$ | $k=4$ |
|---|---|---|---|---|
| $R_{NF}^{\infty}$ | 0.735 | 0.8564 | 0.900 | 0.923 |
| and $R_{NFD}^{\infty}$ | 0.710 | 0.840 | 0.891 | 0.918 |

On the other hand, Knödel [4] showed that the first-fit ($FF$) algorithm is asymptotically optimal for the "classical" bin-packing problem if the elements of the input list are drawn independently from the following distribution:

$$a_i = \begin{cases} 1/3 \text{ with probability } 1/3, \\ 2/3 \text{ with probability } 1/3, \\ \ 1 \ \text{ with probability } 1/3. \end{cases}$$

(In the $FF$ algorithm we try to pack the element $a_i$ into all opened bins, i.e. into $B_1, B_2, \ldots, B_i$, and open a new bin only if none of them has enough room for it.)

Csirik [2] generalized this result for the input sequence:

$$a_i = \begin{cases} b & \text{with probability } 1/2, \\ 1-b & \text{with probability } 1/2, \end{cases} \tag{I}$$

where $0<b<1/2$ and it was proved that the $FF$ is asymptotically optimal for these sequences, too.

In this note we investigate the expected behaviour of the $NF$ algorithm at sequence (I) for the dual version of the bin-packing problem.

## Results

First we present our method for special lists. Let the elements of $L=(a_1, a_2, \ldots, a_n)$ be chosen independently of the following distribution:

$$a_i = \begin{cases} 1/3 \text{ with probability } 1/2, \\ 2/3 \text{ with probability } 1/2. \end{cases} \tag{1}$$

Let us denote by $E_n$ the expected number of full bins for the lists $L=(a_1, a_2, \ldots, a_n)$ (the elements are drawn independently from (1)), and by $E_{n,k}$ the expected number of bins for lists with a number $k$ of 2/3 elements (and so a number $(n-k)$ of 1/3 elements) if we pack $L$ by $NF$. Then

$$E_n = \frac{1}{2^n} \sum_{k=0}^{n} \binom{n}{k} E_{n,k} \tag{2}$$

On the other hand, in the packing of $L$ by $NF$ the first bin will be full after packing $a_1$ and $a_2$ if at least one of them is a 2/3 element. If both of them are 1/3 elements, then the first bin is full after the packing of $a_3$. Thus, we have the following recursion:

$$E_{n,k} = \frac{k(k-1)}{n(n-1)}(E_{n-2,k-2}+1)+2\frac{k(n-k)}{n(n-1)}(E_{n-2,k-1}+1)+$$
$$+\frac{(n-k)(n-k-1)}{n(n-1)}\left(\frac{n-k-2}{n-2}E_{n-3,k}+\frac{k}{n-2}E_{n-3,k-1}+1\right), \quad \text{if} \quad n \geqq 3 \quad \text{and} \quad k \geqq 1. \tag{3}$$

It is easy to see that $E_{n,0}=\lfloor n/3 \rfloor$, $E_{1,1}=0$, $E_{2,1}=1$, $E_{2,2}=1$.
Using (3), from (2) we get:

$$E_n = \frac{3}{4}E_{n-2}+\frac{1}{4}E_{n-3}+1 \tag{4}$$

and we know that $E_0=0$, $E_1=0$, $E_2=3/4$.
Let us search $E_n$ in the following form:

$$E_n = \frac{n}{2}(1-A)-B_n \tag{5}$$

Then from (4) we have

$$B_n = \frac{3}{4}B_{n-2}+\frac{1}{4}B_{n-3}+\frac{1}{8}(1-9A) \tag{6}$$

Our aim is to give the asymptotic behaviour of $E_n$. From (5) it would be enough to choose an appropriate $A$ so that $|B_n|<T$, where $T$ is a constant. If now $A=1/9$, then from (6)

$$B_n = \frac{3}{4}B_{n-2}+\frac{1}{4}B_{n-3} \tag{7}$$

and from (4) and (5)

$$B_1 = \frac{4}{9}, \quad B_2 = \frac{5}{36}, \quad B_3 = \frac{1}{3}$$

By induction on $i$, we can prove from (7) that for all $i \geqq 4$

$$5/36 \leqq B_i \leqq 4/9$$

and hence $B_i$ is bounded. But then from (5) we have the following

**Lemma.**

$$\lim_{n\to\infty} \frac{E_n}{n/2} = 8/9.$$

We now generalize our result for the following input sequences: let the elements of $L=(a_1, a_2, \ldots, a_n)$ be independent, identically distributed, random variables with distribution (I). Let $l_1=\lceil 1/b \rceil$. We use the notation $E_n$ in the above sense, and let $E_{n,k}$ denote the expected number of bins for the lists $L=(a_1, a_2, \ldots, a_n)$ with a number $k$ of elements $1-b$. Then (2) is true for these sequences as well, and our lemma is valid for $l_1=3$.

In the packing $L$ by $NF$ we have two cases:

1. If $a_1=1-b$, then the first bin is always full after the packing of $a_2$.

2. If $a_1=b$, then the first bin is full with the first element $1-b$ in the sequel $a_2, a_3, \dots, a_{l_1-1}$. If all of $a_2, \dots, a_{l_1-1}$ are equal $b$, then the first bin is full after packing of the element $a_{l_1}$.

Similarly to (3), from these two cases we have

$$E_{n,k} = \frac{k(k-1)}{n(n-1)}(E_{n-2,k-2}+1)+2\frac{k(n-k)}{n(n-1)}(E_{n-2,k-1}+1)+$$

$$+\frac{(n-k)(n-k-1)k}{n(n-1)(n-2)}(E_{n-3,k-1}+1)+\dots+\frac{(n-k)(n-k-1)\dots(n-k-l_1+3)k}{n(n-1)\dots(n-l_1+2)}\times$$

$$\times(E_{n-l_1+1,k-1}+1)+\frac{(n-k)(n-k+1)\dots(n-k-l_2+2)}{n(n-1)\dots(n-l_1+2)}\times$$

$$\times\left(\frac{n-k-l_1+1}{n-l_1+1}E_{n-l_1,k}+\frac{k}{n-l_1+1}E_{n-l_1,k-1}+1\right) \tag{8}$$

and hence from (2)

$$E_n = \frac{3}{4}E_{n-2}+\frac{1}{2^3}E_{n-3}+\dots+\frac{1}{2^{l_1-1}}E_{n-l_1+1}+\frac{1}{2^{l_1-1}}E_{n-l_1}+1 \tag{9}$$

We look for $E_n$ again in the form given in (5). Then

$$B_n = \frac{3}{4}B_{n-2}+\frac{1}{2^3}B_{n-3}+\dots+\frac{1}{2^{l_1-1}}B_{n-l_1+1}+\frac{1}{2^{l_1-1}}B_{n-l_1}+$$

$$+(1-A)\left[\frac{n}{2}\left(1-\frac{3}{4}-\frac{1}{2^3}-\dots-\frac{1}{2^{l_1-1}}-\frac{1}{2^{l_1-1}}\right)\right]+$$

$$+(1-A)\left(\frac{3}{4}\cdot\frac{2}{2}+\frac{1}{2^3}\cdot\frac{3}{2}+\frac{1}{2^4}\cdot\frac{4}{2}+\dots+\frac{1}{2^{l_1-1}}\frac{l_1-1}{2}+\frac{1}{2^{l_1-1}}\frac{1}{2}\right)-1 =$$

$$= \frac{3}{4}B_{n-2}+\frac{1}{2^3}B_{n-3}+\dots+\frac{1}{2^{l_1-1}}B_{n-l_1+1}+\frac{1}{2^{l_1-1}}B_{n-l_1}+\frac{2^{l_1-2}-1}{2^{l_1}}-A\frac{2^{l_1}+2^{l_1-2}-1}{2^{l_1}}. \tag{10}$$

If we now choose

$$A = \frac{2^{l_1-2}-1}{2^{l_1}+2^{l_1-2}-1}$$

then

$$B_n = \frac{3}{4}B_{n-2}+\frac{1}{2^3}B_{n-3}+\dots+\frac{1}{2^{l_1-1}}B_{n-l_1+1}+\frac{1}{2^{l_1-1}}B_{n-l_1}$$

and thus $B_n$ is again a bounded sequence. Accordingly, we have proved our main result:

**Theorem.** Let the elements of $L=(a_1, a_2, ..., a_n)$ be independent, identically distributed, random variables with distribution

$$a_i = \begin{cases} b & \text{with probability } 1/2, \\ 1-b & \text{with probability } 1/2, \end{cases}$$

where $0<b<1/2$. Let $l_1=\ulcorner 1/b \urcorner$. If we pack the list $L$ by the *NF* algorithm and if $E_n$ denotes the expected number of filled bins, then

$$\lim_{n\to\infty} \frac{E_n}{n/2} = \frac{2^{l_1}}{2^{l_1}+2^{l_1-2}-1}$$

J. CSIRIK
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF SZEGED
ARADI VÉRTANÚK TERE 1.
SZEGED
HUNGARY

G. GALAMBOS
KALMÁR LABORATORY
OF CYBERNETICS
ÁRPÁD TÉR 2.
SZEGED
HUNGARY

## References

[1] Assmann, S. F., Johnson, D. S., Kleitman, D. J., Leung, J. Y. T.: On a dual version of the one-dimensional Bin-packing problem, Journal of Algorithm, 5 (1984), 505.
[2] Csirik, J.: Bin packing as a random walk: a note on Knödel's paper, Op. Res. Letters, 5 (1986), 161.
[3] Csirik, J., Frenk, J. B. G., Galambos, G., Rinnooy Kan, A. H. G.: Expected performance of simple algorithms for a dual bin-packing problem, to be publised.
[4] Knödel, W.: Über das mittlere Verhalten von On-Line Packungs Algorithmen, EIK 19 (1983), 427.

*(Received Apr. 6, 1986)*

# On the supplement of sets in functional systems

V. B. KUDRYAVTSEV

## Introduction

One of the main problems for functional systems (f.s.) [1] is that of completeness. It consists in indicating all subsets whose functions make a complete set of the functions of a given f.s. by means of f.s. operations. Such subsets are called complete. This problem is closely related to a supplementation problem, i.e. to the question of comparison of representable possibilities of two sets of functions under consideration. The problem is to find out when one of the sets is extended to a complete set "more easily" than the other, and when they behave identically in this sense. The paper consists of three sections. Section 1 deals with the problem on supplement for the systems $\mathcal{P}=(P, I)$ of a general type where $P$ is a set and $I$ is a closure operator determined on the subsets of the set $P$. In Section 2 the results of Section 1 are applied to the supplementation problem for finite f.s. Section 3 deals with an analysis of two-valued logics. For major notions see [1, 2].

## § 1. Supplementation problem for the system $\mathcal{P}$

Let us consider a pair $(P, I)$ or $\mathcal{P}$ in brief. $P$ is here a nonempty set, $I$ is a closure operator determined on the set $\mathcal{B}(P)$ of all subsets of $P$ i.e. $I$ possesses the properties that $I(Q)\supseteq Q$, $I(I(Q))=I(Q)$ and $I(Q_1)\supseteq I(Q_2)$ if $Q_1\supseteq Q_2$ for all $Q, Q_1, Q_2\in\mathcal{B}(P)$. The set $I(Q)$ is called the closure of $Q$, the set $Q$ is called closed if $I(Q)=Q$ and is called complete if $I(Q)=P$. The completeness problem for $\mathcal{P}$ consists in finding all complete sets. As mentioned above this problem is the main one for $\mathcal{P}$. It may be interpreted in a broader sense. Namely, to find out $Q'$ for a given $Q$ what supplements $Q'$ make it a complete set. In the case when $Q$ is empty we have a completeness problem. The treatment leads to the following question. Let $Q_1$ and $Q_2$ be given, we are to know which of them is "nearer" to being complete, or to be more precise, when with equal supplements $Q'$ the completeness of $Q_1\cup Q'$ will follow from the completeness of $Q_2\cup Q'$. We shall denote this relation by $Q_1\Box Q_2$. It is easy to see that it is equivalent to $I(Q_1)\Box I(Q_2)$, therefore we can consider $Q$ to be closed sets. Let us denote by $\mathcal{B}(\mathcal{P})$ the set of all closed subsets from $P$ and consider the relation $\Box$ on $\mathcal{B}(\mathcal{P})$. It is clear that this relation is reflexive and transitive, and as a relation of preorder it reduces to the equivalence relation $\approx$ on $\mathscr{B}(\mathscr{P})$ determined by both $Q_1 \square Q_2$ and $Q_2 \square Q_1$ and to the partial order relation $\prec$ on the factor set $\tilde{\mathscr{B}}(\mathscr{P})$ of the set $\mathscr{B}(\mathscr{P})$ with respect to this equivalence. The relation $\prec$ is determined as follows. Let $Q_1, Q_2 \in \mathscr{B}(\mathscr{P})$ and $\tilde{Q}_1, \tilde{Q}_2$ be the corresponding equivalence classes. Suppose $\tilde{Q}_1 \prec \tilde{Q}_2$ if $Q_1 \square Q_2$. Thus the study of the relation $\square$ is reduced to one of the relation $\approx$ on $\mathscr{B}(\mathscr{P})$ and $\prec$ on $\tilde{\mathscr{B}}(\mathscr{P})$. We shall call the description of the relation supplementation problem. Its solution enables us to determine which sets are "more complicated" and which are "simpler" by completing them in the same manner, and which sets have similar behaviour under these conditions. In considering the problem it is natural to use the properties of the inclusion lattice formed by $\mathscr{B}(\mathscr{P})$. Let us first recall some facts concerning the completeness problem. From [1] we know that its solution may be obtained by constructing a so-called criterial system. Namely, $\theta \subseteq \mathscr{B}(\mathscr{P})$ is a criterial system, if for any set $Q \subseteq P$ its completeness is equivalent to non-entry of $Q$ as a subset in every set from $\theta$. Criterial systems are known [1] to exist for $I(\emptyset) \neq P$ among them we may choose such a system which can be represented as $\theta_1(\mathscr{P}) \cup \theta_2(\mathscr{P})$ where $\theta_1(\mathscr{P})$ is the set of all precomplete classes in $\mathscr{P}$ and $\theta_2(\mathscr{P})$ is the set of all elements $Q$, $Q \neq P$, from $\mathscr{B}(\mathscr{P})$ such that no precomplete class has $Q$ as a subset. Let us remind that $Q \in \mathscr{B}(\mathscr{P})$ is called a precomplete class if $I(Q) \neq P$ but $I(Q \cup \{a\}) = P$ holds for any $a \in P \setminus Q$. According to [1] we have in the general case that $\theta_1(\mathscr{P}) \neq \emptyset$ and $\theta_2(\mathscr{P}) = \emptyset$, $\theta_1(\mathscr{P}) \neq \emptyset$ and $\theta_2(\mathscr{P}) \neq \emptyset$, $\theta_1(\mathscr{P}) = \emptyset$ and $\theta_2(\mathscr{P}) \neq \emptyset$, $\theta_1(\mathscr{P}) = \emptyset$ and $\theta_2(\mathscr{P}) = \emptyset$. The last situation holds when $I(\emptyset) = P$. Further we shall assume that this condition is not fulfilled for $\mathscr{P}$ and the pair $\mathscr{P}$ for which the additional condition $\theta_2(\mathscr{P}) = \emptyset$ holds is correct.

**Theorem 1.** If $Q_1, Q_2 \in \mathscr{B}(\mathscr{P})$ then the relation $Q_1 \square Q_2$ holds if and only if either $Q_2 = P$ or if $Q_2 \neq P$ then $I(Q_1 \cup Q') \in \mathscr{B}(\mathscr{P}) \setminus \{P\}$ is valid for any $Q' \in \mathscr{B}(\mathscr{P}) \setminus \{P\}$ such that $Q' \supseteq Q_2$.

*Proof.* If $Q_2 = P$ or $Q_2 \neq P$ and the above conditions are fulfilled, then $Q_1 \square Q_2$ is obvious. Let $Q_2 \neq P$, $Q_1 \square Q_2$ and $Q_2 \subseteq Q'$ hold for any $Q' \in \mathscr{B}(\mathscr{P}) \setminus \{P\}$. Consider the set $I(Q_1 \cup Q')$. If $I(Q_1 \cup Q') \neq P$ then our statement is valid. If $I(Q_1 \cup Q') = {} = P$ then by virtue of the relation $Q_1 \square Q_2$ there must be $I(Q_2 \cup Q') = P$ but $I(Q_2 \cup Q') = Q'$, $Q' \neq P$, what disproves the assumed equality $I(Q_1 \cup Q') = P$. The theorem is proved.

**Corollary 1.1.** Different precomplete classes from $\mathscr{B}(\mathscr{P})$ are not comparable with respect to $\square$.

The theorem, if symmetrically used, gives a criterion of equivalence of two sets. It also demonstrates an obvious sufficient condition of equivalence of two sets. Let $Q_1, Q_2 \in \mathscr{B}(\mathscr{P})$, $Q_1 \subseteq Q_2$ and for any $Q_3 \in \mathscr{B}(\mathscr{P})$ such that $Q_1 \subseteq Q_3$ if $Q_3 \nsubseteq Q_2$ then $Q_3 \supseteq Q_2$. In this case we shall write $Q_1 \boxplus Q_2$. It is clear that this relation will hold for $Q_1 \square Q_2$. The converse does not, generally speaking.

**Proposition 1.** If $Q_1, Q_2 \in \mathscr{B}(\mathscr{P})$ and $Q_1 \boxplus Q_2$, $Q_2 \neq P$ then $Q_1 \approx Q_2$.

*Proof.* Since by definition $Q_1 \subseteq Q_2$ then $Q_1 \square Q_2$. Now we prove that $Q_2 \square Q_1$. Let $Q' \in \mathscr{B}(\mathscr{P}) \setminus \{P\}$ and $Q' \supseteq Q_1$ then, because of $Q_1 \boxplus Q_2$ we have either $Q' \subseteq Q_2$ or $Q' \supseteq Q_2$. In the first case we have $I(Q_2 \cup Q') = Q_2$ in the second case we have

$I(Q_2 \cup Q')=Q'$ i.e. $I(Q_2 \cup Q') \neq P$. Hence by theorem 1 we arrive at $Q_2 \square Q_1$. Consequently, $Q_1 \approx Q_2$ the proposition is proved.

Moreover the class of equivalence with respect to $\approx$ is also characterized by the following obvious proposition.

**Proposition 2.** If $Q_1, Q_2 \in \mathscr{B}(\mathscr{P})$ and $Q_1 \approx Q_2$ then $I(Q_1 \cup Q_2) \approx Q_1$.

Theorem 1 also permits to describe the relation $\square$ in a different form using the notion of type of a set. Let $Q \in \mathscr{B}(\mathscr{P})$ and $\tau_{\mathscr{P}}(Q)$ be a set of all precomplete classes each of which contains $Q$ as a subset; $\tau_{\mathscr{P}}(Q)$ will be called the type of the set $Q$. Obviously, $\tau_{\mathscr{P}}(\emptyset)=\theta_1(\mathscr{P})$ and $\tau_{\mathscr{P}}(P)=\emptyset$.
We have

**Theorem 2.** If $Q_1, Q_2 \in \mathscr{B}(\mathscr{P})$ then for $Q_1 \square Q_2$ we have $\tau_{\mathscr{P}}(Q_1) \supseteq \tau_{\mathscr{P}}(Q_2)$.

*Proof.* If $Q_2=P$ or $\tau_{\mathscr{P}}(Q_2)=\emptyset$ then the statement is valid. Now let $Q_2 \neq P$ and $\tau_{\mathscr{P}}(Q_2) \neq \emptyset$. Consider $\pi \in \tau_{\mathscr{P}}(Q_2)$. Since $Q_1 \square Q_2$ and $\pi \supseteq Q_2$ we have by theorem 1 that $I(Q_1 \cup \pi) \in \mathscr{B}(\mathscr{P}) \setminus \{P\}$. Hence by virtue of the precompleteness of $\pi$ we have $Q_1 \subseteq \pi$ and thereby $\tau_{\mathscr{P}}(Q_1) \supseteq \tau_{\mathscr{P}}(Q_2)$. The theorem is proved.

**Corollary 2.1.** If $Q_1$, $Q_2 \in \mathscr{B}(\mathscr{P})$ and $Q_1 \approx Q_2$ then $\tau_{\mathscr{P}}(Q_1)=\tau_{\mathscr{P}}(Q_2)$.

Note that the statements reverse to theorem 2 as well as to corollary 2.1 are wrong, generally speaking. They may not hold even for $\mathscr{P}$ such that $\theta_1(\mathscr{P})=\emptyset$. However for correct $\mathscr{P}$ we have

**Theorem 3.** If $Q_1$, $Q_2 \in \mathscr{B}(\mathscr{P})$ and $\mathscr{P}$ is a correct system, then $Q_1 \square Q_2$ if and only if $\tau_{\mathscr{P}}(Q_1) \supseteq \tau_{\mathscr{P}}(Q_2)$.

*Proof.* The "only if" part follows from theorem 2. Now let $\tau_{\mathscr{P}}(Q_1) \supseteq \tau_{\mathscr{P}}(Q_2)$ hold. We shall prove that $Q_1 \square Q_2$. If $\tau_{\mathscr{P}}(Q_2)=\emptyset$ we have in view of $\mathscr{P}$ being correct that $Q_2=P$ and, therefore, $Q_1 \square Q_2$. Let $\tau_{\mathscr{P}}(Q_2) \neq \emptyset$ and suppose that the relation $Q_1 \square Q_2$ does not hold. By theorem 1 it means that for some $Q' \in \mathscr{B}(\mathscr{P}) \setminus \{P\}$ we have $Q' \supseteq Q_2$ and $I(Q_1 \cup Q')=P$. Consider $\tau_{\mathscr{P}}(Q')$. It is obvious that $\tau_{\mathscr{P}}(Q') \subseteq \tau_{\mathscr{P}}(Q_2)$ and in view of $\mathscr{P}$ being correct we have $\tau_{\mathscr{P}}(Q') \neq \emptyset$. Let $\pi \in \tau_{\mathscr{P}}(Q')$. Since $\pi \supseteq Q'$ we get $I(Q_1 \cup \pi)=P$.
It follows that $\pi' \notin \tau_{\mathscr{P}}(Q')$ for any $\pi' \in \tau_{\mathscr{P}}(Q_1)$ what is contrary to the relations $\tau_{\mathscr{P}}(Q_1) \supseteq \tau_{\mathscr{P}}(Q')$ and $\tau_{\mathscr{P}}(Q') \neq \emptyset$. So, the assumption concerning the incorrectness of the relation $Q_1 \square Q_2$ is false. The theorem is proved.

**Corollary 3.1.** If $Q_1$, $Q_2 \in \mathscr{B}(\mathscr{P})$ and $\mathscr{P}$ is a correct system, then $Q_1 \approx Q_2$ if and only if $\tau_{\mathscr{P}}(Q_1)=\tau_{\mathscr{P}}(Q_2)$.

Theorem 3 and corollary 3.1 permit to describe the relation $\square$ when $\mathscr{P}$ is a correct system. If $\tau(Q)$ is known for $Q \in \mathscr{B}(\mathscr{P})$ then the class of all sets equivalent to the set $Q$ consists of all $Q'$ such that $\tau(Q')=\tau(Q)$ i.e. this class is uniquely determined by the value of $\tau$. We denote it by $K_\tau$. Then the relation $K_\tau \prec K_{\tau'}$ on $\tilde{\mathscr{B}}(\mathscr{P})$ is equivalent to $\tau' \subseteq \tau$.

Let $|A|$ be the cardinality of the set $A$. Consider $|\tilde{\mathscr{B}}(\mathscr{P})|$. It characterizes the variety of systems in the supplementation problem. By corollary 1.1 we prove the validity of the following statement.

**Proposition 3.** We have

$$|\theta_1(\mathscr{P})| \leqq |\tilde{\mathscr{B}}(\mathscr{P})| \leqq 2^{|P|}.$$

According to [1] $|\theta_1(\mathscr{P})|$ may take any value $\varkappa \leqq 2^{|P|}$ depending on $P$ and $I$ thus $|\tilde{\mathscr{B}}(\mathscr{P})|$ is majorized from below by the same values. In particular, the equiality $|\tilde{\mathscr{B}}(\mathscr{P})| = 2^{|P|}$ is possible which implies extremely great variations of cardinality of the class $\tilde{\mathscr{B}}(\mathscr{P})$.

## § 2. Supplementation problem for functional systems

A functional system (f.s.) is such a system $\mathscr{P}=(P, I)$ in which the set $P$ is a set of functions, and $I$ is the closure operator given by the automation. If $P$ consists of functions defined on the collections from the subsets of a natural series with values of the very functions taken from the natural series, then the f.s. is called a truth functional system (t.f.s.). If $P$ consists of lexicographic functions, then the functional system is called a sequential functional system (s.f.s.). Typical examples of f.s. are many-valued logics (examples of t.f.s.) and algebras of automata (examples of s.f.s.). An important class of t.f.s. is formed by finite t.f.s; (f.t.f.s.). They are defined as follows. Let $E_k=\{0, 1, \dots, k-1\}$, $k>1$, $U=\{u_1, u_2, \dots\}$ be the alphabet of the variables $u_m$ whose values are the elements from $E_k$, let $P_k$ be a set of all functions $f(u_{i_1}, \dots, u_{i_n})$ with values from $E_k$, $M_k \subseteqq P_k$ and let $I_\Phi$ be a special closure operator called a finite automaton-given operator. $I_\Phi$ is specified by a collection $\Omega$ of finite-place operations $\omega$ given by the automaton over the elements from $M_k$ which falls into two parts $\Omega_1$ and $\Omega_2$. The collection $\Omega_1$ gives the closure operator $I_{\mathscr{S}P}$ corresponding to the closure of the subsets $M' \subseteqq M_k$ with respeet to taking all the superpositions of functions from $M'$. The collection $\Omega_2$ is finite. The system $\mathscr{M}_k=(M_k, I_\Phi)$ is called a finite t.f.s. Consider the partial order $\prec$ on the factor set $\tilde{\mathscr{B}}(\mathscr{M}_k)$ in the form of an oriented graph. The elements of $\tilde{\mathscr{B}}(\mathscr{M}_k)$ will be points in space. Any two points $a$ and $b$ are connected with an oriented edge from $a$ to $b$ if $b \prec a$ and there is no point $c$ distinct from $a$ and $b$ such that $b \prec c \prec a$. The graph obtained is denoted by $G(\mathscr{M}_k)$ and the number of its vertices is denoted by $|G(\mathscr{M}_k)|$. It is to the description of this graph that the supplementation problem is reduced for f.t.s.f. We introduce some notions to characterize f.t.f.s. $\mathscr{M}_k$. Let $M \subseteqq P_k$ and $M^{(n)}$ be a set of all functions from $M$ which depend only on variables from the alphabet $U_n=\{u_1, u_2, \dots; u_n\}$, let $p_k^{(n)}$ be a number of elements in $P_k^{(n)}$. It is clear that $p_k^{(n)} = \sum_{i=1}^{n} C_n^i \cdot k^{k^i}$. Let $S(P_k^{(n)})$ be a set of functions from $P_k^{(n)}$ each of them is equal to $u_i$, $i=1, 2, \dots, n$ for some $i$. For the finite set $M' \subseteqq P_k$; we shall use $m(M')$ for the greatest index of the variable of the functions from $M'$. Let $\Omega_2=\{\omega_1, \omega_2, \dots, \omega_r\}$ hold in the f.t.f.s. $\mathscr{M}_k$. Let the value of $\omega_j(f_1, f_2, \dots, f_{s_j})$ be defined and

$$m_j = m(\{f_1, f_2, \dots, f_{s_j}, \omega_j(f_1, f_2, \dots, f_{s_j})\}), \quad j = 1, 2, \dots, r.$$

Since $m_j$ depends only on $\omega_j$ then we can introduce the notation $|\omega_j|$ for $m_j$. Let $m(\Omega_2) = \max\{|\omega_1|, |\omega_2|, \dots, |\omega_r|\}$. If $\mathscr{M}_k$ is finitely generated and $M_k^*$ is a set of finite $M' \subseteqq M_k$ such that $I_\Phi(M')=M_k$ then let $m_0 = \inf_{M' \in M_k^*} m(M')$ and $s=\max(m_0, m(\Omega_2))$.

Let the nonempty set $M \subseteq P_k^{(s)} \cap M_k$ be called $R$-set if $I_\Phi(M) \cap P_k^{(s)} = M$ and $M \neq P_k^{(s)} \cap M_k$. We denote it by $R$. Let $\mathscr{R} = (R \cup S(P_k^{(s)}), R)$. We shall say that the function $f(x_1, x_2, \dots, x_n)$ from $P_k$ retains $\mathscr{R}$ if $f(g_1, g_2, \dots, g_n) \in R$ holds for any collection of functions $g_1, g_2, \dots, g_n$ from $R \cup S(P_k^{(s)})$. The class of all functions from $M_k$ with $\mathscr{R}$ will be denoted by $\cup(\mathscr{R})$. We will call the $R$-set $R$ maximal unless there exists an $R$-set $R'$ such that $\cup(R') \supseteq \cup(R)$, $R \neq R'$. Let, for f.t.f.s. $\mathscr{M}_k$, $\mathbf{R}$ be the set of all maximal $R$-sets, and let $\mathbf{R}$ be a set of all pairs $\mathscr{R}$ for which $R \in \mathbf{R}$.

**Theorem 4.** If an f.t.f.s. $\mathscr{M}_k = (M_k, I_\Phi)$ is finitely generated, then the following statements are true:

1) $$|G(\mathscr{M}_k)| \leqq 2^{2^{p_k(s)}}$$

holds for the graph $G(\mathscr{M}_k)$;

2) the graph $G(\mathscr{M}_k)$ can be constructed effectively.

*Proof.* We start from some given finite set $M \subseteq M_k$ such that $I_\Phi(M) = M_k$. According to [1] the finitely generated f.t.f.s. $\mathscr{M}_k$ is correct, $s$ and $\mathbf{R}$ can be found by $M$ effectively, $\cup(\mathbf{R})$ coincides with the set of all precomplete classes in $\mathscr{M}_k$ and $|\cup(\mathbf{R})| \leqq 2^{p_k^{(s)}}$. Let $\mathbf{R} = \{\mathscr{R}_1, \mathscr{R}_2, \dots, \mathscr{R}_l\}$ and $\cup_i = \cup(\mathscr{R}_i)$, $i = 1, 2, \dots, l$. Consider the set $\Sigma$ whose elements are expressions of the form $\mathfrak{A} = \mathop{\&}\limits_{j=1}^{t} \cup_{i_j}$ where $i_{j'} < i_{j''}$ and $\cup_{i_{j'}} \neq \cup_{i_{j''}}$ for $j' < j''$, $t \geqq 0$. The formula of $\mathfrak{A}$ is interpreted as a set of functions which is equal to the intersection of the sets $\cup_{i_j}$ which form $\mathfrak{A}$. This set is denoted by $\hat{\mathfrak{A}}$. For $t = 0$ we have an empty conjunction $\mathfrak{A}_0$ which by definition generates the whole set $M_k$. Let $\tau(\mathfrak{A}) = \tau(\hat{\mathfrak{A}})$. We introduce a partial preorder relation on $\Sigma$ putting $\mathfrak{A} \lesssim \mathfrak{B}$ if and only if $\tau(\mathfrak{A}) \supseteq \tau(\mathfrak{B})$, $\tau(\mathscr{M}_k)$ being an empty set by definition. It is obvious that this preorder coincides on $\Sigma$ (by the above interpretation) with the relation $\Box$, and is reduced to the equivalence relation $\approx$ and the relation of partial order $\prec$ on the factor set $\tilde{\Sigma}$ of the set $\Sigma$ with respect to this equivalence. Represent this partial order as a graph $\mathfrak{G}(\mathscr{M}_k)$ in the same way as it was done in constructing the graph $G(\mathscr{M}_k)$. Now establish a connection between these graphs. We see that for any $\tau \subseteq \theta_1(\mathscr{M}_k)$ the following holds: if, in the graph $G(\mathscr{M}_k)$ and $\mathfrak{G}(\mathscr{M}_k)$ there exists a vertex which denotes a class of sets of a given type $\tau$ (a vertex of type $\tau$), then this vertex is unique, and both graphs have such vertices simultaneously. Let us establish the correspondence between the vertices of the graphs $G(\mathscr{M}_k)$ and $\mathfrak{G}(\mathscr{M}_k)$ by the property of coincidence of their types $\tau$. Since it is a one-to-one correspondence, then $l \leqq |G(\mathscr{M}_k)| \leqq 2^{2^{p_k(s)}}$ holds and thereby relation 1) is established. Now, if we extend the correspondence between the graphs to the correspondence between the edges connecting the corresponding vertices, we can see that these graphs are isomorphic, so, to establish property 2) it is suffices to show that the graph $\mathfrak{G}(\mathscr{M}_k)$ can effectively be constructed and the types of its vertices can effectively be determined. For this purpose we first establish that the relations: (1) $\hat{\mathfrak{A}}_1 \subseteq \hat{\mathfrak{A}}_2$ and (2) $\hat{\mathfrak{A}}_1^{(q)} \subseteq \hat{\mathfrak{A}}_2^{(q)}$ are equivalent if $\mathfrak{A}_1$; $\mathfrak{A}_2 \in \Sigma$ and $q = k^{k^s \cdot 2^{p_k^{(s)}}}$. This results from relation (1) being a consequence of (2). Now let us prove it. Suppose $M \subseteq P_k$. Let $\bar{M}^{(n)}$ denote a set of all functions from $M^{(n)}$ depending exactly on all the variables from $U_n$. We construct a pair $\bar{\mathscr{R}}_i = (\bar{R}_i^{(s)} \cup \bar{S}(P_k^{(s)})^{(s)}, \bar{R}_i^{(s)})$ corresponding to $\mathscr{R}_i = (R_i \cup S(P_k^{(s)}), R_i)$ and introduce a set $\cup(\bar{\mathscr{R}}_i)$ by analogy with the set $\cup(\mathscr{R}_i)$. We see that $\cup(\mathscr{R}_i)=\cup(\bar{\mathscr{R}}_i)$ holds. For a set $\bar{R}_i^{(s)}\cup\bar{S}(P_k^{(s)})^{(s)}$ we construct a matrix pair $T_i$ as follows. Let all $h_1, h_2, \dots, h_r$ be the functions from $\bar{R}_i^{(s)}$ and all $\lambda_1, \lambda_2, \dots, \lambda_s$ be the functions from $\bar{S}(P_k^{(s)})^{(s)}$. Their choice can be represented by the summary table

| $u_1$ | $u_2 \dots u_s$ | $h_1$ | $h_2 \dots h_r$ | $\lambda_1$ | $\lambda_2 \dots \lambda_s$ |
|---|---|---|---|---|---|
| $a_{11}$ | $a_{12} \dots a_{1s}$ | $b_{11}$ | $b_{12} \dots b_{1r}$ | $c_{11}$ | $c_{12} \dots c_{1s}$ |
| | $\dots$ | | $\dots$ | | $\dots$ |
| $a_{v1}$ | $a_{v2} \dots a_{vs}$ | $b_{v1}$ | $b_{v2} \dots b_{vr}$ | $c_{v1}$ | $c_{v2} \dots c_{vs}$ |
| | $\dots$ | | $\dots$ | | $\dots$ |
| $a_{k^s1}$ | $a_{k^s2} \dots a_{k^ss}$ | $b_{k^s1}$ | $b_{k^s2} \dots b_{k^sr}$ | $c_{k^s1}$ | $c_{k^s2} \dots c_{k^ss}$ |

Let us single out two of its matrices

$$T_i' = \begin{pmatrix} b_{11} & b_{12} \dots b_{1r} & c_{11} & c_{12} \dots c_{1s} \\ & \dots & & \dots \\ b_{k^s1} & b_{k^s2} \dots b_{k^sr} & c_{k^s1} & c_{k^s2} \dots c_{k^ss} \end{pmatrix},$$

$$T_i'' = \begin{pmatrix} b_{11} & b_{12} \dots b_{kr} \\ & \dots \\ b_{k^s1} & b_{k^s2} \dots b_{k^sr} \end{pmatrix}$$

and consider the pair $T_i=(T_i', T_i'')$. We shall say that $f(x_1, x_2, \dots, x_m)$ from $P_k$ retains $T_i$ if for any matrix

$$T = \begin{pmatrix} d_{11} & d_{12} \dots d_{1m} \\ & \dots \\ d_{k^s1} & d_{k^s2} \dots d_{k^sm} \end{pmatrix}$$

whose columns are all taken from the matrix $T_i'$ the column

$$f(T) = \begin{pmatrix} f(d_{11}, d_{12}, \dots, d_{1m}) \\ \dots \\ f(d_{k^s1}, d_{k^s2}, \dots, d_{k^sm}) \end{pmatrix}$$

will be the column of the matrix $T_i''$. Let $\cup(T_i)$ be the set of all functions from $M_k$ retaining $T$. We see that $\cup(T_i)=\cup(\bar{\mathscr{R}}_i)$. Now we introduce an operation with respect to the matrices $A$ and $B$. $C=A\cdot B$, if $C$ consists exactly of all such columns $\gamma$ which result from placing the column $\alpha$ of $A$ on top of the column $\beta$ of $B$.

We denote by $A^r$ the result of multiplication of $A$ by itself $r$ times. For the expression $\mathfrak{A}=\mathop{\&}\limits_{j=1}^{t}\cup_{i_j}$, $t>0$ we shall construct a pair $T(\mathfrak{A})=(T('\mathfrak{A}), T''(\mathfrak{A}))$ where

$$T'(\mathfrak{A}) = T_{i_1}'^{r_1}\cdot T_{i_2}'^{r_2}\cdot \dots \cdot T_{i_t}'^{r_t},$$

$$T''(\mathfrak{A}) = T_{i_1}''^{r_1}\cdot T_{i_2}''^{r_2}\cdot \dots \cdot T_{i_t}''^{r_t}, \quad \sum_{w=1}^{t} r_w = 2^{p_k(s)}$$

and $r_w > 0$ for all $w$. By introducing a set $\mathsf{U}(T(\mathfrak{A}))$ by analogy with the set $\mathsf{U}(T_i)$ we can see that $\mathsf{U}(T(\mathfrak{A})) = \bigcap_{j=1}^{t} \mathsf{U}(T_{i_j})$ and hence $\mathsf{U}(T(\mathfrak{A})) = \bar{\mathfrak{A}}$.

As established in [1] the relation $\mathsf{U}_i^{(q)} \neq M_k^{(q)}$ holds. Therefore in the course of proving relation (1) to be a consequence of (2) we may assume that $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are distinct from $\mathfrak{A}_0$.

Now suppose that relation (2) holds whereas relation (1) does not. It means that in $\hat{\mathfrak{A}}_1$ there is a function $f$ such that $f \notin \hat{\mathfrak{A}}_2$. It is clear that it must depend on $v$ variables and $v > q$. Notice that by construction the matrices $T'(\mathfrak{A})$ and $T''(\mathfrak{A})$ have columns of the same length equal to $k^s \cdot 2^{p_k(s)}$ for any $\mathfrak{A}$ from $\Sigma$, $\mathfrak{A} \neq \mathfrak{A}_0$. Thus, these matrices have not more than $q$ different columns. By assumption $f$ does not retain $T(\mathfrak{A}_2)$. This means that there is a matrix $T$ consisting of the columns of the matrix $T'(\mathfrak{A}_2)$ such that $f(T) \notin T''(\mathfrak{A}_2)$. The matrix $T$ has not more than $q$ different columns, so, we may assume without loss of generality that it is formed by successive groups of equal columns. In accordance with these groups we divide the variables of $f$ into the same groups and in $f$ replace every variable of the $j$-th group by the variable $x_j$, $1 \leqq j \leqq q$. As a result we get $f'$ from $\hat{\mathfrak{A}}_1^{(q)}$ not retaining $T(\mathfrak{A}_2)$ either, what is at variance with relation (2). Thus, relation (1) is a consequence of relation (2).

Let now $\mathfrak{A} \in \Sigma \diagdown \{\mathfrak{A}_0\}$. Construct $T(\mathfrak{A})$. According to [1] we can effectively construct the set $M_k^{(q)}$ and, consequently, the set $\mathsf{U}(T(\mathfrak{A}))^{(q)} = (\hat{\mathfrak{A}})^{(q)}$. Since (1) is a consequence of (2) we can effectively define all precomplete classes $\mathsf{U}_i$ such that $\mathsf{U}_i \supseteqq \hat{\mathfrak{A}}^{(q)}$ and thereby estimate $\tau(\mathfrak{A})$. Knowing these values and the value of $\tau(\mathfrak{A}_0)$ we can construct the graph $\mathfrak{G}(\mathscr{M}_k)$ and, consequently $G(\mathscr{M}_k)$. The theorem is proved.

It is known from [1] that f.t.f.s. $\mathscr{P}_k = (P_k, I_{sp})$ is finitely generated and for the number $|\theta_1(\mathscr{P}_k)|$ of precomplete classes in it we have

$$|\theta_1(\mathscr{P}_k)| \sim \delta(k) \cdot k \cdot 2^{C_{k-1}^{[k-1/2]}} \quad \text{for} \quad k \to \infty$$

where $\delta(k) = 2$ if $k$ is even, and $\delta(k) = 1$ if $k$ is odd. By this we arrive at

**Corollary 4.1.** The graph $G(\mathscr{P}_k)$ can effectively be constructed and

$$\delta(k) \cdot k \cdot 2^{C_{k-1}^{[k-1/2]}} \lesssim |G(\mathscr{P}_k)| \lesssim 2^{\delta(k) \cdot k \cdot 2^{C_{k-1}^{[k-1/2]}}} \quad \text{for} \quad k \to \infty.$$

## § 3. Supplementation problem for Post classes

Let us consider the supplementation problem for the f.t.f.s. $\mathscr{M}_2 = (M_2, I_{sp})$ where $M_2 \subseteqq P_2$. E. Post is known to have described all the closed classes $M_2$ [2]. He established that the set of these classes is countable and that they are finite-generated. He constructed an inclusion lattice formed by these classes. The set of the classes in question is reduced to the following: $C_i$, $A_i$, $D_j$, $L_{k'}$, $O_l$, $S_r$, $P'_r$, $F_s^n$, $F_s^\infty$ where $i = 1, 2, 3, 4$, $j = 1, 2, 3$, $k' = 1, 2, 3, 4, 5$, $l = 1, 2, \ldots, 9$, $r = 1, 3, 5, 6$, $s = 1, 2, \ldots, 8$, $n = 2, 3, \ldots$.

The class $C_1$ contains all the functions of the algebra of logic and coincides with $P_2$. $C_2$ consists of all $f(x_1, x_2, \ldots, x_n)$ such that $C_3$ consists of all $f(x_1, x_2, \ldots, x_n)$ such that $C_4 = C_2 \cap C_3$. The class $A_1$ comprises all monotone functions; $A_2 = C_2 \cap A_1$; $A_3 = C_3 \cap A_1$; $A_4 = A_2 \cap A_3$. The class $D_3$ consists of all functions

$f(x_1, x_2, \ldots, x_n)$ such that $f(x_1, x_2, \ldots, x_n) = \bar{f}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$. The function $f^*(x_1, x_2, \ldots, x_n) = \bar{f}(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ being called dual with respect to $f$ and the set $\mathfrak{M}^*$ consisting of all functions dual with respect to the functions of $\mathfrak{M}$ is called dual with respect to $\mathfrak{M}$; the class $\mathfrak{M}$ is called self-dual if $\mathfrak{M} = \mathfrak{M}^*$; $D_1 = C_4 \cap D_3$; $D_2 = A_1 \cap D_3$.

The class $L_1$ consists of all functions $f(x_1, x_2, \ldots, x_n)$ such that $f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} x_i + d \pmod 2$; $L_2 = C_2 \cap L_1$; $L_3 = C_3 \cap L_1$; $L_4 = L_2 \cap L_3$; $L_5 = D_3 \cap L_1$. $O_9$ comprises all the functions essentially depending on not more than one variable; $O_8 = A_1 \cap O_9$; $O_4 = D_3 \cap O_9$; $O_5 = C_2 \cap O_9$; $O_6 = C_3 \cap O_9$; $O_1 = O_5 \cap O_6$; $O_7$ consists of all constant functions; $O_2 = O_5 \cap O_7$; $O_3 = O_6 \cap O_7$. The class $S_6$ consists of all functions $f(x_1, x_2, \ldots, x_n) = x_1 \vee x_2 \vee \ldots \vee x_n$ and all constants; $S_3 = C_2 \cap S_6$; $S_5 = C_3 \cap S_6$; $S_1 = S_3 \cap S_5$. The class $P_6'$ consists of all functions $f(x_1, x_2, \ldots, x_n) = x_1 \& x_2 \& \ldots \& x_n$ and all constants; $P_5' = C_2 \cap P_6'$; $P_3' = C_3 \cap P_6'$; $P_1' = P_5' \cap P_3'$. A function is said to satisfy the condition $a^n$, $n \geqq 2$ if any $n$ collections in which it is equal to 0 have a common coordinate equal to 0. Analogously, with the replacement of 0 by 1 we introduce the condition $A^n$. The class $F_4^n$ consists of all functions with property $a^n$; $F_1^n = C_4 \cap F_4^n$; $F_3^n = A_1 \cap F_4^n$; $F_2^n = F_1^n \cap F_3^n$; $F_8^n$ consists of all functions with property $A^n$; $F_5^n = C_4 \cap F_8^n$; $F_7^n = A_3 \cap F_8^n$; $F_6^n = F_5^n \cap F_7^n$. A function is said to satisfy the condition $a^\infty$ if all the collections in which it is equal to zero have a common coordinate equal to zero. Again by replacing 0 by 1 we introduce the property $A^\infty$. The class $F_4^\infty$ consists of all functions with property $a^\infty$; $F_1^\infty = C_4 \cap F_4^\infty$; $F_3^\infty = A_1 \cap F_4^\infty$; $F_2^\infty = F_1^\infty \cap F_3^\infty$; $F_8^\infty$ consists of all functions with property $A^\infty$; $F_5^\infty = C_4 \cap F_8^\infty$; $F_7^\infty = A_3 \cap F_8^\infty$; $F_6^\infty = F_5^\infty \cap F_7^\infty$. The above inclusion lattice formed by these classes is given in Fig. 1. In this figure classes are represented by points. Two points are connected by an arc if the underlying point denotes a class contained immediately in the top class (i.e. there are no intermediate classes between them). The lattice has an axis of symmetry. Self-dual classes are represented by points on the axis; classes dual with respect to each other are represented symmetrically with respect to the axis. The self-dual classes are $C_1, C_4, A_1, A_4, D_1, D_2, D_3, L_1, L_4, L_5, O_1, O_4, O_7, O_8, O_9$. For other classes we have

$$C_2 = C_3^*, \quad A_2 = A_3^*, \quad L_2 = L_3^*, \quad P_1' = S_1^*, \quad P_3' = S_3^*, \quad P_5' = S_5^*,$$
$$P_6' = S_6^*, \quad O_5 = O_6^*, \quad O_2 = O_3^*, \quad F_1^n = (F_5^n)^*, \quad F_2^n = (F_6^n)^*,$$
$$F_3^n = (F_7^n)^*, \quad F_4^n = (F_8^n)^*, \quad F_1^\infty = (F_5^\infty)^*, \quad F_2^\infty = (F_6^\infty)^*,$$
$$F_3^\infty = (F_7^\infty)^*, \quad F_4^\infty = (F_8^\infty)^*.$$

Thus, the supplementation problem for any f.t.f.s. $\mathcal{M}_2$ is reduced to considering such f.t.f.s. $\mathcal{M}_2$ for which $M_2$ coincides with one of the classes of the set

$$Z = \{C_1, C_3, C_4, A_1, A_3, A_4, D_1, D_2, D_3, L_1, L_3, L_4, L_5,$$
$$F_5^n, F_6^n, F_7^n, F_8^n, F_5^\infty, F_6^\infty, F_7^\infty, F_8^\infty, P_1', P_3', P_5', P_6', O_1, O_3, O_4, O_6, O_7, O_8, O_9\}.$$

By virtue of theorem 3 and corollary 3.1, the solution of the supplementation problem, under condition that $M_2 \in Z$, is as follows. By the aid of a Post lattice we calculate $\tau_{\mathcal{M}}(M')$ for every closed class $M' \subseteqq M_2$. The set of all classes of the same type of $\tau$ is declared the class $K_\tau$. On the set of these classes we introduce the relation of

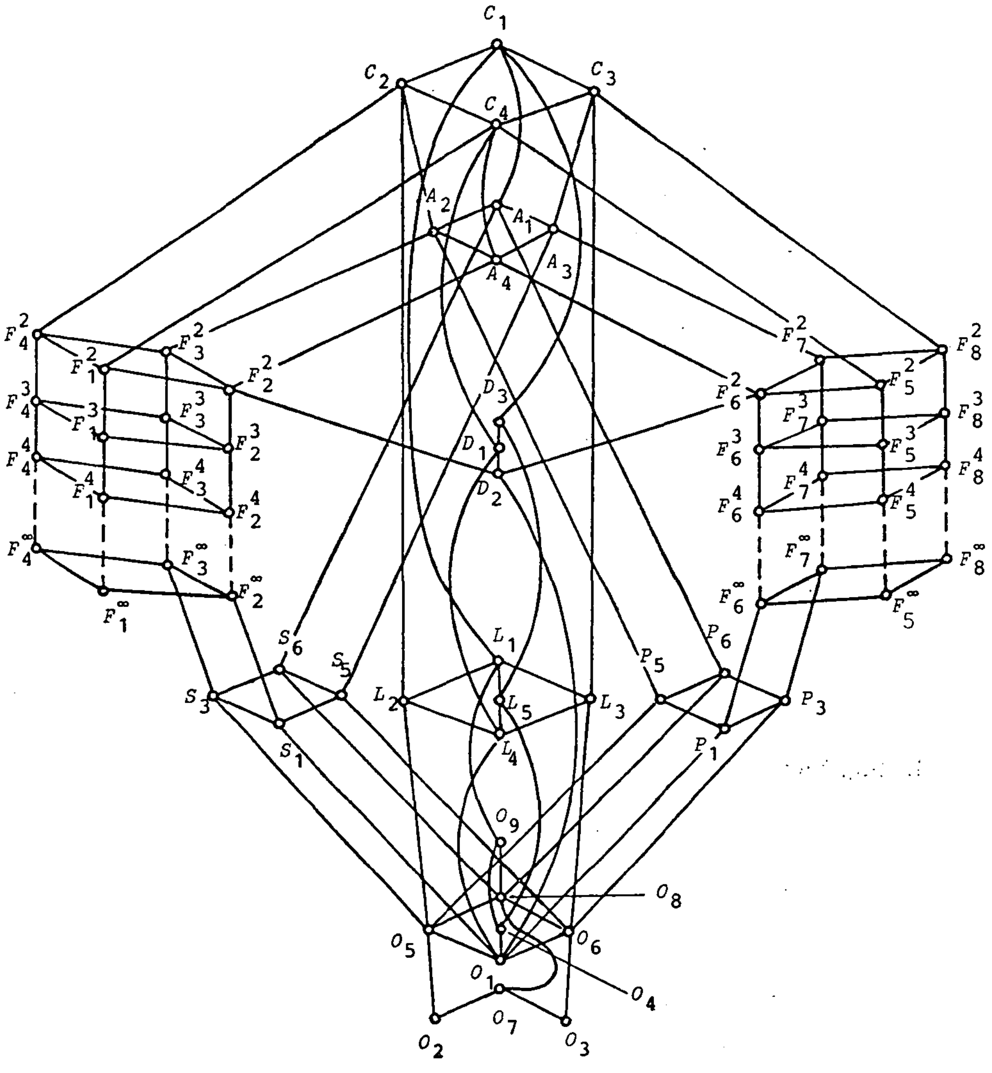


*Fig. 1*

partial order which coincides with the relation of inclusion for types of classes. As already mentioned the attributing of closed classes to a single class $K_\tau$ corresponds to $\approx$. In this way we construct the relation $\square$ over $\mathscr{B}(\mathscr{M}_2)$. Now we give the Post lattice and the results of calculations for the graph $G(\mathscr{M}_2)$. There turned out to be eleven graphs of this kind accurate to isomorphism. They are given in Fig. 2—8 with the edges oriented from top to bottom. Now let us describe the values of the parameters $K_i$ for different graphs and classes $M_2$.

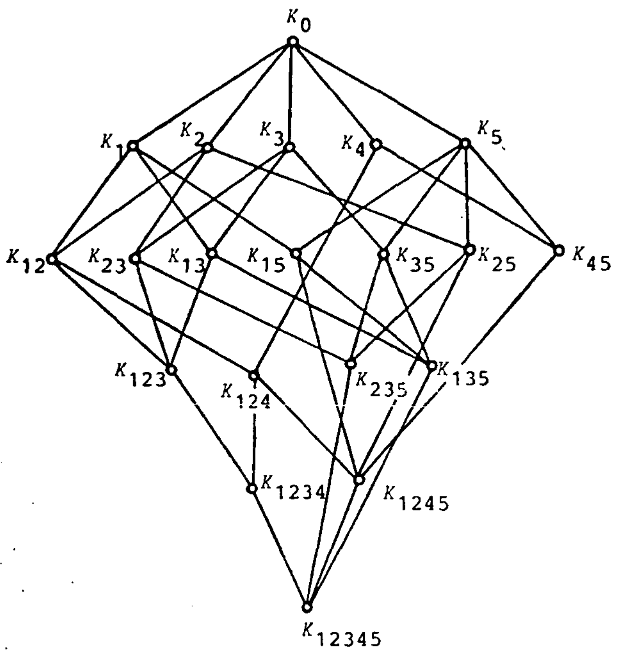


*Fig. 2*

In Fig. 2 we have the graph $G(C_1)$. Here

$$K_0 = \{C_1\}, \quad K_1 = \{C_2, F_4^2, F_4^3, \dots, F_4^n, \dots, F_4^\infty\},$$

$$K_2 = \{C_3, F_8^2, F_8^3, \dots, F_8^n, \dots, F_8^\infty\}, \quad K_3 = \{A_1, P_6', S_6'\},$$

$$K_4 = \{D_3\}, \quad K_5 = \{L_1, O_9\},$$

$$K_{12} = \{C_4, F_5^2, F_5^3, \dots, F_5^n, \dots, F_5^\infty, F_1^2, F_1^3, \dots, F_1^n, \dots, F_1^\infty\},$$

$$K_{23} = \{A_3, F_7^2, F_7^3, \dots, F_7^n, \dots, F_7^\infty, S_5, P_3'\},$$

$$K_{13} = \{A_2, F_3^2, \dots, F_3^\infty, F_5', S_3\}, \quad K_{15} = \{L_2\},$$

$$K_{35} = \{O_8, O_7\}, \quad K_{25} = \{L_3\}, \quad K_{45} = \{L_5, O_4\},$$

$$K_{123} = \{A_4, F_6^2, F_6^3, \dots, F_6^\infty, F_2^2, F_2^3, \dots, F_2^\infty; P_1', S_1\},$$

$$K_{124} = \{D_1\}, \quad K_{135} = \{O_5, O_2\}, \quad K_{235} = \{O_6, O_3\},$$

$$K_{1234} = \{D_2\}, \quad K_{1245} = \{L_4\}, \quad K_{12345} = \{O_1\}.$$

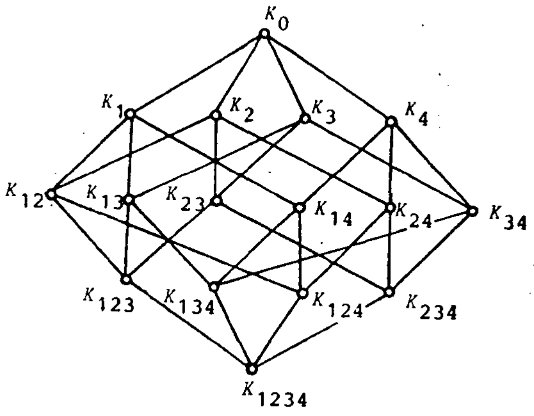


*Fig. 3*

In Fig. 3 we have the graph $G(\mathscr{A}_1)$. Here

$$K_0 = \{A_1\}, \quad K_1 = \{A_2, F_3^2, F_3^3, \ldots, F_3^\infty\},$$
$$K_2 = \{A_3, F_7^2, F_7^3, \ldots, F_7^\infty\}, \quad K_3 = \{S_6\}, \quad K_4 = \{P_6'\},$$
$$K_{12} = \{A_4, F_6^2, F_6^3, \ldots, F_6^\infty, F_2^2, F_2^3, \ldots, F_2^\infty, D_2\},$$
$$K_{13} = \{S_3\}, \quad K_{23} = \{S_5\}, \quad K_{14} = \{P_5'\},$$
$$K_{24} = \{P_3\}, \quad K_{34} = \{O_8, O_7\}, \quad K_{123} = \{S_1\},$$
$$K_{134} = \{P_1'\}, \quad K_{124} = \{O_5, O_2\}, \quad K_{234} = \{O_6, O_3\},$$
$$K_{1234} = \{O_1\}.$$

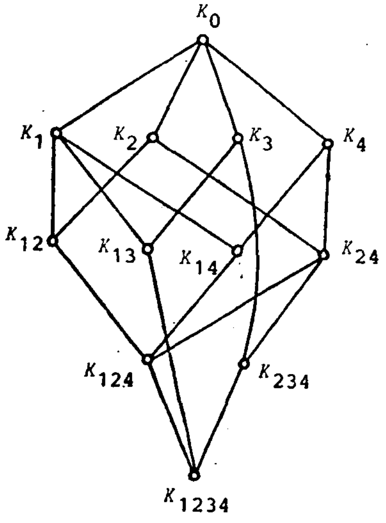


*Fig. 4*

In Fig. 4 we have the graph $G(C_3)$. Here

$$K_0 = \{C_3\},\quad K_1 = \{C_4, F_1^2, F_1^3, \ldots, F_1^\infty, D_1\},$$
$$K_2 = \{A_3, S_5\},\quad K_3 = \{L_3\},\quad K_4 = \{F_8^2, F_8^3, \ldots, F_8^\infty\},$$
$$K_{12} = \{A_4, F_2^2, F_2^3, \ldots, F_2^\infty, S_1\}\quad K_{13} = \{L_4\},$$
$$K_{14} = \{F_5^2, F_5^3, \ldots, F_5^\infty\},\quad K_{24} = \{F_7^2, F_7^3, \ldots, F_7^\infty, P_3'\},$$
$$K_{124} = \{F_6^2, F_6^3, \ldots, F_6^\infty, P_1', D_2\},\quad K_{234} = \{O_6, O_3\},\quad K_{1234} = \{O_1\}.$$

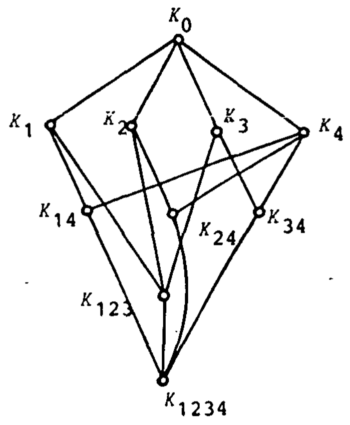


*Fig. 5*

In Fig. 5 we have the graph $G(\mathscr{L}_1)$. Here

$$K_0 = \{L_1\},\quad K_1 = \{L_2\},\quad K_2 = \{L_3\},\quad K_3 = \{L_5\},\quad K_4 = \{O_9, O_8, O_7\},$$
$$K_{14} = \{O_5, O_2\},\quad K_{24} = \{O_6, O_3\},\quad K_{34} = \{O_4\},\quad K_{123} = \{L_4\},\quad K_{1234} = \{O_1\}.$$

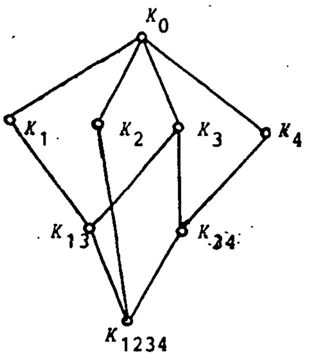


*Fig. 6*

In Fig. 6 we have the graph $G(C_4)$. Here

$$K_0 = \{C_4\},\quad K_1 = \{F_1^2, F_1^3, \ldots, F_1^\infty\},\quad K_2 = \{D_1, L_4\},\quad K_3 = \{A_4\},$$
$$K_4 = \{F_5^2, F_5^3, \ldots, F_5^\infty\},\quad K_{13} = \{F_2^2, F_2^3, \ldots, F_2^\infty, S_1\},$$
$$K_{34} = \{F_6^2, F_6^3, \ldots, F_6^\infty, P_1'\},\quad K_{1234} = \{D_2, O_1\}.$$

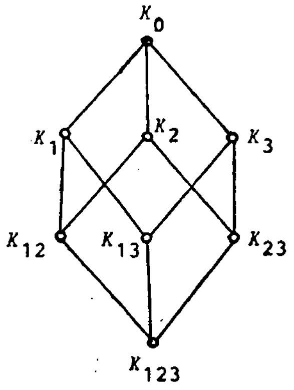


*Fig. 7*

In Fig. 7 we have the graph $G(\mathscr{F}_8^n)$. Here

$$K_0 = \{F_8^n\},\quad K_1 = \{F_8^{n+1}, F_8^{n+2}, \ldots, F_8^\infty\},\quad K_2 = \{F_5^n\},\quad K_3 = \{F_7^n\},$$
$$K_{12} = \{F_5^{n+1}, F_5^{n+2}, \ldots, F_5^\infty\},\quad K_{13} = \{F_7^{n+1}, F_7^{n+2}, \ldots, F_7^\infty, P_3', O_6, O_3\},$$
$$K_{23} = \{F_6^n\}\quad (n > 2),\quad K_{23} = \{F_6^2, D_2\}\quad (n = 2),$$
$$K_{123} = \{F_6^{n+1}, F_6^{n+2}, \ldots, F_6^\infty, P_1', O_1\}.$$

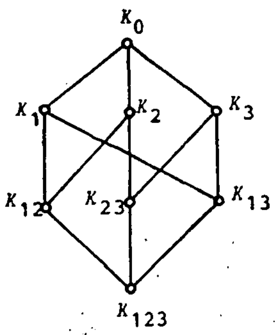


*Fig. 8*

In Fig. 8 we have the graph $G(\mathcal{M}_2)$ where

$$M_2 \in \{A_3, P_6'\}.$$

For $M_2 = A_3$ we have $K_0 = \{A_3\}$, $K_1 = \{A_4, F_2^2, F_2^3, \ldots, F_2^\infty\}$, $K_2 = \{S_5\}$, $K_3 = \{F_7^2, F_7^3, \ldots, F_7^\infty, P_3'\}$, $K_{12} = \{S_1\}$, $K_{13} = \{F_6^2, F_6^3, \ldots, F_6^\infty, P_1', D_2\}$, $K_{23} = \{O_6\}$, $K_{123} = \{O_1\}$.

For $M_2 = P_6'$ we have $K_0 = \{P_6'\}$, $K_1 = \{P_5'\}$, $K_2 = \{O_7, O_8\}$, $K_3 = \{P_3'\}$, $K_{12} = \{O_2, O_5\}$, $K_{23} = \{O_3, O_6\}$, $K_{13} = \{P_1'\}$, $K_{123} = \{O_1\}$.

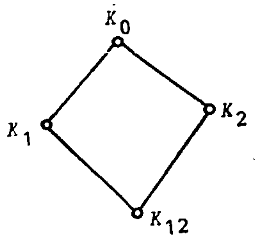


*Fig. 9*

In Fig. 9 we have the graph $G(\mathcal{M}_2)$ where

$$M_2 \in \{F_7^n, F_7^\infty, F_6^2, F_5^n, F_8^\infty, P_3', P_5', D_1, D_3, A_4, L_3, L_5, O_8\}.$$

For $M_2 = F_7^n$ we have $K_0 = \{F_7^n\}$, $K_1 = \{F_6^2, D_2\}$ for $n = 2$, and $K_1 = \{F_6^n\}$ for $n > 2$, $K_2 = \{F_7^{n+1}, F_7^{n+2}, \ldots, F_7^\infty, P_3', O_6, O_3\}$, $K_{12} = \{F_6^{n+1}, \ldots, F_6^\infty, P_1, O_1\}$.

For $M_2 = F_7^\infty$ we have $K_0 = \{F_7^\infty\}$, $K_1 = \{F_6^\infty\}$, $K_2 = \{P_3', O_6, O_3\}$, $K_{12} = \{P_1', O_1\}$.

For $M_2 = F_6^2$ we have $K_0 = \{F_6^2\}$, $K_1 = \{D_2\}$, $K_2 = \{F_6^3, F_6^4, \ldots, F_6^\infty, P_1'\}$, $K_{12} = \{O_1\}$.

For $M_2 = F_5^n$ we have $K_0 = \{F_5^n\}$, $K_1 = \{F_5^{n+1}, F_5^{n+2}, \ldots, F_5^\infty\}$, $K_2 = \{F_6^2, D_2\}$ for $n = 2$ and $K_2 = \{F_6^n\}$ for $n > 2$, $K_{12} = \{F_6^{n+1}, F_6^{n+2}, \ldots, F_6^\infty, P_1', O_1\}$.

For $M_2 = F_8^\infty$ we have $K_0 = \{F_8^\infty\}$, $K_1 = \{F_7^\infty, P_3', O_6, O_3\}$, $K_2 = \{F_5^\infty\}$, $K_{12} = \{F_6^\infty, P_1', O_1\}$.

For $M_2 = P_3'$ we have $K_0 = \{P_3'\}$, $K_1 = \{O_6, O_3\}$, $K_2 = \{P_1'\}$, $K_{12} = \{O_1\}$.

For $M_2 = P_5'$ we have $K_0 = \{P_5'\}$, $K_1 = \{P_1'\}$, $K_2 = \{O_5\}$, $K_{12} = \{O_1\}$.

For $M_2 = D_1$ we have $K_0 = \{D_1\}$, $K_1 = \{L_4\}$, $K_2 = \{D_2\}$, $K_{12} = \{O_1\}$.

For $M_2 = D_3$ we have $K_0 = \{D_3\}$, $K_1 = \{D_1, D_2\}$, $K_2 = \{L_5, O_4\}$, $K_{12} = \{L_4, O_1\}$.

For $M_2=A_4$ we have $K_0=\{A_4\}$, $K_1=\{F_2^2, F_2^3, \dots, F_2^\infty, S_1\}$, $K_2=\{F_6^2, F_6^3, \dots$ $\dots, F_6^\infty, P_1'\}$, $K_{12}=\{D_2, O_1\}$.

For $M_2=L_3$ we have $K_0=\{L_3\}$, $K_1=\{L_4\}$, $K_2=\{O_3, O_6\}$, $K_{12}=\{O_1\}$.

For $M_2=L_5$ we have $K_0=\{L_5\}$, $K_1=\{L_4\}$, $K_2=\{O_4\}$, $K_{12}=\{O_1\}$.

For $M_2=O_8$ we have $K_0=\{O_8\}$, $K_1=\{O_2, O_5\}$, $K_2=\{O_3, O_6\}$, $K_{12}=\{O_1\}$.

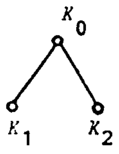


*Fig. 10*

In Fig. 10 we have the graph $G(\mathcal{M}_2)$ where

$$M_2 \in \{O_6, O_7, O_9\}.$$

For $M_2=O_6$ we have $K_0=\{O_6\}$, $K_1=\{O_1\}$, $K_2=\{O_3\}$.

For $M_2=O_7$ we have $K_0=\{O_7\}$, $K_1=\{O_2\}$, $K_2=\{O_3\}$.

For $M_2=O_9$ we have $K_0=\{O_9\}$, $K_1=\{O_1, O_4\}$, $K_2=\{O_2, O_3, O_7, O_8\}$.

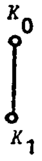


*Fig. 11*

In Fig. 11 we have the graph $G(\mathcal{M}_2)$ where

$$M_2 \in \{P_1', D_2, L_4, F_5^\infty, F_6^n, F_6^\infty, O_4\} \text{ where } n>2.$$

For $M_2=P_1'$ we have $K_0=\{P_1'\}$, $K_1=\{O_1\}$.

For $M_2=D_2$ we have $K_0=\{D_2\}$, $K_1=\{O_1\}$.

For $M_2=L_4$ we have $K_0=\{L_4\}$, $K_1=\{O_1\}$.

For $M_2=F_5^\infty$ we have $K_0=\{F_5^\infty\}$, $K_1=\{F_6^\infty, P_1', O_1\}$.

For $M_2=F_6^n$ we have $K_0=\{F_6^n\}$, $K_1=\{F_6^{n+1}, F_6^{n+2}, \dots, F_6^\infty, P_1', O_1\}$.

For $M_2=F_6^\infty$ we have $K_0=\{F_6^\infty\}$, $K_1=\{P_1', O_1\}$.

For $M_2=O_4$ we have $K_0=\{O_4\}$, $K_1=\{O_1\}$.

In Fig. 12 we have the graph $G(\mathcal{M}_2)$ where $M_2 \in \{O_1, O_3\}$. It has one vertex corresponding to $K_0$ which coincides with $\{O_1\}$ or $\{O_3\}$.

*Fig. 12*

KATEDRE DISCRETE MATHEMATICK
MEH-MAT FACULTET
MGU
MOSCOW

## References

[1] KUDRYAVTSEV, V. B., Functional Systems, 158 pages, Moscow State University, 1982.
[2] YABLONSKY, S. V., GAVRILOV, G. P., KUDRYAVTSEV, V. B., Functions of the Algebra of Logic and Post Classes, 120 pages, Moscow, Nauka, 1966.

*(Received Febr. 3, 1986)*

# Formal properties of literal shuffle

B. Berard

## Abstract

We introduce the literal shuffle operation, that is a more constrained form of the well-known shuffle operation. In order to describe concurrent processes, the shuffle operation models the asynchronous case, while the literal shuffle operation corresponds to a synchronous behaviour.

The closure properties of some classical families of languages under literal shuffle are studied and properties of families of languages defined by means of literal shuffle are given.

## Introduction

The shuffle operation naturally appears in several problems, like concurrency of processes ([9], [10], [11]), or multi-point communication, where all stations share a single bus ([5]). That is one of the reasons of the large theoretical literature about this operation (see for instance [1], [3], [6], [7]). In the latter example, general shuffle operation models the asynchronous case, where each transmitter uses asynchronously the single communication channel. If the hypothesis of synchronism is made (step-lock transmission), the situation is modeled by what can be named "literal" shuffle. Each transmitter emits, in turn, one elementary signal. The same remark holds for concurrency, where general shuffle corresponds to asynchronism and literal shuffle to synchronism.

There are no specific studies of literal shuffle. One of the reasons is perhaps that, when adding the full trio operations, literal shuffle is as powerful as general shuffle. Nevertheless, when a more precise approach is made, literal shuffle appears as satisfying specific properties. In the present paper, we study the literal shuffle operation, particularly in relation with the classical families of languages: regular sets, context-free languages, context-sensitive languages and recursively enumerable sets. The paper is divided in three sections. The first one contains some specific definitions about shuffle and literal shuffle, and some basic properties of these operations. In the second section, we study the closure properties of the families Rat, $\mathcal{C}f$, $\mathcal{CS}$ and $\mathcal{RE}$ under literal shuffle and we show that the family of recursively enumerable sets is the smallest full trio closed under iterated literal shuffle, thus extending a result of M. Jantzen [6] about the shuffle operation.

In the third section, we give some properties of the language families obtained by using literal shuffle, in the same way as the families Shuf and $\mathscr{SE}$ were studied in [6]. The main purpose of this section is to state that the two families obtained that way and $\mathscr{SE}$ are incomparable.

## Notations and basic definitions

Let $X$ be an alphabet. $X^*$ is the free monoid generated by $X$, and $\varepsilon$ will denote the empty word in $X^*$.

Let $f$ be a word in $X^*$, $|f|$ is the length of $f$ and if $f$ is not the empty word, $f^{(i)}$ is the $i^{\text{th}}$ letter of $f$, $|f|_x$ is the number of occurrences of the letter $x$ in $f$.

A word $g$ in $X^*$ is a *subword* of $f$ if $f=ugv$, for some, $u, v$ in $X^*$. If $u$ is the empty word, $g$ is a *prefix* of $f$.

Fin, Rat, $\mathscr{C}f$, $\mathscr{CS}$, $\mathscr{RE}$ will respectively denote the family of finite sets, regular sets, context-free languages, context-sensitive languages, recursively enumerable sets. Let $X$ and $Y$ be two alphabets. A homomorphism $h$ from $X^*$ into $Y^*$ is said to be: *non erasing* if $h(X)\subseteq Y^+$, where $Y^+=Y^*-\{\varepsilon\}$,
*alphabetical* if $h(X)\subseteq Y\cup\{\varepsilon\}$,
*a coding* if $h(X)\subseteq Y$,
*an isomorphism* if $h$ and $h^{-1}$ are codings. In that case, $Y$ is called a copy of $X$ and if $L$ is a language in $X^*$, $h(L)$ is called a copy of $L$.

$\hat{\mathscr{H}}$ is the class of all homomorphism and $\mathscr{H}^{-1}$ is the class of all inverse homomorphisms.

A *full trio* is a family of languages closed under homomorphisms, inverse homomorphisms and intersections with regular sets. $\hat{\mathscr{M}}=(\hat{\mathscr{H}}, \mathscr{H}^{-1}, \wedge\mathscr{R})$ will denote the full trio operations, where $\wedge\mathscr{R}$ is the class of intersections with regular sets. $D_1'^*$ is the resticted Dyck set over the alphabet $\{a, b\}$ generated by the context-free grammar with productions:

$$S\to aSb+SS+\varepsilon \text{ (see [4] and [3] for details).}$$

## Part 1 — Shuffle and literal shuffle

The shuffle operation will be denoted by the symbol ш and is defined for languages $L$ and $M$ in $X^*$ by

$$L ш M = \{f = u_1v_1\dots u_nv_n,\ u_i, v_i \text{ in } X^*,\ u_1\dots u_n\in L,\ v_1\dots v_n\in M\}.$$

The iterated shuffle will be denoted by ш*. Let $L$ be a language in $X^*$, then $L^{ш*}= =\bigcup_{i\geq 0} L_i$, where $L_0=\{\varepsilon\}$ and $L_{i+1}=L_i ш L$. The families Shuf and $\mathscr{SE}$ were introduced by M. Jantzen [6]: Shuf$=(\cup, ш, ш^*)$(Fin) is the least family of languages including Fin and closed under union, shuffle and iterated shuffle. $\mathscr{SE}=(\cup, \cdot, *, ш, ш^*)$(Fin) is the least family of languages including Fin and closed under union, product, Kleene star, shuffle and iterated shuffle.

We give now the specific notations of this paper and make the ideas more precise about literal shuffle.

Let $f$ and $g$ be two words in $X^*$ with the same length $p$. The interleaving $I$ of the words, $f$, $g$ is defined by:

$$I(\varepsilon, \varepsilon) = \varepsilon \quad \text{if} \quad p = 0,$$
$$I(f, g) = f^{(1)} g^{(1)} \ldots f^{(p)} g^{(p)} \quad \text{if} \quad p > 0.$$

Let $L$ and $M$ be languages in $X^*$, we define:

1) *The initial literal shuffle* $ш_1$ :

$$L ш_1 M = \{I(f_1, f_2)\, g \mid f_1, f_2, g \ \text{ in } \ X^*, \ |f_1| = |f_2|, \\ (f_1 g \in L \ \text{ and } \ f_2 \in M) \ \text{ or } \ (f_1 \in L \ \text{ and } \ f_2 g \in M)\}.$$

2) *The literal shuffle* $ш_2$ :

$$\begin{aligned} L ш_2 M = \{f I(g_1, g_2) h \mid f, g_1, g_2, h \ \text{ in } X^*, \ |g_1| = |g_2|, \\ (f g_1 h \in L \ \text{ and } \ g_2 \in M) \ \text{ or } \\ (g_1 \in L \ \text{ and } \ f g_2 h \in M) \ \text{ or } \\ (f g_1 \in L \ \text{ and } \ g_2 h \in M) \ \text{ or } \\ (g_1 h \in L \ \text{ and } \ f g_2 \in M)\}. \end{aligned}$$

*Example:* $L = a^*$ and $M = b^*$

$$L ш_1 M = (ab)^* (a^* \cup b^*),$$
$$L ш_2 M = (a^* \cup b^*)(ab)^* (a^* \cup b^*).$$

3) *The iterated initial literal shuffle* $ш_1^*$ and the *iterated literal shuffle* $ш_2^*$:

$$L^{ш_1^*} = \bigcup_{i \geq 0} L_i, \quad \text{where} \quad L_0 = \{\varepsilon\} \quad \text{and} \quad L_{i+1} = L_i ш_1 L,$$
$$L^{ш_2^*} = \bigcup_{i \geq 0} L_i, \quad \text{where} \quad L_0 = \{\varepsilon\} \quad \text{and} \quad L_{i+1} = L_i ш_2 L.$$

We then define four families of languages:

$$\mathscr{L}_1 \mathscr{S} h = (\cup, ш_1, ш_1^*)(\text{Fin})$$
$$\mathscr{L} \mathscr{S} h = (\cup, ш_2, ш_2^*)(\text{Fin})$$
$$\mathscr{L}_1 \mathscr{S} \mathscr{E} = (\cup, \cdot, *, ш_1, ш_1^*)(\text{Fin})$$
$$\mathscr{L} \mathscr{S} \mathscr{E} = (\cup, \cdot, *, ш_2, ш_2^*)(\text{Fin}).$$

At the end, we summarize some basic properties of the initial literal shuffle and the literal shuffle.

**Proposition 1.1.** Let $X$ be an alphabet and $A$, $B$ languages in $X^*$.

a) The initial literal shuffle and the literal shuffle are not associative operations.
b) The literal shuffle is commutative but the initial literal shuffle is not commutative.

c) $AB \subseteqq A\text{ш}_2B$, $\quad A\text{ш}_1B \subseteqq A\text{ш}_2B \subseteqq A\text{ш}B$.

d) $X^* = X^{\text{ш}_1^*} = X^{\text{ш}_2^*}$.

e) Let $f, g, h$ be words in $X^*$ such that $h=f\text{ш}_1g$ or $h\in f\text{ш}_2g$, then $|h|=|f|+|g|$.

Recall ([1]) that $D_1'^*=(ab)^{\text{ш}*}$; we have:

**Proposition 1.2.**

a) $(ab)^{\text{ш}_1^*} = \{\varepsilon, ab\}\cup a^2(ab)^*b^2$,

b) $(ab)^{\text{ш}_2^*} = D_1'^*$.

The initial literal shuffle seems then to be less powerful than both shuffle and literal shuffle. However, we will see that even a very simple language like $((ab)^{\text{ш}_1^*})^{\text{ш}_1^*}$ is not context-free. Furthermore, the three families $\mathscr{S}\mathscr{E}$, $\mathscr{L}_1\mathscr{S}\mathscr{E}$ and $\mathscr{L}\mathscr{S}\mathscr{E}$ are pairwise incomparable.

*Proof.*
a) The proof is straightforward.
b) From the definition, we can write $(ab)^{\text{ш}_2^*} = \bigcup\limits_{p\geqq 0} L_p$, where

$$L_0 = \{\varepsilon\} \quad\text{and}\quad L_{p+1} = L_p\text{ш}_2\{ab\}.$$

Since $A\text{ш}_2B\subseteqq A\text{ш}B$ (Proposition 1.1.c), it is easy to verify that

$$A^{\text{ш}_2^*} \subseteqq A^{\text{ш}^*}, \quad\text{thus}\quad (ab)^{\text{ш}_2^*} \subseteqq D_1'^*.$$

For the converse inclusion let $f$ be in $D_1'^*$ with $|f|=2p$. An induction argument proves that $f$ is in $L_p$.

The basis when $p=1$ is trivial.

Induction step. Assume the result for words of length $2p$ and consider a word $f$ in $D_1'^*$ of length $2(p+1)$. There are two possibilities:

*Case 1.* $f=(ab)^{p+1}$. By the induction hypothesis, $(ab)^p$ is in $L_p$, thus $f$ belongs to $L_p\{ab\}$. Since $L_p\{ab\}\subseteqq L_p\text{ш}_2\{ab\}$ (Prop. 1.1.c), $f\in L_{p+1}$.

*Case 2.* $f=f_1f_2f_3$, where $f_2$ is a word of $D_1'$, the set of restricted Dyck primes, with $|f_2|\geqq 4$.

Let $u_0=\varepsilon, u_1, \dots, u_{2k}=f_2$, be the sequence of prefixes of $f_2$, $k\geqq 2$, and let $\|u_j\|=|u_j|_a-|u_j|_b$ be the height of the word $u_j$. If $i$ is the greatest integer such that $\|u_i\|$ is maximum, then there exists a letter $x$ in $\{a, b\}$ and a word $v$ in $\{a, b\}^*$ with $f_2= =u_{i-2}xabbv$. We define $g=f_1u_{i-2}$, $v_1=xb$, $v_2=ab$, $h=vf_3$. $f=gI(v_1, v_2)h$, thus $f$ is in $gv_1h\text{ш}_2ab$. Since $gv_1h$ is a word in $D_1'^*$ of length $2p$, $gv_1h$ is in $L_p$ by induction hypothesis. Consequently, $f$ is in $L_{p+1}$.

## Part 2 — Closure properties of the families Rat, $\mathscr{C}f$, $\mathscr{CS}$ and $\mathscr{RE}$ under literal shuffle

We first show that, when adding the full trio operations, literal shuffle is a powerful as shuffle.

Recall ([3]) that a full trio is closed under shuffle if and only if it is closed under intersection.

**Proposition 2.1.** Let $\mathscr{L}$ be a full trio. The following properties are equivalent:

a) $\mathscr{L}$ is closed under shuffle.
b) $\mathscr{L}$ is closed under literal shuffle.
c) $\mathscr{L}$ is closed under initial literal shuffle.

*Proof.* The result is easily obtained from the two following facts. Let $L$ and $M$ be languages respectively in $X^*$ and $Y^*$.

*Fact 1.* Assume that $X$ and $Y$ are disjoint alphabets; we define regular languages in $(X\cup Y)^*$ by:

$$R_1 = (XY)^*(X^*\cup Y^*) \quad \text{and} \quad R_2 = (X^*\cup Y^*)(XY)^*(X^*\cup Y^*).$$

Then

$$L ш_1 M = (L ш M)\cap R_1 \quad \text{and} \quad L ш_2 M = (L ш M)\cap R_2.$$

*Fact 2.* If $\$$ is a new letter and if $h$ is the homomorphism from $(X\cup Y\cup\{\$\})^*$ onto $(X\cup Y)^*$ defined by:

$$h(z)=z, \text{ for each } z \text{ in } X\cup Y, \text{ and } h(\$)=\varepsilon,$$

then

$$L ш M = h[h^{-1}(L) ш_1 h^{-1}(M)] = h[h^{-1}(L) ш_2 h^{-1}(M)].$$

**Proposition 2.2.** Let $L$ be a language in $X^*$, let $\$$ be a letter not in $X$ and let $h$ be the homomorphism from $(X\cup\{\$\})^*$ onto $X^*$ defined by: $h(x)=x$ if $x$ is in $X$, $h(\$)=\varepsilon$. Then,

$$L^{ш^*} = h[(h^{-1}(L))^{ш_1^*}] = h[(h^{-1}(L))^{ш_2^*}].$$

*Proof.* Using Proposition 1.1.c, we can get

$$h[(h^{-1}(L))^{ш_1^*}] \subseteqq h[(h^{-1}(L))^{ш_2^*}].$$

Furthermore, if $\varphi$ is an arbitrary homomorphism and if $A$, $B$ are languages, then

$$\varphi(A ш B) \subseteqq (A) ш \varphi(B).$$

Therefore, we have the following inclusions:

$$h[(h^{-1}(L))^{ш_1^*}] \subseteqq h[(h^{-1}(L))^{ш_2^*}] \subseteqq L^{ш^*}.$$

Conversely, we use the definition of iterated shuffle and initial literal shuffle:

$$L^{ш^*} = \bigcup_{n\geqq 0} L_n, \quad L_0 = \{\varepsilon\}, \quad L_{n+1} = L_n ш L$$

and

$$(h^{-1}(L))^{ш_1^*} = \bigcup_{n\geqq 0} M_n, \quad M_0 = \{\varepsilon\}, \quad M_{n+1} = M_n ш_1 h^{-1}(L).$$

We prove that for each integer $n \geqq 0$, $L_n \subseteqq h(M_n)$. If $n=0$ or $n=1$, the result is immediate. Assume $n \geqq 2$ and let $u$ be a word in $u_1 ш \dots ш u_n$, where $u_i \in L$. There exists an integer $p \geqq 1$ such that

$$u = \prod_{j=1}^{p} (u_{1,j} \dots u_{n,j}), \quad u_{i,j} \text{ in } X^*, \quad u_i = u_{i,1} \dots u_{i,p}.$$

We define a sequence of words $f_i$, $1 \leqq i \leqq n$, by:

$$f_i = f_{i,1} \cdots f_{i,p}, \quad f_{i,j} = \$^{r_{i,j}} u_{i,j} \$^{s_{i,j}}, \quad \text{with:}$$

$$r_{1,j} = 0$$

$$r_{i,j} = 2^{i-2}(|u_{1,j}| + \dots + |u_{i-1,j}|), \quad i \geqq 2$$

$$s_{1,j} = |u_{2,j}| + \dots + |u_{n,j}|$$

$$s_{i,j} = (2^{i-2}-1)|u_{i,j}| + 2^{i-2}(|u_{i+1,j}| + \dots + |u_{n,j}|), \quad i \geqq 2.$$

Clearly, $f_i$ belongs to $h^{-1}(L)$, $1 \leqq i \leqq n$,

$$|f_1| = |u| \quad \text{and} \quad |f_i| = 2^{i-2}|u| \quad \text{for each} \quad i \geqq 2.$$

Define:

$$g_1 = f_1 \quad \text{and for} \quad 1 \leqq i \leqq n-1, \quad g_{i+1} = g_i ш_1 f_{i+1}.$$

Obviously, $g_i$ is in $M_i$, $1 \leqq i \leqq n$. Further, $|g_i| = |f_{i+1}| = 2^{i-1}|n|$ for $1 \leqq i \leqq n$, and $|g_n| = 2^{n-1}|u|$.

Then, we can write $g_i = g_{i,1} \dots g_{i,p}$, where $|g_{i,j}| = |f_{i+1,j}|$, $1 \leqq i < n$, and $|g_{n,j}| = 2|f_{n,j}|$. It is easy to prove by induction on $i \geqq 2$ that:

$$g_{i,j} = g'_{i,j} \$^{t_{i,j}}, \quad \text{where} \quad t_{i,j} = s_{i,j} + |u_{i+1,j}|$$

and

$$h(g_{i,j}) = u_{1,j} \dots u_{i,j}.$$

For $i=n$, we obtain:

$$g_n = g_{n,1} \cdots g_{n,p}, \quad h(g_{n,j}) = u_{1,j} \cdots u_{n,j},$$

hence $h(g) = u$ and $u$ is in $h(M_n)$.

From $L_n \subseteqq h(M_n)$, we have $L^{ш^*} \subseteqq h[(h^{-1}(L))^{ш_1^*}]$, and the proof is complete.

We now state the closure properties of the families Rat, $\mathscr{CS}$ and $\mathscr{RE}$ under literal shuffle. They can be obtained by easy machine constructions.

**Proposition 2.3.**

a) The families Rat, $\mathscr{CS}$ and $\mathscr{RE}$ are closed under $ш_1$ and $ш_2$.

b) Moreover, the families $\mathscr{CS}$ and $\mathscr{RE}$ are closed under $ш_1^*$ and $ш_2^*$.

**Corollary 1.** The families $\mathscr{L}_1\mathscr{SE}$ and $\mathscr{LSE}$ are both contained in the family of context-sensitive languages.

We will see in the next section that there are, in fact, proper containments.

Using Propositions 2.2 and 2.3 together with a result of M. Jantzen ([6]): $\mathscr{RE} = (\mathscr{M}\, ш^*)(\text{Fin})$, we can show:

**Corollary 2.** The family of recursively enumerable sets is the least family of languages including the finite sets and closed under the full trio operations and the iterated literal shuffle.

The same result holds with the iterated initial literal shuffle:

$$\mathcal{RE} = (\hat{\mathcal{M}}, ш_1^*)(\text{Fin}) = (\hat{\mathcal{M}}, ш_2^*)(\text{Fin}).$$

Property 2.3.a) does not remain true for context-free languages: let $L$ and $M$ be two different copies of the restricted Dyck set over the disjoint alphabets $\{a, b\}$ and $\{c, d\}$, respectively. Then, neither $L ш_1 M$ nor $L ш_2 M$ are context-free languages.

We mention a strong result of M. Latteux about the shuffle operation:

**Proposition 2.4.** ([7]) Let $L$ and $M$ be two languages over disjoint alphabets $X$ and $Y$ respectively. $L ш M$ is a context-free language if and only if either $L$ or $M$ is a regular language.

This result does not extend to the initial literal shuffle: Let $G$ be the context-free, non regular language over the alphabet $\{a, b\}$ defined by:

$$G = \{a^{n_1}b \dots a^{n_k}b \mid k \geqq 1, \quad n_i \geqq 0, \quad \exists i \neq n_i\}.$$

($G$ is known as the Goldstine's language.) If $\bar{G}$ is a copy of $G$ over the alphabet $\{\bar{a}, \bar{b}\}$, we have:

**Proposition 2.5.** $G ш_1 \bar{G}$ is a context-free language.

*Scheme of the proof.* Let $\$$ be a new letter and let $\hat{G}$ be the following language in $(\{a, b, \$\} \times \{a, b, \$\})^*$:

$$\hat{G} = \left\{ \begin{bmatrix} f\$^p \\ g\$^q \end{bmatrix}, \quad f \in G, \quad g \in G \quad \text{and} \quad p + |f| = q + |g| \right\}^{1)}.$$

Let $h$ be the homomorphism from $(\{a, b, \$\} \times \{a, b, \$\})^*$ into $\{a, b, \bar{a}, \bar{b}\}^*$ defined by:

$$h\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = x\bar{y} \quad \text{if} \quad x, y \in \{a, b\}, \quad h\left(\begin{bmatrix} \$ \\ \$ \end{bmatrix}\right) = \varepsilon,$$

$$h\left(\begin{bmatrix} x \\ \$ \end{bmatrix}\right) = x \quad \text{if} \quad x \in \{a, b\} \quad \text{and} \quad h\left(\begin{bmatrix} \$ \\ y \end{bmatrix}\right) = \bar{y} \quad \text{if} \quad y \in \{a, b\}.$$

Clearly enough, $h(\hat{G}) = G ш_1 \bar{G}$. Then, it suffices to prove the context-freeness of $\hat{G}$, and we build a pushdown automaton recognizing $\hat{G}$. We will use two different versions of non-deterministic pushdown automata recognizing $G$ (by final states).

*First version.* The underlying idea of how this automaton works is the following: let $w$ be a word in $\{a, b\}^*$. Non-deterministically, a block of $a$'s is chosen. The $b$'s preceding this block are pushed into the stack. Then, each $a$ in the chosen block makes

1) If $x, y \in \{a, b, \$\}^*$ with $|x| = |y| = n$, we write $\begin{bmatrix} x \\ y \end{bmatrix}$ for $\begin{bmatrix} x^{(1)} \\ y^{(1)} \end{bmatrix} \dots \begin{bmatrix} x^{(n)} \\ y^{(n)} \end{bmatrix}$.

a $b$ to be popped from the stack. The word $w$ is accepted if the number of $a$'s in the chosen block does not match the number of $b$'s in the stack. (Initially, the stack contains a single $b$.)

*Second version.* It allows to keep in the stack, after checking, the rank of the chosen block of $a$'s. It is based upon the fact that $G$ is also defined by:

$$G = \{a^{n_1}b \ldots a^{n_p}b \mid n_1 \neq 1 \quad \text{or there exists a } k, \quad 1 \leqq k \leqq p-1,$$
$$\text{such that } n_{k+1} \neq n_k + 1\}.$$

The automaton first checks wether or not $n_1 = 1$ or chooses a block of $a$'s. (Let $k+2$ be its rank.) In the second case, the first $k$ $b$'s are pushed into the stack, then the $a$'s of the following block (their number is $n_{k+1}$) are also pushed into the stack. The $b$ is skipped and it is then checked if the number of $a$'s in the following block is different of $n_{k+1}+1$ (by using the $n_{k+1}$ $a$'s on the top of the stack). If this test is positive, the word is accepted and the rank of the current block can be retrieved from the stack (number of $b$'s plus 2).

Now we can describe a non-deterministic pushdown automaton recognizing $\hat{G}$. As long as couples of letters $\begin{bmatrix} a \\ a \end{bmatrix}$ or $\begin{bmatrix} b \\ b \end{bmatrix}$ are encountered, the automaton works as in the second version. As soon as a couple $\begin{bmatrix} a \\ b \end{bmatrix}$ or $\begin{bmatrix} b \\ a \end{bmatrix}$ is encountered (involving that one of the upper and lower words is then known to be in $G$), the automaton uses the $a$'s at the top of the stack for determining which word is in $G$ (say the upper word). Using the $b$'s in the stack and switching for first version, the automaton checks then that the other word (here the lower one) does belong to $G$.

Clearly, no problem appears if the first encountered couple of different letters is $\begin{bmatrix} x \\ \$ \end{bmatrix}$ or $\begin{bmatrix} \$ \\ y \end{bmatrix}$, $x, y \in \{a, b\}$.

*Open question:* Do there exist two non regular languages $L$ and $M$ over disjoint alphabets, such that $L ш_2 M$ is context-free?

Property 2.3b) does not hold for Rat or $\mathscr{C}f$. We use Proposition 2.2 with the language $L = \{abc\}$. It is easy to see that $L^{ш^*}$ is not context-free. $M = h^{-1}(L) = \$^*a\$^*b\$^*c\$^*$ is a regular language and since $L^{ш^*} = h(M^{ш_1^*}) = h(M^{ш_2^*})$ is not in $\mathscr{C}f$, neither $M^{ш_1^*}$ nor $M^{ш_2^*}$ is a context-free language.

However, regular languages or context-free languages can be obtained in some very particular cases:

**Proposition 2.6.** Let $F$ be a finite set. $F^{ш_1^*}$ is a regular language.

**Proposition 2.7.** Let $F$ be a finite set such that for any word $f$ in $F$, the length of $f$ is less than or equal to 2. Then, $F^{ш_2^*}$ is a context-free language.

*Proof of Prop. 2.6.* The proof consists of a construction of a left linear grammar such that $L(G) = F^{ш_1^*}$.

Since $\emptyset^{ш_1^*}=\{\varepsilon\}$ and for any language $A$, $(A\cup\{\varepsilon\})^{ш_1^*}=A^{ш_1^*}$, we may assume that $F$ is not the empty set and does not contain the empty word; $F=\{f_1, \ldots, f_k\}$, $k\geqq 1$. If $X$ is the alphabet of $F$, we set $p=\text{card}\,(X)$, $t=\max\,\{|f_j|, 1\leqq j\leqq k\}$ and we consider the set $X^t$ of words in $X^*$ with length $t$: $X^t=\{g_1, \ldots, g_m\}$ where $m=p^t$. We can write

$$F^{ш_1^*} = \bigcup_{i\geqq 0} L_i, \quad L_0 = \{\varepsilon\}, \quad L_{i+1} = L_i\, ш_i\, F.$$

Let $n_0$ be the smallest integer greater than or equal to $k$, such that for each word $f$ in $L_{n_0}$, $|f|\geqq t$.

Since $\varepsilon\notin F$, the words in $L_i$ are strictly shorter than the words in $L_{i+1}$ and such an integer $n_0$ can be found.

We define: $R=\bigcup_{i\leqq n_0-1} L_i$, $R$ is a finite set,

$$J(i) = \{f\in L_{n_0}|g_i \text{ is a prefix of } f\}, \quad 1 \leqq i \leqq m,$$

$$I = \{i\in\{1, \ldots, m\}|J(i) \neq \emptyset\}$$

and for each $i\in I$, $q_i=\text{card}\,(J(i))$, so that

$$J(i) = \{h_{i,1}, \ldots, h_{i,q_i}\} \text{ with } h_{i,r} = g_i u_{i,r} \text{ for some } u_{i,r} \text{ in } X^*,$$

$1 \leqq r \leqq q_i$,

$$L_{n_0} = \bigcup_{i\in I} J(i).$$

For each $(i,j)$, $1\leqq i\leqq m$, $1\leqq j\leqq k$, there exists a unique integer $s(i,j)$ in $\{1, \ldots, m\}$ and a unique word $v_{i,j}$ in $X^*$ such that:

$$g_i\, ш_1\, f_j = g_{s(i,j)}\, v_{i,j}.$$

Now we can finish the proof by constructing a grammar $G=(X, N, S, P)$; $N=\{S, D_1, \ldots, D_m\}$, where $S, D_1, \ldots, D_m$ are new letters. The rules of $P$ are the following:

(i) $S\to w$ for each word $w$ in $R$;

(ii) $S\to D_i u_{i,r}$ for each $r$, $1\leqq r\leqq q_i$, for each $i$ in $I$;

(iii) $D_i \to g_i$, $1 \leqq i \leqq m$;

(iv) $D_i \to D_{s(i,j)} v_{i,j}$, $1 \leqq j \leqq k$, $1 \leqq i \leqq m$.

$G$ is left linear and it is easy to see that $L(G, S)=F^{ш_1^*}$.

*Proof of Prop. 2.7.* Let $F$ be a finite set in $X^*$ and $L=F^{ш_2^*}$. If every word in $F$ is of length less than or equal to 1 and if $X$ is of minimal cardinality, then $L=X^*$ is a regular language. Since $(A\cup\{\varepsilon\})^{ш_2^*}=A^{ш_2^*}$ for any language $A$, we may assume that $F$ does not contain the empty word.

We define a sequence of languages $F_n$, $n\geqq 1$, inductively by: $F_1=F$,

$F_{n+1}=\{f\in X^*|$there exists a word $g$ in $F_n$ such that:

either $g=g_1g_2$, $g_2\neq\varepsilon$ and $f=g_1yg_2$, where $y$ is a word of length 1 in $F$, or $g=g_1xg_2$; for some $x$ in $X$ and $f=g_1y_1xy_2g_2$, where $y_1y_2$ is a word of length 2 in $F$.}

For each $n\geqq 1$, the set $F_n$ is contained in $L$, therefore the language $M$ defined by $M=\bigcup_{n\geqq 1} F_n$ is also contained in $L$. It is straightforward to verify that $L$ is a submonoid of $X^*$; it follows that $M^*\subseteqq L$. The converse inclusion also holds; the argument is an induction on the length of a word in $L$.

Since $L=M^*$, it suffices to show that $M$ is a context-free language. Thus, we construct a context-free grammar $G=(X, N, S, P)$ such that $L(G, S)=M$.

We consider the fixed alphabet $X=\{a_1, \dots, a_p\}$ and we define: $N=\{S, T_1, \dots, T_p\}$, where $S, T_1, \dots, T_p$ are new letters; the homomorphism $h$ from $X^*$ into $N^*$ such that $h(a_i)=T_i$, $1\leqq i\leqq p$;

$$I = \{i\in\{1, \dots, p\} \mid a_i\in F\}$$

and $w_j=a_{j_1}a_{j_2}$, $1\leqq j\leqq k$; the words of length 2 in $F$.

The productions of $P$ are the following:

(i) $S \to T_i, \quad i\in I$

$S \to T_{j_1}T_{j_2}, \quad 1\leqq j\leqq k$

(ii) $\left.\begin{array}{ll} T \to T_iT; & i\in I, \\ T \to T_{j_1}TT_{j_2}, & 1\leqq j\leqq k, \end{array}\right\}$ for any variable $T\in\{T_1, \dots, T_p\}$

(iii) $T_i \to a_i, \quad 1\leqq i\leqq p$.

Clearly, this grammar generates $M$.

## Part 3 — Properties of the families $\mathscr{L}_1\mathscr{S}\mathscr{E}$ and $\mathscr{L}\mathscr{S}\mathscr{E}$

We do not mention in this part specific properties of the families $\mathscr{L}_1\mathscr{S}h$ and $\mathscr{L}\mathscr{S}h$; however, we state two useful results about some particular languages in these families.

**Proposition 3.1.** The language $N=((ab)^{\text{Ш}*}_1)^{\text{Ш}*}_1$ (in $\mathscr{L}_1\mathscr{S}h$) is not context-free.

*Proof*. (the details are omitted)

a) Let $f$ be a word in $\{a, b\}^*$. The height of $f$ is $\|f\|=|f|_a-|f|_b$ and $\mathrm{PR}(n)$ denotes the set of all prefixes $g$ of the words in the language $N$, satisfying: $|g|\leqq n$.

We define, for each integer $n\geqq 0$, $H(n)=\mathrm{Max}\,\{\|g\|,\ g\in\mathrm{PR}(n)\}$. By induction on $n\geqq 2$, we can obtain the following inequality:

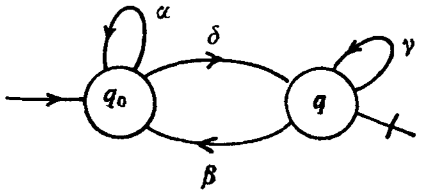
$$H(n) \leqq 6\log_2(n).$$

b) A sequence $f_k$; $k\geqq 1$, of words in $N$ can be constructed, such that: $f_k=g_kb^{3k+4}$, for some word $g_k$ in $\{a, b\}^*$.

c) We suppose now that the language $N$ is context-free and, using the Iteration Theorem, [4], we will obtain a contradiction. Let $N_0$ be the integer from the Iteration Theorem and let $h=f_{N_0}$ be the word of $N$, obtained as in b): $h=g_{N_0}b^{3N_0+4}$, where the last $3N_0+4$ $b$'s are distinguished. There exists a factorization $h=\alpha u\beta v\gamma$, such that $h_p=\alpha u^p\beta v^p\gamma\in N$, for any $p\geqq 0$. The height of $h_p$ is 0, for any $p\geqq 0$, and $v$ is a subword of $b^{3N_0+4}$. Thus, $\|u\|>0$ and, using a), we obtain a sequence $\alpha u$ of prefixes of $N$, such that $\|\alpha\|+p\|u\|\leqq 6\log_2(|\alpha|+p|u|)$, which is impossible. Hence, $N$ is not context-free.

**Proposition 3.2.** The language $P=\{ab, cd\}^{\text{Ш}_2^*}$ (in $\mathcal{L}\mathcal{S}h$) is a generator of the family of context-free languages.

*Proof.* We define the words $\alpha=a^{m+n}$, $\beta=b^n(ac)^p$, $\gamma=(bd)^pb^m$ and $\delta=ab$, where $m$, $n$ and $p$ are integers, $m\geqq 2$, $p\geqq 2$, $n\geqq p+1$. We then define a regular set $K$ recognized by the transition system ([4]) of the figure below:



At the end, we introduce the context-free language $A$, generated by the grammar with productions: $T\to\alpha T\beta T\gamma+\delta$.

We shall prove that $P\cap K=A$. Since $\{\alpha,\beta,\gamma,\delta\}$ is a code[2)], it proves that $P$ is a generator of $f$([2]).

We will say that a word $f$ is directly reduced in a word $g$ if $f=f'axbf''$ or $f=f'cxdf''$ and $g=f'xf''$, for some letter $x$ in $X$ and some words $f'$, $f''$ in $X^*$. We will write $f\to g$ and $\xrightarrow{*}$ will denote the reflexive and transitive closure of $\to$. If $f\xrightarrow{*}g$, we say that $f$ is reduced in $g$.

a) It can be shown by induction on $k\geqq 2$ that, if $f$ is a word in $A$, $|f|\leqq k$, then $f$ is reduced in $ab$. This gives the inclusion $A\subseteq P$.

b) It is easy to see, by induction on the length of a word in $A$, that $A\subseteq K$.

c) So far, we have obtained the inclusion $A\subseteq P\cap K$. To get the converse inclusion, we need two facts:

*Fact 1:* Let $f$ be a word in $P$, neither $\alpha\delta\gamma$ nor $\beta\delta\beta$ is a subword of $f$.

*Fact 2:* Let $f=f'\alpha\delta\beta\delta\gamma f''$ be a word in $P$. Then $f$ is reduced in $f'\delta f''$, and this reduction is the only one which can concern the subword $\alpha\delta\beta\delta\gamma$ of $f$. Let $f$ be a word in $P\cap K$. The argument is again an induction on the length of $f$.

*Case 1.* $\alpha$ is not a subword of $f$. Since $f$ is in $K$, it can be written as: $f=(\delta\gamma^{r_1}\beta)\ldots\ldots(\delta\gamma^{r_k}\beta)\delta\gamma^{r_{k+1}}$. Since $f$ is in $P$, $|f|_a=|f|_b$, therefore $k=0$ and $r_{k+1}=0$; hence $f=ab$ is in $A$.

*Case 2.* $\alpha$ is a subword of $f$. We then consider the last occurrence of $\alpha$ in $f$, so that $f=f'\alpha f''$, $\alpha$ is not a subword of $f''$. Since $f$ is in $K$ and in $P$, using Fact 1, we obtain:

2) A subset $C$ in $X^+$ is a code if $C^*$ is a free monoid with base $C$.

$f''=\delta\beta\delta\gamma h$, $f=f'\alpha\delta\beta\delta\gamma h$ is reduced in $g=f'\delta h$. Obviously, $g$ is in $K$ and, using Fact 2, it turns out that $g$ is in $P$, too. By induction hypothesis, $g$ belongs to $A$, and we consider the place where the rule $T\to\delta$ has been applied in a derivation for $g$: $T\overset{*}{\Rightarrow}m'Tm''\overset{*}{\Rightarrow}m'\delta m''\overset{*}{\Rightarrow} g$, where $m'\overset{*}{\Rightarrow}f'$ and $m''\overset{*}{\Rightarrow}h$. Since $T\overset{*}{\Rightarrow}m'Tm''\overset{*}{\Rightarrow}$ $\overset{*}{\Rightarrow}m'\alpha T\beta T\gamma m''\overset{*}{\Rightarrow}f$, $f$ belongs to $A$. At the end, we have $A=P\cap K$ and the proof is complete.

Before comparing the families $\mathscr{L}_1\mathscr{S}\mathscr{E}$, $\mathscr{L}\mathscr{S}\mathscr{E}$ and $\mathscr{S}\mathscr{E}$, we provide some necessary conditions for a language to belong to one of them. Recall ([6]) that every infinite language in $\mathscr{S}\mathscr{E}$ contains an infinite regular set. Using Proposition 2.6 and an inductive proof, we can extend this property:

**Lemma 3.1.** Every infinite language in $\mathscr{L}_1\mathscr{S}\mathscr{E}$ or in $\mathscr{L}\mathscr{S}\mathscr{E}$ contains an infinite regular set.

**Proposition 3.3.**

a) The language $\{a^n b^n | n\geqq 0\}$ is not in $\mathscr{L}_1\mathscr{S}\mathscr{E}\cup\mathscr{L}\mathscr{S}\mathscr{E}$,
b) the language $\{a^{2^n} | n\geqq 0\}$ is not in $\mathscr{L}_1\mathscr{S}\mathscr{E}\cup\mathscr{L}\mathscr{S}\mathscr{E}$.

Proposition 3.3 b) gives the proper inclusions:

**Corollary 1.** $\mathscr{L}_1\mathscr{S}\mathscr{E}\subsetneqq\mathscr{C}\mathscr{S}$, $\mathscr{L}\mathscr{S}\mathscr{E}\subsetneqq\mathscr{C}\mathscr{S}$.

Using proposition 3.3 a) and the preceding results, we have:

**Corollary 2.** Each of the families $\mathscr{L}_1\mathscr{S}h$, $\mathscr{L}\mathscr{S}h$, $\mathscr{L}_1\mathscr{S}\mathscr{E}$ and $\mathscr{L}\mathscr{S}\mathscr{E}$ is incomparable with the family of context-free languages.

**Proposition 3.4.** The families $\mathscr{L}\mathscr{S}\mathscr{E}$ and $\mathscr{L}$ (EDTOL) are incomparable.

*Proof.* The language $\{a^{2^n} | n\geqq 0\}$ is in $\mathscr{L}$ (EDTOL) ([11]) and does not belong to $\mathscr{L}\mathscr{S}\mathscr{E}$. The language $P=\{ab, cd\}^{\text{Ш}*}$ is in $\mathscr{L}\mathscr{S}\mathscr{E}$ but it does not belong to $\mathscr{L}$ (EDTOL), since it is context-free generator ([8], Proposition 3.2).

**Lemma 3.2.** Let $L$ be a language in $X^*$, where $X$ is of minimal cardinality, $L\in\mathscr{S}\mathscr{E}$, then
either $L$ is regular,
or for each letter $x$ in $X$, for each integer $p\geqq 0$, there exists a word $f$ in $L$, such that $x^p$ is a subword of $f$.

**Lemma 3.3.** Let $L$ be a language in $X^*$, $L\in\mathscr{L}\mathscr{S}\mathscr{E}$. Then, either $L$ is regular or the two conditions hold:

(i) there exists a letter $x$ in $X$ and an integer $n_0$ such that, for each integer $p\geqq 0$, a word $f=gh$ can be found in $L$, where $|g|\leqq n_0+p$ and $x^p$ is a subword of $g$.

(ii) there exists a letter $y$ in $X$ such that, for each integer $p\geqq 0$, a word $f=gh$ can be found in $L$, where $|g|\geqq p$ and $y^p$ is a subword of $h$.

We now consider languages over a fixed ordered alphabet $X=\{a_1, \ldots, a_n\}$ with $n=2$, satisfying:

(*) There exist integers $k_1, \ldots, k_{n-1}$ in $\mathbf{Z}$, such that for each word $f$ in $L$, $|f|_{a_i}-|f|_{a_{i+1}}=k_i$, $1\leqq i\leqq n-1$.

**Lemma 3.4.** Let $L$ be a language in $X^*$, satisfying the property $(*)$ above.

a) if $L$ is in $\mathscr{L}_1\mathscr{S}\mathscr{E}$, then $L \cap a_1^* a_2^* \dots a_n^*$ is a finite set,
b) If $L$ is in $\mathscr{L}\mathscr{S}\mathscr{E}$ and $n \geqq 3$, then $L \cap a_1^* a_2^* \dots a_n^*$ is a finite set.

We can now state the main result of this section:

**Proposition 3.5.**

The families $\mathscr{L}_1\mathscr{S}\mathscr{E}$, $\mathscr{L}\mathscr{S}\mathscr{E}$ and $\mathscr{S}\mathscr{E}$ are pairwise incomparable.
The families $\mathscr{L}_1\mathscr{S}h$, $\mathscr{L}\mathscr{S}h$ and Shuf are pairwise incomparable.

*Proof.*

— The language $L = (abc)^{\sqcup\!\sqcup_2^*}$ is in $\mathscr{L}\mathscr{S}h$ and it is easy to see that $L$ is not regular. Moreover, if $b^p$ is a subword of a word in $L$, then $p \leqq 3$. Using Lemma 3.2, we obtain: $L \notin \mathscr{S}\mathscr{E}$.

— The language $M = (abc)^{\sqcup\!\sqcup^*}$ is in Shuf and $M \cap a^* b^* c^*$ is equal to $\{a^n b^n c^n | n \geqq 0\}$. Since $M$ has property $(*)$, we can use Lemma 3.4 a) and b). Thus $M$ is neither in $\mathscr{L}_1\mathscr{S}\mathscr{E}$ nor in $\mathscr{L}\mathscr{S}\mathscr{E}$.

— The restricted Dyck set $D_1'^*$ is in the families Shuf and $\mathscr{L}\mathscr{S}h$, (Proposition 1.2. b)), and $D_1'^*$ has the property $(*)$. By Lemma 3.4. a), we have: $D_1'^*$ does not belong to the family $\mathscr{L}_1\mathscr{S}\mathscr{E}$.

— The language $N = ((ab)^{\sqcup\!\sqcup_1^*})^{\sqcup\!\sqcup_1^*}$ is in $\mathscr{L}_1\mathscr{S}h$ and is not regular (Proposition 3.1). Using Lemma 3.2 and Lemma 3.3 we can show that $N$ is not in $\mathscr{L}\mathscr{S}\mathscr{E}$ and $N$ is not in $\mathscr{S}\mathscr{E}$.

8, BD DE l'HÔPITAL
75 005 PARIS — FRANCE

## References

[1] Araki, T. and N. Tokura, Flow languages equal recursively enumerable languages, Acta Informatica 15, 209—217, (1981).
[2] Beauquier, J., Générateurs algébriques et systèmes de paires itérantes, Theoretical Computer Science 8, 293—323, (1979).
[3] Ginsburg, S., Algebraic and Automata-Theoretic Properties of Formal Languages, North-Holland (1975).
[4] Harrison, M. A., Introduction to Formal Language Theory, Addison Wesley (1978).
[5] Iwama, K., Unique decomposability of shuffled strings; a fromal treatment of asynchronous time-multiplexed communication, 5th ACM Symp. on Theory of Comput., 374—381, (1983).
[6] Jantzen, M., The power of synchronizing operations on strings, Theoretical Computer Science 14, 127—154, (1981).
[7] Latteux, M., Cônes rationnels commutatifs, Journal of Computer and System Sciences 18, 307—333, (1979).
[8] Latteux, M., Sur les générateurs algébriques et linéaires, Publication du Laboratoire de Calcul de l'Université de Lille I, n° I. T. 11—79, (1979).
[9] Nivat, M., Behaviours of synchronized systems of processes, L.I.T.P. Report n° 81—64, Université de Paris 7, (1981).
[10] Ogden, W. F., W. E. Riddle and W. C. Rounds, Complexity of expressions allowing concurrency, 5th ACM Symp. on Principles of Programming Languages, 185—194, (1978).
[11] Rozenberg, G. and A. Salomaa, The Mathematical Theory of L Systems, Academic Press, (1980).
[12] Shaw, A. C., Software descriptions with flow expressions, IEEE Trans. Engrg., SE-14, 242—254, (1978).

*(Received Dec. 27, 1985)*

# A note on $\alpha_0^*$-products of aperiodic automata

Z. Ésik and J. Virágh

One of the most celebrated results in the field of compositions of automata is the Krohn—Rhodes decomposition theorem. A detailed presentation can be found e.g. in [1]. It has, sometimes implicitly, inspired a great deal of research on various notions of compositions culminating in a series of interesting papers. For references and most recent results, see [2].

The system given by the Krohn—Rhodes theorem has a peculiar lack of symmetry. While it contains all group-like automata on simple groups, all aperiodic automata are in the meantime realized with cascade compositions of a single aperiodic automaton, the two state identity-reset automaton **U**.

If we want to realize a subclass of permutation automata we need exactly those simple groups which are divisors of characteristic groups of automata from the given subclass. Consequently, there is a continuum of different subclasses of permutation automata closed under cascade compositions, subautomata and homomorphic images. On the other hand, if we are given a subclass of aperiodic automata, we do not need the whole strength of **U** either. The reason is that there are numerous subclasses of aperiodic automata closed under cascades, subautomata and homomorphic images. In this note we are going to show that the exact number of these subclasses is continuum even for $\alpha_0^*$-products. The notion of the $\alpha_0^*$-product due to F. Gécseg in [3] is an abstract generalization of the cascade composition.

Although we are using standard automata theoretic concepts we intend to give a very brief account on the notions and notations to be followed throughout the paper.

$N$ and $P$ denote the set of all positive natural numbers and the set of primes $p \neq 1$ in $N$, respectively. We set $N_0 = N \cup \{0\}$.

$X^*$ is the free monoid with identity $\lambda$ generated by a set $X$. We write $u < v$ to mean that $u$ is a proper prefix of $v$, i.e., $u < v$ if and only if there exists a word $u_1 \in X^* - \{\lambda\}$ so that $uu_1 = v$.

Take an ordinary finite automaton $\mathbf{A} = (A, X, \delta)$ with state set $A$, input set $X$ and transition function $\delta: A \times X \to A$. (We use the same $\delta: A \times X^* \to A$ for the usual extension of $\delta$.)

The characteristic semigroup of $\mathbf{A}$ is the factor semigroup $X^*/\varrho_{\mathbf{A}}$ where the congruence $\varrho_{\mathbf{A}}$ is defined by $u \varrho_{\mathbf{A}} v$ if and only if $\delta(a, u) = \delta(a, v)$ for all $a \in A$. An automaton $\mathbf{A}$ is said to be aperiodic if $X^*/\varrho_{\mathbf{A}}$ contains only trivial subgroups.

Next we recall the notion of the $\alpha_0^*$-product from [3]. Let $\mathbf{A}_j = (A_j, X_j, \delta_j)$, $1 \leqq j \leqq n$, be arbitrary automata, $X$ a finite nonvoid set and take a system of feedback functions $\varphi_j\colon A_1\times\ldots\times A_{j-1}\times X\to X_j^*$, $1\leqq j\leqq n$. The $\alpha_0^*$-product $\mathbf{A}_1\times\ldots\times\mathbf{A}_n[X,\varphi]$ of these automata $\mathbf{A}_j$ with respect to $X$ and $\varphi$ is $(A_1\times\ldots\times A_n, X, \delta)$, where

$$\delta\big((a_1,\ldots,a_n),x\big)=\big(\delta_1(a_1,u_1),\ldots,\delta_n(a_n,u_n)\big),$$
$$u_j=\varphi_j(a_1,\ldots,a_{j-1},x)$$

for all $a_1,\ldots,a_n\in A$, $x\in X$, $1\leqq j\leqq n$. We put for an arbitrary class $\mathscr{K}$ of automata

$\mathbf{P}^*_{\alpha_0}(\mathscr{K})$: all $\alpha_0^*$-products of automata from $\mathscr{K}$,
$\mathbf{S}(\mathscr{K})$ : all subautomata of automata from $\mathscr{K}$,
$\mathbf{H}(\mathscr{K})$ : all homomorphic images of automata from $\mathscr{K}$.

Let $\mathscr{K}$ be a class of automata. $\mathscr{K}$ is called closed under $\mathbf{H}$, $\mathbf{S}$ and $\mathbf{P}^*_{\alpha_0}$ if $\mathbf{H}(\mathscr{K})$, $\mathbf{S}(\mathscr{K})$ and $\mathbf{P}^*_{\alpha_0}(\mathscr{K})$ are all subclasses of $\mathscr{K}$. Given $\mathscr{K}$, $\mathbf{HSP}^*_{\alpha}(\mathscr{K})$ is the smallest class containing $\mathscr{K}$ and closed under $\mathbf{H}$, $\mathbf{S}$ and $\mathbf{P}^*_{\alpha_0}$. Thus, $\mathscr{K}$ is closed under $\mathbf{H}$, $\mathbf{S}$ and $\mathbf{P}^*_{\alpha_0}$ if and only if $\mathbf{HSP}^*_{\alpha_0}(\mathscr{K})\subseteqq\mathscr{K}$. The Krohn—Rhodes theorem gives that the class of all aperiodic automata is closed under $\mathbf{H}$, $\mathbf{S}$ and $\mathbf{P}^*_{\alpha_0}$.

We now define a special aperiodic automaton $\mathbf{A}_p$ for every $p\in P$.

$$\mathbf{A}_p=\big(\{0,1,\ldots,2p\},\ \{x,y\},\ \delta_p\big),$$

$$\delta_p(i,x)=\begin{cases} i+1 & \text{if } 1\leqq i\leqq p,\\ 0 & \text{otherwise,}\end{cases}$$

$$\delta_p(i,y)=\begin{cases} i+1 & \text{if } p+1\leqq i\leqq 2p-1,\\ 1 & \text{if } i=2p,\\ 0 & \text{in all other cases.}\end{cases}$$

The following statement enlists some pecularities of $\mathbf{A}_p$. In this statement $u$ and $v$ denote arbitrary words in $\{x,y\}^*$, $i$ is a natural number $1\leqq i\leqq 2p$, and

$$w_i=\begin{cases} x^{p-i+1}y^px^{i-1} & \text{if } 1\leqq i\leqq p,\\ y^{2p-i+1}x^py^{i-p-1} & \text{if } p+1\leqq i\leqq 2p.\end{cases}$$

**Claim.**

(1a) $\delta_p(i,u)\neq 0$ if and only if $u=w_i^ku_1$ where $u_1<w_i$ and $k\in N_0$.
(1b) $\delta_p(i,u)=i$ if and only if $u=w_i^k$ for an integer $k\in N_0$.
(2a) $\delta_p(i,w_j)=0$ if $i\neq j$.
(2b) If $u_1$ contains both $x$ and $y$, $u_1<w_j$ then

$$\delta_p(i,u_1^2)=0 \quad \text{for every } i\neq j.$$

(3) $\delta_p(i,u^k)=i$ implies $\delta_p(i,u)=i$ for every $k\in N$.
(4) If $q\in N$, $q\neq 1$ and $q\neq p$ then $\delta_p(i,u^qv^q)=i$ implies $\delta_p(i,u)=\delta_p(i,v)=i$.

*Proof.* (1a), (1b), (2a)&(2b): Observe that there exists a single cycle

$$1\xrightarrow{x}2\xrightarrow{x}\ldots\xrightarrow{x}p\xrightarrow{x}p+1\xrightarrow{y}p+2\xrightarrow{y}\ldots\xrightarrow{y}2p\xrightarrow{y}1$$

which does not pass through the 'trapped state' 0. Thus we have to move along this cycle if we want to avoid that state.

(3) By (1a) we have $u=w_i^m u_1$, $u_1<w_i$, $m\in N_0$. Assume that $u_1\neq\lambda$. Since $\delta_p(i, u_1 w_i)=0$ by (2a) we obtain $m=0$. From (1b) it follows that $u_1^k=w_i^l$ for some $l\in N$. However, this is clearly impossible, thus we have $u_1=\lambda$. We can conclude using (1b).

(4) We get $u=w_i^m u_1$ and $v=w_j^l v_1$ from (1a), where $u_1<w_i$, $v_1<w_j$, $m, l\in N_0$ and $j=\delta_p(i, u^q)$. If $i=j$ we are ready by (3). Supposing $i\neq j$ we have $u_1\neq\lambda$, $v_1\neq\lambda$ by (1b). If $m>0$, then by (2a) $\delta_p(i, u_1 w_i)=0$, which would imply $\delta_p(i, u^q v^q)=0$. Thus, $m=0$, and similarly, $l=0$. By (2b), $u_1$ cannot contain both $x$'s and $y$'s, and the same holds for $v_1$. By (1b), $u^q v^q=u_1^q v_1^q=w_i^l$ for some $l\in N$, or even, $l=1$, and $i=1$ or $i=p+1$. Suppose that $i=1$, the case $i=p+1$ can be handled likewise. Then $u_1=x^r$ so that $rq=p$. Since $p$ is a prime and $q\neq 1$, $q\neq p$, this gives a contradiction.

As an immediate consequence of Claim (3) we get the following:

**Corollary.** $\mathbf{A}_p$ is aperiodic for every $p\in P$. (Or even, since we did not use the fact that $p$ is a prime in the proof of Claim (3), $\mathbf{A}_p$ is aperiodic for every natural number $p\in N$)

**Theorem.** The class of all aperiodic automata contains a continuum of different subclasses closed under **H**, **S** and $\mathbf{P}_{\alpha_0}^*$.

*Proof.* Let $Q$ be a non-void subset of $P$. Put $\mathcal{K}_Q=\{\mathbf{A}_q | q\in Q\}$. We show that for $q\in P$, $\mathbf{A}_q\in\mathbf{HSP}_{\alpha_0}^*(\mathcal{K}_Q)$ only if $q\in Q$.

Supposing $\mathbf{A}_q\in\mathbf{HSP}_{\alpha_0}^*(\mathcal{K}_Q)$ there is an $\alpha_0^*$-product $\mathbf{B}=\mathbf{A}_{p_1}\times\ldots\times\mathbf{A}_{p_l}[\{x, y\}, \varphi]$ of automata from $\mathcal{K}_Q$ such that $\mathbf{A}_q$ is a homomorphic image of a subautomaton $\mathbf{C}=(C, \{x, y\}, \delta)$ of $\mathbf{B}$ under a homomorphism $h: C\to A_q$. We can choose $l$ minimal with this property. Further, it can be assumed that no subautomata of $\mathbf{C}$ other than itself can be mapped homomorphically onto $\mathbf{A}_q$. Let $c\in h^{-1}(1)$. Obviously, $c$ generates $\mathbf{C}$, and $\delta(c, (x^q y^q)^m)=c$ for some $m\in N$. Since the class of all aperiodic automata is closed under $\alpha_0^*$-products and subautomata we have $\delta(c, x^q y^q)=c$ as well.

Let us now suppose to the contrary $q\notin Q$. Put $c=(i_1, \ldots, i_l)$, $u=\varphi_1(x)$, $v=\varphi_1(y)$. (Observe that $l>0$.) From the definition of the $\alpha_0^*$-product we then have $\delta_{p_1}(i_1, u^q v^q)=i_1$ which by Claim (4) gives $\delta_{p_1}(i_1, u)=\delta_{p_1}(i_1, v)=i_1$. Since $c$ generates $\mathbf{C}$ it follows that the only state of $\mathbf{A}_{p_1}$ appearing as the first component of a state of $\mathbf{C}$ is $i_1$. However, this implies that a subautomaton of an $\alpha_0^*$-product $\mathbf{A}_{p_2}\times\ldots\times\mathbf{A}_{p_l}$ $[\{x, y\}, \psi]$ can be mapped homomorphically onto $\mathbf{A}_q$ contradicting the minimality of $l$.

A. JÓZSEF UNIVERSITY
BOLYAI INSTITUTE
ARADI V. TERE 1.
SZEGED, HUNGARY
H—6720

## References

[1] ARBIB, M. A. (Ed.), Algebraic Theory of Machines, Languages and Semigroups, Academic Press, 1968.
[2] GÉCSEG, F., Products of automata, Springer-Verlag, 1986.
[3] GÉCSEG, F., On products of abstract automata, Acta Sci. Math. 38 (1976), 21—43.

*(Received Jan. 27, 1986)*

# Loop products and loop-free products

Z. Ésik

We introduce loop products of automata and show that, in the presence of input signs inducing the identity state transformation, loop products followed by loop-free products, (i.e. $\alpha_0$-products) are just as stong as the most general product. See [3] for notations and unexplained concepts. Most recent results on $\alpha_0$-products can be found in [2].

Take a $g^*$-product $\mathbf{A}=\mathbf{A}_1\times\ldots\times\mathbf{A}_n(X,\varphi)$ of automata $\mathbf{A}_t=(A_t, X_t, \delta_t)$, $t=1, \ldots, n$, $n\geqq 0$. We call $\mathbf{A}$ an $l^*$-product (i.e. generalized loop product) if for every $t>1$, $\varphi_t(a_1, \ldots, a_n, x)$ $\big((a_1, \ldots, a_n)\in A_1\times\ldots\times A_n$, $x\in X\big)$ only depends on $x$ and $a_{t-1}$, and $\varphi_1$ only depends on $a_n$ and $x$. In the special case that $\varphi_t(a_1, \ldots, a_n, x)\in$ $\in X\cup\{\lambda\}$ $\big(\varphi_t(a_1, \ldots, a_n, x)\in X\big)$ we speak about an $l^\lambda$-product ($l$-product, i.e., loop product).

Let $\mathbf{K}$ be a class of automata. We put

$\mathbf{P}_l^*(\mathbf{K})$: all $l^*$-products of automata from $\mathbf{K}$,
$\mathbf{P}_l^\lambda(\mathbf{K})$: all $l^\lambda$-products of automata from $\mathbf{K}$,
$\mathbf{P}_l(\mathbf{K})$: all $l$-products of automata from $\mathbf{K}$.

Further, we write $\mathbf{P}_{1l}^*(\mathbf{K})$ $\big(\mathbf{P}_{1l}^\lambda(\mathbf{K}), \mathbf{P}_{1l}(\mathbf{K})\big)$ for the class of all $l^*$-products ($l^\lambda$-products, $l$-products) with a single factor of automata from $\mathbf{K}$.

Our result is the following statement.

**Theorem.** $\mathbf{HSP}_{\alpha_0}\mathbf{P}_l^\lambda(\mathbf{K})=\mathbf{HSP}_{\alpha_0}\mathbf{P}_l^*(\mathbf{K})=\mathbf{HSP}_g^*(\mathbf{K})$ for every class $\mathbf{K}$.

*Proof.* The inclusions from left to right are obvious. To see that

$$\mathbf{HSP}_g^*(\mathbf{K}) \subseteqq \mathbf{HSP}_{\alpha_0} P_l^\lambda(\mathbf{K}),$$

by $\mathbf{P}_{\alpha_0}^\lambda(\mathbf{K}) = \mathbf{P}_{\alpha_0}^\lambda\mathbf{P}_l^\lambda(\mathbf{K})$, it suffices to show that $\mathbf{HSP}_g^*(\mathbf{K})\subseteqq\mathbf{HSP}_{\alpha_0}^\lambda\mathbf{P}_l^\lambda(\mathbf{K})$.

If $\mathbf{K}$ contains only monotone automata, then $\mathbf{HSP}_g^*(\mathbf{K})=\mathbf{ISP}_{\alpha_0}^\lambda(\mathbf{K})$ by the proof of Theorem 4 in [3] and the inclusion holds. Suppose that $\mathbf{K}$ contains an automaton which is not cycle-free. We claim that $\mathbf{HSP}_{\alpha_0}^\lambda\mathbf{P}_l^\lambda(\mathbf{K})$ is the class of all automata. To this, by Corollary 2 in [3], we have to show the following:

(i) $\mathbf{P}_l^\lambda(\mathbf{K})$ is not counter-free.

(ii) $\mathbf{A}_0\in\mathbf{HSP}_{\alpha_0}^\lambda\mathbf{P}_l^\lambda(\mathbf{K})$.

(iii) For every finite simple group $G$ there exists an automaton $\mathbf{A}\in\mathbf{P}_l^\lambda(\mathbf{K})$ such that $G$ is a homomorphic image of a subgroup of $S(\mathbf{A})$.

*Proof of (i).* There is an automaton $\mathbf{A} \in \mathbf{K}$ containing a nontrivial cycle, i.e., a cycle with length $n > 1$. Obviously, a counter with length $n$ is in $\mathbf{SP}_{1l}(\mathbf{K})$, therefore, $\mathbf{P}_l^\lambda(\mathbf{K})$ is not counter-free.

*Proof of (ii).* By Lemma 3 in [3], $\mathbf{A}_0 \in \mathbf{HSP}_{\alpha_1}^\lambda(\mathbf{K})$. However,

$$\mathbf{HSP}_{\alpha_1}^\lambda(\mathbf{K}) = \mathbf{HSP}_{\alpha_0}^\lambda \mathbf{P}_{1\alpha_1}^\lambda(\mathbf{K}) = \mathbf{HSP}_{\alpha_0}^\lambda P_{1l}^\lambda(\mathbf{K}) \subseteq \mathbf{HSP}_{\alpha_0}^\lambda \mathbf{P}_l^\lambda(\mathbf{K}).$$

*Proof of (iii).* We show that for every integer $n \geqq 3$ there are an automaton $\mathbf{B} \in (B, Y, \delta') \in \mathbf{P}_l^\lambda(\mathbf{K})$ and a subset $B' = \{b_1, ..., b_n\} \subseteq B$ so that every such permutation of $B'$ which fixes $b_n$ can be induced by a word in $Y^*$. Of course, it is enough to prove for transpositions $(b_s b_{s+1})$ with $1 \leqq s \leqq n-2$.

Let $\mathbf{A} \in (A, X, \delta) \in \mathbf{K}$ be an automaton containing a nontrivial cycle, i.e. a sequence of states $0, 1, ..., p-1$ $(p \geqq 2)$ and input signs $x_1, ..., x_{p-1}, x_0$ with $\delta(0, x_1) = 1, ..., \delta(p-2, x_{p-1}) = p-1$, $\delta(p-1, x_0) = 0$. In the case that $p = 2$ the result follows by the proof of Theorem 2 in [1] (observation due to J. Virágh). Hence we assume $p > 2$.

Define $\mathbf{B}$ to be the $l^\lambda$-power $\mathbf{A}^n(Y, \varphi)$ with

$$Y = \{y(k, i, j) | 1 \leqq k \leqq n, \quad 0 \leqq i, j \leqq p-1\}$$

and

$$\varphi_t(a, y(k, i, j)) = \begin{cases} x_j & \text{if} \quad t = k \quad \text{and} \quad a = i, \\ \lambda & \text{otherwise}, \end{cases}$$

$$\text{where} \quad 1 \leqq t \leqq n, \quad a \in A, \quad y(k, i, j) \in Y.$$

Put

$$b_t = 0^{t-1} 1 0^{n-t},$$

$t = 1, ..., n$. (We use the shorthand $a_1 ... a_n$ for the elements of $B$.) Fix an integer $s$, $1 \leqq s \leqq n-2$. In five steps we shall construct a word $u = u_1 ... u_5 \in Y^*$ such that

$$\delta'(b_t, u) = b_t \quad \text{if} \quad t \neq s, \ s+1,$$

$$\delta'(b_s, u) = b_{s+1},$$

$$\delta'(b_{s+1}, u) = b_s.$$

(The construction is indicated in the Figure for $p = 3$, $n = 6$ and $s = 3$. Blank entries are meant 0.)

*Step 1.*

$$\begin{aligned} u_1 = {} & y(s+1, 1, 1) ... y(s+1, 1, p-1) \cdot \\ & y(s+2, p-1, 1) ... y(s+2, p-1, p-1) \cdot \\ & \vdots \\ & y(n, p-1, 1) ... y(n, p-1, p-1) \cdot \\ & y(1, p-1, 1) ... y(n, p-1, p-1) \cdot \\ & \vdots \\ & y(s-1, p-1, 1) ... y(s-1, p-1, p-1) \cdot \\ & y(s, p-1, 2) ... y(s, p-1, p-1). \end{aligned}$$

We have

$$\delta(b_t, u_1) = b_t \quad \text{if} \quad t \neq s,$$
$$\delta(b_s, u_1) = (p-1)^n.$$

*Step 2.*

$$u_2 = y(s+3, 1, 1) \dots y(n, 1, 1) \cdot$$
$$y(1, 1, 1) \dots y(s, 1, 1).$$

We have

$$\delta(b_1, u_1 u_2) = 1^{s-1} 100^{n-s-1}$$
$$\delta(b_2, u_1 u_2) = 01^{s-2} 100^{n-s-1}$$
$$\vdots$$
$$\delta(b_{s-1}, u_1 u_2) = 0^{s-2} 1100^{n-s-1}$$
$$\delta(b_s, u_1 u_2) = (p-1)^{s-1}(p-1)(p-1)(p-1)^{n-s-1}$$
$$\delta(b_{s+1}, u_1 u_2) = 0^{s-1} 010^{n-s-1}$$
$$\delta(b_{s+2}, u_1 u_2) = 1^{s-1} 101^{n-s-1}$$
$$\delta(b_{s+3}, u_1 u_2) = 1^{s-1} 1001^{n-s-2}$$
$$\vdots$$
$$\delta(b_n, u_1 u_2) = 1^{s-1} 100^{n-s-2} 1.$$

*Step 3.*

$$u_3 = y(s+1, 0, 2) \dots y(s+1, 0, p-1)\, y(s+1, 0, 0) \cdot$$
$$y(s+1, p-1, 0)\, y(s+1, p-1, 1)\, y(s, 0, 1)\, y(s, p-1, 0).$$

We have

$$\delta(b_t, u_1 u_2 u_3) = (b_t, u_1 u_2), \quad t \neq s, s+1,$$
$$\delta(b_s, u_1 u_2 u_3) = (p-1)^{s-1} 01 (p-1)^{n-s-1}$$
$$\delta(b_{s+1}, u_1 u_2 u_3) = 0^{s-1} 100^{n-s-1}.$$

*Step 4.*

$$u_4 = y(s-1, p-1, 0) \dots y(1, p-1, 0) \cdot$$
$$y(n, p-1, 0) \dots y(s+3, p-1, 0)\, y(s+2, 1, 0).$$

We have

$$\delta(b_t, u_1 u_2 u_3 u_4) = \delta(b_t, u_1 u_2 u_3), \quad t \neq s,$$
$$\delta(b_s, u_1 u_2 u_3 u_4) = 0^{s-1} 010^{n-s-1}.$$

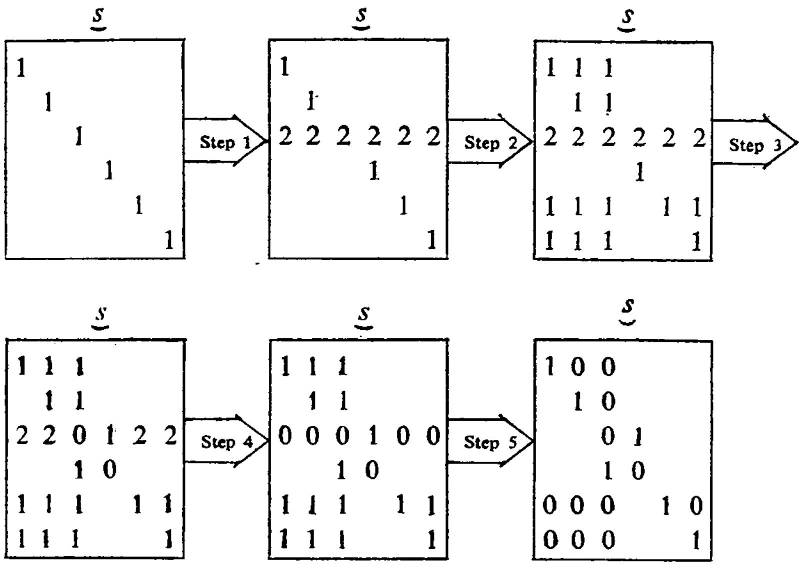
*Step* 5.

$$\begin{aligned}u_5 = {} & y(s,1,2)\ldots y(s,1,p-1)\,y(s,1,0)\cdot \\ & \vdots \\ & y(1,1,2)\ldots y(1,1,p-1)\,y(1,1,0)\cdot \\ & y(n,1,2)\ldots y(n,1,p-1)\,y(n,1,0)\cdot \\ & \vdots \\ & y(s+3,1,2)\ldots y(s+3,1,p-1)\,y(s+3,1,0).\end{aligned}$$

We obtained:

$$\delta(b_t, u) = b_t, \quad t \neq s,\ s+1,$$
$$\delta(b_s, u) = b_{s+1},$$
$$\delta(b_{s+1}, u) = b_s.$$

This ends the proof.



BOLYAI INSTITUTE
A. JÓZSEF UNIVERSITY
ARADI V. TERE 1
SZEGED, HUNGARY
H—6720

## References

[1] Dömösi, P. and Imreh, B., On $\nu_i$-products of automata, Acta Cybernet., 6 (1983), 149—162.
[2] Ésik, Z. and Dömösi, P., Complete classes of automata for the $\alpha_0$-product, Theoret. Comput. Sci., 47 (1986), 1—14.
[3] Ésik, Z. and Virágh, J., On products of automata with identity, Acta Cybernet., 7 (1986), 299—311.

*(Received Feb. 10, 1986)*.

# Results on compositions of deterministic root-to-frontier tree transformations

Z. FÜLÖP and S. VÁGVÖLGYI

## Introduction

In this paper we examine the class of deterministic root-to-frontier tree transformations ($\mathcal{DR}$) and some of its usual subclasses such as linear, nondeleting, homomorphism and so on. We present some equalities and inclusions between the compositions of different classes and, as an application, show that $\mathcal{DR}^2=\mathcal{DR}^n$ for each $n\geqq 2$.

We also study all the classes which can be written in the form $\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n$ where each $\mathcal{K}_i$ is $\mathcal{DR}$ or one of its subclasses. We pick out a finite number of these classes and show that every class $\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n$ either equals to one of them or has a rather special form.

## 1. Notions and notations

For an arbitrary set $Y$, we denote by $|Y|$ and $\mathcal{P}(Y)$ the cardinality and the power set of $Y$, respectively. If $Y$ is a singleton, then we identify it with its unique element. $Y^*$ is the free monoid generated by $Y$ with empty word $\lambda$.

The set of nonnegative integers is denoted by $N$. For every $n\in N$, $[n]$ denotes the set $\{1, \ldots, n\}$, especially $[0]=\emptyset$.

By a ranked alphabet we mean an ordered pair $(F, \nu)$ where $F$ is a finite set and $\nu\colon F\to N$ is the arity function. Elements of $F$ are considered as operational symbols, more exactly, if $f\in F$ and $\nu(F)=n$ then $f$ is an $n$-ary operational symbol. We use the notation $F=\bigcup_{n\in N} F_n$ where the sets $F_n=\nu^{-1}(n)$ are pairwise disjoint.

Now let $F$ be a ranked alphabet and $Y$ a set. The set of all terms or trees over $Y$ of type $F$ is defined as the smallest set $T_F(Y)$ satisfying

(a) $Y\cup F_0\subseteqq T_F(Y)$ and,
(b) $f(p_1, \ldots, p_n)\in T_F(Y)$ whenever $f\in F_n$ $(n\in N)$ and $p_i\in T_F(Y)$ $(i\in[n])$.

If $Y=\emptyset$ then $T_F(Y)$ is simply written as $T_F$.

We define the height $h(p)\in N$, frontier $fr(p)\subseteqq Y^*$ and the set of subtrees $sub(p)\subseteqq T_F(Y)$ of a tree $p\in T_F(Y)$ by induction:

(a) if $p \in F_0$ then $h(p)=0$, $fr(p)=\lambda$ and $sub(p)=\{p\}$;
(b) if $p \in Y$ then $h(p)=0$, $fr(p)=p$ and $sub(p)=\{p\}$;
(c) if $p=f(p_1, \ldots, p_n)$ then $h(p)=1+max\ \{h(p_i)|i \in [n]\}$, $fr(p)=fr(p_1)\ldots fr(p_n)$ and $sub(p)=\bigcup_{i \in [n]} sub\,(p_i) \cup \{p\}$.

We shall need a countably infinite set $X=\{x_1, x_2, \ldots\}$, elements of which are considered as auxiliary variables. The set of the first $n$ elements $x_1, \ldots, x_n$ of $X$ is denoted by $X_n$.

Letting $Y=X_n$ we have the set $T_F(X_n)$. Here, the elements of $X_n$ can be used to point out places in the frontier of a tree $p \in T_F(X_n)$. There is a distinguished subset $\hat{T}_F(X_n)$ of $T_F(X_n)$ defined as follows: $p \in \hat{T}_F(X_n)$ iff $p \in T_F(X_n)$ and $fr(p)$ is a permutation of $X_n$, in other words, each element of $X_n$ appears exactly once in $p$.

Now let $p \in T_F(X_n)$ and $y_1, \ldots, y_n \in Y$. We denote by $p(y_1, \ldots, y_n)$ the tree obtained by substituting all the occurrences of $x_i$ in $p$ by $y_i$ for each $i \in [n]$. Note that $p(y_1, \ldots, y_n)$ is an element of $T_F(Y)$.

By a tree transformation $\tau$ we mean a relation from $T_F$ to $T_G$ where $F$ and $G$ are arbitrary ranked alphabets, that is we have $\tau \subseteq F_F \times T_G$. In this way, the identical relation $\iota_F=\{(p, p)|p \in T_F\}$ is clearly a tree transformation. The class of all identical tree transformations is denoted by $\mathscr{I}$. The restriction $\tau|T$ of $\tau$ to a subset $T$ of $T_F$ is defined by

$$\tau|T = \{(p, q)|(p, q) \in \tau \quad \text{and} \quad p \in T\}.$$

For any tree transformations $\tau \subseteq T_F \times T_G$ and $\sigma \subseteq T_G \times T_H$ the domain ($dom\ \tau$) range ($range\ \tau$) of $\tau$ and the composition $(\tau \circ \sigma)$ of $\tau$ and $\sigma$ are defined as usual in the case of relations.

Let $\mathscr{K}_1$ and $\mathscr{K}_2$ be two classes of tree transformations. By their composition $\mathscr{K}_1 \circ \mathscr{K}_2$ we mean $\{\tau_1 \circ \tau_2 | \tau_1 \in \mathscr{K}_1 \text{ and } \tau_2 \in \mathscr{K}_2\}$. For any class $\mathscr{K}$ of tree transformations and $n \in N$ we put $\mathscr{K}^n=\mathscr{K}$ if $n=1$ and $\mathscr{K}^n=\mathscr{K}^{n-1} \circ \mathscr{K}$ if $n>1$. We say that $\mathscr{K}$ is closed under composition if $\mathscr{K}^2 \subseteq \mathscr{K}$ holds. If $\mathscr{I} \subseteq \mathscr{K}$, as with most of the reasonable classes $\mathscr{K}$, then obviously $\mathscr{K}^2 \subseteq \mathscr{K}$ iff $\mathscr{K}^2=\mathscr{K}$.

In this paper we are interested only in tree transformations which can be induced by deterministic root-to-frontier tree transducers.

By a deterministic root-to-frontier tree transducer (or shortly *DR* transducer) we mean a system

$$\mathfrak{A} = (F, A, G, P, a_0), \quad \text{where} \tag{1}$$

(a) $F$ and $G$ are ranked alphabets;
(b) $A$ is a ranked alphabet — disjoint with $F$ and $G$ — consisting of unary operational symbols, the state set of $\mathfrak{A}$;
(c) $a_0$ is a distinguished element of $A$, the initial state;
(d) $P$ is a finite set of productions (or rewriting rules) of the form

$$af(x_1, \ldots, x_m) \to q, \tag{2}$$

where $a \in A$, $m \geq 0$, $f \in F_m$ and $q \in T_G(A \times X_m)$. To guarantee $\mathfrak{A}$ a deterministic behaviour, any two different productions of $P$ are required to have different left-hand sides.

Throughout the paper terms of the form $a(p)$ ($a \in A$ and $p$ is a term) are written simply as $ap$. If we need to specify a production (2) in a more detailed form, then we can write (2) as

$$af(x_1, \ldots, x_m) \to \bar{q}(a_1 x_{i_1}, \ldots, a_n x_{i_n}) \tag{3}$$

for a suitable $n\geqq 0$, $\bar{q}\in\hat{T}_G(X_n)$, $a_j\in A$, $x_{i_j}\in X_m$ $(j\in[n])$, or as

$$af(x_1, \dots, x_m)\to\bar{q}(a_{1_1}x_1, \dots, a_{1_{n_1}}x_1, \dots, a_{m_1}x_m, \dots, a_{m_{n_m}}x_m) \tag{4}$$

for some $n_i\geqq 0$, $a_{i_j}\in A$, $(i\in[m], j\in[n_i])$ and $\bar{q}\in\hat{T}_G(X_n)$ where $n=n_1+\ldots+n_m$.

Productions of $P$ can be used to transform (or rewrite) terms of $A\times T_F$ to terms of $T_G$, by defining the relation $\underset{\mathfrak{A}}{\Rightarrow}$ (called direct derivation) on the set $T_G(A\times T_F(X))$ in the following way: for $p, q\in T_G(A\times T_F(X))$ we say that $p\underset{\mathfrak{A}}{\Rightarrow}q$ iff $q$ can be obtained from $p$ by replacing an occurrence of a subtree $af(p_1, \dots, p_m)$ of $p$ by the tree $\bar{q}(a_1p_{i_1}, \dots, a_np_{i_n})$ provided the rule (3) is in $P$. Denoting the reflexive-transitive closure (i.e. the iterated application) of the direct derivation by $\overset{*}{\underset{\mathfrak{A}}{\Rightarrow}}$, the tree transformation $\tau_{\mathfrak{A}(a)}$ induced by $\mathfrak{A}$ with state $a\in A$ is defined by

$$\tau_{\mathfrak{A}(a)}=\{(p, q)|p\in T_F,\ q\in T_G \text{ and } ap\overset{*}{\underset{\mathfrak{A}}{\Rightarrow}}q\}.$$

By the tree transformation $\tau_{\mathfrak{A}}$ induced by $\mathfrak{A}$ we mean $\tau_{\mathfrak{A}(a_0)}$, i.e.,

$$\tau_{\mathfrak{A}}=\{(p, q)|p\in T_F,\ q\in T_G \text{ and } a_0p\overset{*}{\underset{\mathfrak{A}}{\Rightarrow}}q\}.$$

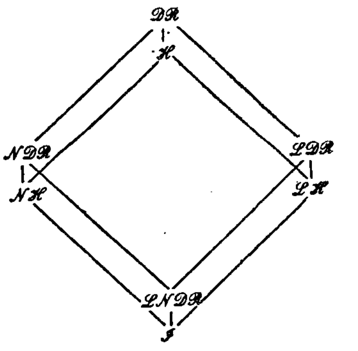
We say that a tree transformation $\tau\subseteq T_F\times T_G$ can be induced by a $DR$ transducer if $\tau=\tau_{\mathfrak{A}}$ for some $DR$ transducer $\mathfrak{A}$.

Next we introduce some restrictions on $DR$ transducers. A $DR$ transducer (1) is totally defined if for each $a\in A$ and $f\in F$ there is a rule (2) in $P$. (1) is called a homomorphism $(H)$ transducer if it is totally defined and has only one state, i.e. $A=\{a_0\}$. Moreover, we say that (1) is

(a) linear $(L)$ if for every rule (4) of $P$ and $i\in[m]$, $n_i\leqq 1$,
(b) nondeleting $(N)$ if for each rule (4) of $P$ and $i\in[m]$, $n_i\geqq 1$,
(c) linear nondeleting $(LN)$ if it is both linear and nondeleting.

The subclasses $L$, $N$ and $LN$ of $H$ transducers are defined in a similar way.

If $K$ is some subclass of the class of all $DR$ transducers defined above, then the class of all tree transformations that can be induced by $K$ transducers is denoted by $\mathcal{K}$. For example $\mathcal{LDR}$ denotes the class of all tree transformations that can be induced by $LDR$ transducers. Finally, we present a diagram showing the inclusion relations among the classes $\mathcal{DR}$, $\mathcal{NDR}$, $\mathcal{LDR}$, $\mathcal{LNDR}$, $\mathcal{H}$, $\mathcal{NH}$, $\mathcal{LH}$, $\mathcal{I}$.



## 2. Equalities and inclusions

By one of the earlier results $\mathcal{DR}$ is not closed under composition (see [4]). This means that, in general, we cannot give a $DR$ transducer $\mathfrak{C}$, for any two $DR$ transducers $\mathfrak{A}$ and $\mathfrak{B}$ such that $\tau_{\mathfrak{C}}=\tau_{\mathfrak{A}}\circ\tau_{\mathfrak{B}}$. However, we can define a $DR$ transducer $\mathfrak{A}\circ\mathfrak{B}$ called the syntactic composition of $\mathfrak{A}$ and $\mathfrak{B}$ with a series of useful properties. This was also stated, in an implicit form, in [3].

**Definition 1.** Let $\mathfrak{A}=(F, A, G, P, a_0)$ and $\mathfrak{B}=(G, B, H, P', b_0)$ be $DR$ transducers. By the syntactic composition of $\mathfrak{A}$ and $\mathfrak{B}$ we mean the $DR$ transducer $\mathfrak{A}\circ\mathfrak{B}=$ $=(F, B\times A, H, P'', (b_0, a_0))$ where $P''$ is defined in the following manner: the rule

$$(b, a)f(x_1, \dots, x_m) \to q'((b_{1_1}, a_1)x_{i_1}, \dots, (b_{1_{v_1}}, a_1)x_{i_1}, \dots,$$
$$(b_{n_1}, a_n)x_{i_n}, \dots, (b_{n_{v_n}}, a_n)x_{i_n})$$

is in $P''$ for some $v_j\in N$, $b_{j_k}\in B$, $(j\in[n], k\in[v_j])$ and $q'\in T_G(X_v)$ $(v=v_1+\dots+v_n)$ if and only if there is a rule (3) in $P$ and a state $b\in B$ with

$$b\bar{q} \overset{*}{\underset{\mathfrak{B}}{\Rightarrow}} q'(b_{1_1}x_1, \dots, b_{1_{v_1}}x_1, \dots, b_{n_1}x_n, \dots, b_{n_{v_n}}x_n).$$

(We let $\mathfrak{B}$ work on $\bar{q}$ as long as it can.)

**Lemma 2.** Under the notations of the above definition, for any $a\in A$, $b\in B$, $p\in T_F$ and $q\in T_H$

$$(\exists r\in T_G)\left(ap \overset{*}{\underset{\mathfrak{A}}{\Rightarrow}} r \wedge br \overset{*}{\underset{\mathfrak{B}}{\Rightarrow}} q\right) \Rightarrow (b, a)p \overset{*}{\underset{\mathfrak{A}\circ\mathfrak{B}}{\Rightarrow}} q. \tag{5}$$

The proof, as usual, can easily be performed by induction on $h(p)$. □

Now we can make the following observations.

(a) We cannot converse (5) because $\mathfrak{B}$ may be deleting, therefore there may be a tree $p\in T_F$ such that $\mathfrak{A}\circ\mathfrak{B}$ can transform $p$ to a tree $q\in T_H$ by deleting some subtree $p'$ of $p$ but $\mathfrak{A}$ can not transform $p'$ with any state $a\in A$. Thus $p$ may be in *dom* $\tau_{\mathfrak{A}\circ\mathfrak{B}}$ but not in *dom* $\tau_{\mathfrak{A}}$ hence not even in *dom* $(\tau_{\mathfrak{A}}\circ\tau_{\mathfrak{B}})$. It can be seen that we can eliminate this problem by requiring $\mathfrak{A}$ to be totally defined (see also in [4]) or $\mathfrak{B}$ to be nondeleting.

(b) Moreover, (5) can also be conversed if $p$ is in *dom* $\tau_{\mathfrak{A}}$ since in this case $\mathfrak{A}$ can always transform $p'$ with some state $a\in A$.

(c) $\mathfrak{A}\circ\mathfrak{B}$ inherits any property $\mathfrak{A}$ and $\mathfrak{B}$ have, where property means one of the following: completely defined, one-state, $L$, $N$, $LN$.

We give a summary of the above observations:

**Lemma 3.** For any $DR$ transducers $\mathfrak{A}$ and $\mathfrak{B}$ the following hold:

(a) if $\mathfrak{A}$ is totally defined or $\mathfrak{B}$ is nondeleting then $\tau_{\mathfrak{A}\circ\mathfrak{B}}=\tau_{\mathfrak{A}}\circ\tau_{\mathfrak{B}}$,

(b) $\tau_{\mathfrak{A}\circ\mathfrak{B}}|\text{dom}\,\tau_{\mathfrak{A}}=\tau_{\mathfrak{A}}\circ\tau_{\mathfrak{B}}$,

(c) if $\mathfrak{A}$ and $\mathfrak{B}$ are $x$ then $\mathfrak{A}\circ\mathfrak{B}$ is also $x$ where $x$=completely defined, one-state, $L$, $N$, $LN$. □

From the above lemma we have a series of equalities some of which were stated in different works.

$$\mathscr{H}\circ\mathscr{DR}=\mathscr{DR} \quad (6) \qquad \mathscr{H}\circ\mathscr{H}=\mathscr{H} \quad (11)$$

$$\mathscr{LH}\circ\mathscr{DR}=\mathscr{DR} \quad (7) \qquad \mathscr{H}\circ\mathscr{NH}=\mathscr{H} \quad (12)$$

$$\mathscr{NH}\circ\mathscr{DR}=\mathscr{DR} \quad (8) \qquad \mathscr{H}\circ\mathscr{LH}=\mathscr{H} \quad (13)$$

$$\mathscr{LH}\circ\mathscr{LDR}=\mathscr{LDR} \quad (9) \qquad \mathscr{NH}\circ\mathscr{H}=\mathscr{H} \quad (14)$$

$$\mathscr{NH}\circ\mathscr{NDR}=\mathscr{NDR} \quad (10) \qquad \mathscr{LH}\circ\mathscr{H}=\mathscr{H} \quad (15)$$

$$\mathscr{NH}\circ\mathscr{NH}=\mathscr{NH} \quad (16)$$

$$\mathscr{LH}\circ\mathscr{LH}=\mathscr{LH} \quad (17)$$

These follow from the fact that an $H(LH, NH)$ transducer is always completely defined.

Moreover, we also obtain

$$\mathscr{DR}\circ\mathscr{NDR}=\mathscr{DR} \quad (18)$$

$$\mathscr{DR}\circ\mathscr{LNDR}=\mathscr{DR} \quad (19)$$

$$\mathscr{DR}\circ\mathscr{NH}=\mathscr{DR} \quad (20)$$

$$\mathscr{NDR}\circ\mathscr{NDR}=\mathscr{NDR} \quad (21)$$

$$\mathscr{NDR}\circ\mathscr{LNDR}=\mathscr{NDR} \quad (22)$$

$$\mathscr{NDR}\circ\mathscr{NH}=\mathscr{NDR} \quad (23)$$

$$\mathscr{LDR}\circ\mathscr{LNDR}=\mathscr{LDR} \quad (24)$$

$$\mathscr{LNDR}\circ\mathscr{NDR}=\mathscr{NDR} \quad (25)$$

$$\mathscr{LNDR}\circ\mathscr{LNDR}=\mathscr{LNDR} \quad (26)$$

Here we used that the second components are all nondeleting.

A frequently quoted equality is

$$\mathscr{H}\circ\mathscr{LDR}=\mathscr{DR} \quad (27)$$

which can be found in [1] and [2]. From the proof of (27), it turns out that we may also declare it in the form

$$\mathscr{NH}\circ\mathscr{LDR}=\mathscr{DR} \quad (28)$$

moreover, if we consider only $H$ transducers we get

$$\mathscr{NH}\circ\mathscr{LH}=\mathscr{H}. \quad (29)$$

By lemma 3, $\mathscr{LH}\circ\mathscr{NH}\subseteq\mathscr{H}$ and it is not difficult to see the conversed inclusion shown by the following lemma.

**Lemma 4.** $\mathscr{H}\subseteq\mathscr{LH}\circ\mathscr{NH}$.

*Proof.* Let us be given an $H$ transducer $\mathfrak{A}=(F, a, G, P, a)$. Obviously, each production can be taken in the following form:

$$af(x_1, ..., x_m) \to q(ax_{i_1}, ..., ax_{i_n}) \tag{30}$$

where $m \geqq 0$, $f \in F_m$, $1 \leqq i_1 < ... < i_n \leqq m$ and $q$ is a suitable tree from $T_G(X_n)$ containing at least one occurrence of $x_j$ for every $j \in [n]$. For each production (30) take a new operational symbol $\bar{f}$ with arity $n$ and put $\bar{F}=\{\bar{f} \mid f \in F\}$. Now we can introduce the $H$ transducers $\mathfrak{B}=(F, b, \bar{F}, P', b)$ and $\mathfrak{C}=(\bar{F}, c, G, P'', c)$ as follows: whenever a production (30) is in $P$ let the productions $bf(x_1, ..., x_m) \to \bar{f}(bx_{i_1}, ..., bx_{i_n})$ and $c\bar{f}(x_1, ..., x_n) \to q(cx_1, ..., cx_n)$ be in $P'$ and $P''$, respectively. Clearly, $\mathfrak{B}$ is an $LH$ and $\mathfrak{C}$ is an $NH$ transducer, moreover the equivalence

$$ap \overset{*}{\underset{\mathfrak{A}}{\Rightarrow}} q \Leftrightarrow (\exists p \in T_{\bar{F}})\bigl(bp \overset{*}{\underset{\mathfrak{B}}{\Rightarrow}} r \wedge cr \overset{*}{\underset{\mathfrak{C}}{\Rightarrow}} q\bigr)$$

can be proved, for each $p \in T_F$ and $q \in T_G$, by induction on $h(p)$. Hence we have our lemma and the equality:

$$\mathcal{LH} \circ \mathcal{NH} = \mathcal{H}. \tag{31}$$

Our next lemma follows from exercise 2 on p. 213 of [3]. This states that $dom\ \tau_{\mathfrak{A}}$ can always be recognized by some $DR$ recognizer (for definition see also [3]) for any $DR$ transducer $\mathfrak{A}$. However, we mention that the following correction is needed in the definition of the $DR$ recognizer in [3]: the realisation of an operational symbol of arity 0 must be considered as a subset of the state set and not as an element of it.

**Lemma 5.** For any given $DR$ transducer $\mathfrak{A}=(F, A, G, P, a_0)$ there exists an $LNDR$ transducer $\mathfrak{A}'=(F, \mathcal{P}(A), F, P', \{a_0\})$ such that $\tau_{\mathfrak{A}'} \cdot = \iota_F | \mathrm{dom}\ \tau_{\mathfrak{A}}$.

*Proof.* Let $P'$ be constructed as follows: for any $B=\{a_1, ..., a_k\} \in \mathcal{P}(A)$, $m \in N$ and $f \in F_m$ the rule $Bf(x_1, ..., x_m) \to f(B_1x_1, ..., B_mx_m)$ is in $P'$ if and only if the next conditions hold:

(a) for each $i \in [k]$ there is a production

$$a_i f(x_1, ..., x_m) \to q_i(a^i_{1_1} x_1, ..., a^i_{1_{n_1}} x_1, ..., a^i_{m_1} x_m, ..., a^i_{m_{n_m}} x_m)$$

in $P$ where $n_1, ..., n_m \geqq 0$ (depend on $i$), $a^i_{j_k} \in A$,

$$(j \in [m], k \in [n_j]),\ q_i \in \hat{T}_G(X_n), \quad (n = n_1 + ... + n_m);$$

(b) $$B_j = \bigcup_{i \in [n]} \{a^i_{j_1}, ..., a^i_{j_{n_j}}\}, \quad j \in [m].$$

Then we can verify the following statement: for any

$$B = \{a_1, ..., a_k\} \in \mathcal{P}(A) \quad \text{and} \quad p \in T_F$$

$$Bp \overset{*}{\underset{\mathfrak{A}'}{\Rightarrow}} p \Leftrightarrow (\forall j \in [k])(\exists q \in T_G)(a_j p \overset{*}{\underset{\mathfrak{A}}{\Rightarrow}} q). \quad \square$$

Now let $\mathfrak{A}$ and $\mathfrak{B}$ be two arbitrary $DR$ transducers. Then, by lemmas 3 and 5 we have

$$\tau_{\mathfrak{A}} \circ \tau_{\mathfrak{B}} = \tau_{\mathfrak{A} \circ \mathfrak{B}} | dom\ \tau_{\mathfrak{A}} = (\iota_F | dom\ \tau_{\mathfrak{A}}) \circ \tau_{\mathfrak{A} \circ \mathfrak{B}} = \tau_{\mathfrak{A}'} \circ \tau_{\mathfrak{A} \circ \mathfrak{B}}$$

from where

$$\mathcal{DR}^2 = \mathcal{LNDR} \circ \mathcal{DR} \tag{32}$$

and, if $\mathfrak{A}$ and $\mathfrak{B}$ are *LDR* transducers then

$$\mathcal{LDR}^2 = \mathcal{LNDR} \circ \mathcal{LDR}. \tag{33}$$

We are ready to prove one of our main results.

**Theorem 6.** For any $n \geqq 2$

$$\mathcal{DR}^n = \mathcal{LNDR} \circ \mathcal{DR} \quad \text{and} \tag{34}$$

$$\mathcal{LDR}^n = \mathcal{LNDR} \circ \mathcal{LDR} \tag{35}$$

*Proof.* We follow an induction on $n$. The case $n=2$ is already proved, the induction step of (34) (and similarly that of (35)) is shown by the following computation:

$$\mathcal{DR}^{n+1} \overset{(32)}{=} \mathcal{LNDR} \circ \mathcal{DR} \circ \mathcal{DR}^{n-1} = \mathcal{LNDR} \circ \mathcal{DR}^n \overset{\text{i.h.}}{=} \mathcal{LNDR} \circ \mathcal{LNDR} \circ \mathcal{DR} \overset{(26)}{=}$$

$$\mathcal{LNDR} \circ \mathcal{DR}. \quad \square$$

**Consequence 7.** For every $n \geqq 2$

$$\mathcal{DR}^n = \mathcal{DR}^2 \quad \text{and} \tag{36}$$

$$\mathcal{LDR}^n = \mathcal{LDR}^2. \tag{37}$$

We shall also need the following result.

**Lemma 8.**

$$\mathcal{DR} \subseteq \mathcal{NDR} \circ \mathcal{LH} \tag{38}$$

*Proof.* Let $\mathfrak{A}=(F, A, G, P, a_0)$ be a *DR* transducer. We construct an *NDR* transducer $\mathfrak{B}$ and an *LH* transducer $\mathfrak{C}$ such that $\tau_{\mathfrak{A}}=\tau_{\mathfrak{B}} \circ \tau_{\mathfrak{C}}$. To this end, consider an arbitrary but fixed order of the productions from $P$ and number them from 1 to $|P|$ in the following form

$$i\colon\ af(x_1, \ldots, x_m) \to \bar{q}(a_{1_1} x_1, \ldots, a_{1_{n_1}} x_1, \ldots, a_{m_1} x_m, \ldots, a_{m_{n_m}} x_m), \tag{39}$$

where

$$n_j \geqq 0,\ a_{j_k} \in A,\ (j \in [m],\ k \in [n_j]) \quad \text{and} \quad \bar{q} \in \hat{T}_G(X_n),$$

$(n=n_1+\ldots+n_m)$. We mention that the symbols used in the specification of the $i$-th production depend on $i$. Now, for each $i \in [|P|]$ and $j \in [m]$ define $u_j$ by

$$u_j = \begin{cases} n_j & \text{if} \quad n_j > 0 \\ 1 & \text{if} \quad n_j = 0 \end{cases}$$

and take a new operational symbol $f_i \notin F$ with arity $u=u_1+\ldots+u_m$. Then construct the *DR* transducer $\mathfrak{B}=(F, A \cup \{b\}, F', P', a_0)$ where

(a) $F'=F\cup\{f_i|i\in|P|\}$;
(b) $b\notin A$ is a new state;
(c) $P'$ is defined as follows: the rule

$$af(x_1,\dots,x_m)\to f_i(b_{1_1}x_1,\dots,b_{1_{u_1}}x_1,\dots,b_{m_1}x_m,\dots,b_{m_{u_m}}x_m) \tag{40}$$

is in $P'$ iff the conditions

(i) the $i$-th production of $P$ is of the form (39) and

(ii) $$b_{j_1},\dots,b_{j_{u_j}}=\begin{cases}a_{j_1},\dots,a_{j_{n_j}} & \text{if} \quad n_j>0\\ b & \text{if} \quad n_j=0\end{cases}\quad j\in[m]$$

hold, moreover the rule $bf(x_1,\dots,x_m)\to f(bx_1,\dots,bx_m)$ is in $P'$ for each $m\ge 0$, $f\in F_m$.

Next, introduce the $H$ transducer $\mathfrak{C}=(F',c,G,P'',c)$ where the rule

$$cf_i(x_1,\dots,x_u)\to\bar q(cx_1,\dots,cx_{n_1},\dots,cx_{u_1+\dots+u_{m-1}+1},\dots,cx_{u_1+\dots+u_{m-1}+n_m}) \tag{41}$$

is in $P''$ iff the $i$-th production of $P$ is (39), moreover, to make $\mathfrak{C}$ totally defined, let the rule $cf(x_1,\dots,x_m)\to q$ be in $P''$ with an arbitrary $q\in T_G(\{c\}\times X_m)$ for each $m\ge 0$, $f\in F_m$.

First note that $\mathfrak{B}$ is nondeleting since $u_j\ge 1$, $(j\in[m])$, $\tau_{\mathfrak{B}(b)}|T_F=\iota_F$ and $\mathfrak{C}$ is linear. To prove $\tau_{\mathfrak{A}}=\tau_{\mathfrak{B}}\circ\tau_{\mathfrak{C}}$ it is enough to show that for each $a\in A$, $p\in T_F$ and $q\in T_G$ the equivalence

$$ap\overset{*}{\underset{\mathfrak{A}}{\Rightarrow}}q \tag{42}$$

if and only if

$$(\exists r\in T_{F'})(ap\overset{*}{\underset{\mathfrak{B}}{\Rightarrow}}r\wedge cr\overset{*}{\underset{\mathfrak{C}}{\Rightarrow}}q) \tag{43}$$

holds. We proceed by induction on $h(p)$.

If $h(p)=0$, that is $p=f\in F_0$, then $af\to q\in P$ iff there exists an $i\in[|P|]$ for which $af\to f_i\in P'$ and $cf_i\to q\in P''$. Now let $h(p)>0$, that is $p=f(p_1,\dots,p_m)$, where $m>0$. Suppose that the production applied at the first step of (42) is (39). Then

$$a_{j_k}p_j\overset{*}{\underset{\mathfrak{A}}{\Rightarrow}}q_{j_k}\quad (j\in[m],\ k\in[n_j]) \tag{44}$$

under some $q_{j_k}\in T_G$ for which $q=\bar q(q_{1_1},\dots,q_{1_{n_1}},\dots,q_{m_1},\dots,q_{m_{n_m}})$ holds. From here, by induction hypothesis we have

$$(\exists r'_{j_k}\in T_{F'})(a_{j_k}p_j\overset{*}{\underset{\mathfrak{B}}{\Rightarrow}}r'_{j_k}\wedge cr'_{j_k}\overset{*}{\underset{\mathfrak{C}}{\Rightarrow}}q_{j_k})\quad (j\in[m],\ k\in[n_j]) \tag{45}$$

moreover, by the construction of $\mathfrak{B}$ and $\mathfrak{C}$, (40) and (41) are in $P'$ and $P''$, respectively. Letting

$$r_{j_1},\dots,r_{j_{u_j}}=\begin{cases}r'_{j_1},\dots,r'_{j_{n_j}} & \text{if} \quad n_j>0\\ p_j & \text{if} \quad n_j=0\end{cases}\quad j\in[m],$$

and taking into consideration that $\tau_{\mathfrak{B}(b)}|T_F=\iota_F$ we get that

$$b_{j_k}p_j\overset{*}{\underset{\mathfrak{B}}{\Rightarrow}}r_{j_k},\quad (j\in[m],\ k\in[u_j])\wedge cr_{j_k}\overset{*}{\underset{\mathfrak{C}}{\Rightarrow}}q_{j_k}\quad (j\in[m],\ k\in[n_j]) \tag{46}$$

from where (43) follows with

$$r = f_i(r_{1_1}, \ldots, r_{1_{u_1}}, \ldots, r_{m_1}, \ldots, r_{m_{u_m}}). \tag{47}$$

Conversely, suppose that $r$ in (43) is of the form of (47). Then the productions used in the first step of the derivations of (43) are (40) and (41), respectively. Therefore (39) is in $P$. Moreover, (46) implies (44), by induction hypothesis, hence we have (42). □

We note that if $\mathfrak{A}$ is linear in the above lemma then so is $\mathfrak{B}$. Hence we also obtain:

**Consequence 9.**

$$\mathcal{L}\mathcal{D}\mathcal{R} \subseteq \mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H}. \tag{48}$$

Applying our last two results we have two further interesting identities.

**Theorem 10.** For each $n \geqq 2$

$$\mathcal{D}\mathcal{R}^n = \mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \tag{49}$$

$$\mathcal{L}\mathcal{D}\mathcal{R}^n = \mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \tag{50}$$

*Proof.* $\mathcal{D}\mathcal{R}^n \overset{(34)}{=} \mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{D}\mathcal{R} \overset{(38)}{\subseteq} \mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \overset{(25)}{=} \mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H}$ and in the same way we get (50). □

By the above results we can easily verify the equality

$$\mathcal{D}\mathcal{R}^2 = \mathcal{N}\mathcal{H}\circ\mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H}, \tag{51}$$

namely, we have $\mathcal{D}\mathcal{R}^2 \overset{(49)}{=} \mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \subseteq \mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \overset{(28)}{=} \mathcal{N}\mathcal{H}\circ\mathcal{L}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \overset{(48)}{\subseteq} \mathcal{N}\mathcal{H}\circ \mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H}\circ\mathcal{L}\mathcal{H} \overset{(17)}{=} \mathcal{N}\mathcal{H}\circ\mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \subseteq \mathcal{D}\mathcal{R}^3 \overset{(36)}{=} \mathcal{D}\mathcal{R}^2$.

The equalities (49), (51), (32) are able to produce the class $\mathcal{D}\mathcal{R}^2$ as a composition of two or three simpler classes of tree transformations. Using them we obtain some additional presentations for the class $\mathcal{D}\mathcal{R}^2$ summarized by the following lemma.

**Lemma 11.**

(a) For any $\mathcal{X} \in \{\mathcal{N}\mathcal{D}\mathcal{R}, \mathcal{D}\mathcal{R}\}$ and $\mathcal{Y} \in \{\mathcal{L}\mathcal{H}, \mathcal{H}, \mathcal{L}\mathcal{D}\mathcal{R}, \mathcal{D}\mathcal{R}\}$

$$\mathcal{X}\circ\mathcal{Y} = \mathcal{D}\mathcal{R}^2 \tag{52}$$

(b) For any $\mathcal{X} \in \{\mathcal{N}\mathcal{H}, \mathcal{N}\mathcal{D}\mathcal{R}\}$, $\mathcal{Y} \in \{\mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}, \mathcal{N}\mathcal{D}\mathcal{R}, \mathcal{L}\mathcal{D}\mathcal{R}, \mathcal{D}\mathcal{R}\}$ and $\mathcal{Z} \in \{\mathcal{L}\mathcal{H}, \mathcal{H}, \mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}, \mathcal{D}\mathcal{R}\}$

$$\mathcal{X}\circ\mathcal{Y}\circ\mathcal{Z} = \mathcal{D}\mathcal{R}^2 \tag{53}$$

(c) $\mathcal{L}\mathcal{D}\mathcal{R}\circ\mathcal{D}\mathcal{R} = \mathcal{D}\mathcal{R}^2$ (54)

(d) For arbitrary $\mathcal{X} \in \{\mathcal{L}\mathcal{N}\mathcal{D}\mathcal{R}, \mathcal{L}\mathcal{D}\mathcal{R}\}$ and $\mathcal{Y} \in \{\mathcal{L}\mathcal{H}, \mathcal{L}\mathcal{D}\mathcal{R}\}$

$$\mathcal{X}\circ\mathcal{Y} = \mathcal{L}\mathcal{D}\mathcal{R}^2 \tag{55}$$

*Proof.* We prove the case (a) only, since (b), (c) and (d) can be verified in the same way applying (51), (32) and (50), respectively.

$$\mathcal{D}\mathcal{R}^2 \overset{(49)}{=} \mathcal{N}\mathcal{D}\mathcal{R}\circ\mathcal{L}\mathcal{H} \subseteq \mathcal{X}\circ\mathcal{Y} \subseteq \mathcal{D}\mathcal{R}^2. \quad \square$$

## 3. On mixed composition of different subclasses

We now investigate the set of all classes of tree transformations being a composition of finitely many ones introduced in Section 1. To be more precise we need some further notions and notations. Let $S=\{\mathcal{DR}, \mathcal{NDR}, \mathcal{LDR}, \mathcal{LNDR}, \mathcal{H}, \mathcal{NH}, \mathcal{LH}\}$ and denote by $[S]$ the set of all classes of tree transformations generated by $S$ with composition, that is

$$[S] = \{\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n | n \geqq 1, \quad \mathcal{K}_i\in S\}.$$

One of the most important questions concerning $[S]$ is that whether $[S]$ is infinite. We know, by consequence 7, that $\mathcal{C}\subseteq\mathcal{DR}^2$ for any $\mathcal{C}\in[S]$, however, in spite of this, $[S]$ may be infinite. In this paper we do not answer this question, instead, we present a theorem which, we hope, gives a deep insight into the structure of $[S]$.

First define the classes $\mathcal{C}_k$ for each $k\geqq 0$ as follows

(a) $\mathcal{C}_0 = \mathcal{LNDR}$

(b) $$\mathcal{C}_{k+1} = \begin{cases} \mathcal{C}_k\circ\mathcal{NH} & \text{if} \quad k=2m \\ \mathcal{C}_k\circ\mathcal{LNDR} & \text{if} \quad k=2m+1. \end{cases} \quad (m \geqq 0)$$

Moreover, we shall use Table 1 in the following sense. Each row and each column of the table is marked by a class of tree transformations. Their composition, in row-column order, is written in the corresponding square of the table. To get the depicted form of this composition, the equalities and inclusions the serial numbers of which appear in the lower part of the square can be used. If no serial number is indicated, then the form of the corresponding composition is meant by definition. For example,

$$\mathcal{LDR}^2\circ\mathcal{NH} \overset{(55)}{=} \mathcal{LNDR}\circ\mathcal{LH}\circ\mathcal{NH} \overset{(31)}{=} \mathcal{LNDR}\circ\mathcal{H}.$$

Now we can prove our last theorem.

**Theorem 12.** There are two finite subsets $S_1$ and $S_2$ of $[S]$ such that for any element $\mathcal{C}$ of $[S]$ one of the following conditions holds

(a) $\mathcal{C}\in S_1$,

(b) there exist a $\mathcal{C}'\in S_2$ and a $k\geqq 0$ such that $\mathcal{C}=\mathcal{C}'\circ\mathcal{C}_k$,

(c) $\mathcal{C}=\mathcal{C}_k$ for some $k\geqq 0$.

*Proof.* Define $S_1$ and $S_2$ by

$$S_1 = S\cup\{\mathcal{DR}^2, \mathcal{LDR}\circ\mathcal{NDR}, \mathcal{LDR}^2, \mathcal{LDR}\circ\mathcal{NH}, \mathcal{H}\circ\mathcal{NDR}, \mathcal{LDR}^2\circ\mathcal{NDR}, \mathcal{LNDR}\circ\mathcal{H}\}$$

and

$$S_2 = \{\mathcal{H}, \mathcal{NH}, \mathcal{LH}, \mathcal{LDR}\circ\mathcal{NH}, \mathcal{LNDR}\circ\mathcal{H}\}.$$

For any $\mathcal{C}\in[S]$ there exists a minimal number $n\geqq 1$ such that $\mathcal{C}=\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n$ for some $\mathcal{K}_i\in S$. We prove the theorem by induction on this number $n$.

If $\mathcal{C}=\mathcal{K}_1$ for some $\mathcal{K}_1\in S$ then, by $S\subseteq S_1$, case (a) holds.

Now let $\mathcal{C}=\mathcal{K}_1\circ\ldots\circ\mathcal{K}_{n+1}$ under a minimal $n\geqq 1$ and some $\mathcal{K}_i\in S$. Then, since our theorem is supposed to hold for $\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n$, three main cases are possible.

Table 1.

| | $\mathcal{DR}$ | $\mathcal{NDR}$ | $\mathcal{LDR}$ | $\mathcal{LNDR}$ | $\mathcal{H}$ | $\mathcal{NH}$ | $\mathcal{LH}$ |
|---|---|---|---|---|---|---|---|
| $\mathcal{DR}$ | $\mathcal{DR}^2$ | $\mathcal{DR}$ (18) | $\mathcal{DR}^2$ (52) | $\mathcal{DR}$ (19) | $\mathcal{DR}^2$ (52) | $\mathcal{DR}$ (20) | $\mathcal{DR}^2$ (52) |
| $\mathcal{NDR}$ | $\mathcal{DR}^2$ (52) | $\mathcal{NDR}$ (21) | $\mathcal{DR}^2$ (52) | $\mathcal{NDR}$ (22) | $\mathcal{DR}^2$ (52) | $\mathcal{NDR}$ (23) | $\mathcal{DR}^2$ (52) |
| $\mathcal{LDR}$ | $\mathcal{DR}^2$ (54) | $\mathcal{LDR}\circ\mathcal{NDR}$ | $\mathcal{LDR}^2$ | $\mathcal{LDR}$ (24) | $\mathcal{LNDR}\circ\mathcal{H}$ (48) (15) | $\mathcal{LDR}\circ\mathcal{NH}$ | $\mathcal{LDR}^2$ (55) |
| $\mathcal{LNDR}$ | $\mathcal{DR}^2$ (32) | $\mathcal{NDR}$ (25) | $\mathcal{LDR}^2$ (33) | $\mathcal{LNDR}$ (36) | $\mathcal{LNDR}\circ\mathcal{H}$ | $\mathcal{C}_1$ | $\mathcal{LDR}^2$ (55) |
| $\mathcal{H}$ | $\mathcal{DR}$ (6) | $\mathcal{H}\circ\mathcal{NDR}$ | $\mathcal{DR}$ (27) | $\mathcal{H}\circ\mathcal{C}_0$ | $\mathcal{H}$ (11) | $\mathcal{H}$ (12) | $\mathcal{H}$ (13) |
| $\mathcal{NH}$ | $\mathcal{DR}$ (8) | $\mathcal{NDR}$ (10) | $\mathcal{DR}$ (28) | $\mathcal{NH}\circ\mathcal{C}_0$ | $\mathcal{H}$ (14) | $\mathcal{NH}$ (16) | $\mathcal{H}$ (29) |
| $\mathcal{LH}$ | $\mathcal{DR}$ (7) | $\mathcal{H}\circ\mathcal{NDR}$ (10) (31) | $\mathcal{LDR}$ (9) | $\mathcal{LH}\circ\mathcal{C}_0$ | $\mathcal{H}$ (15) | $\mathcal{H}$ (31) | $\mathcal{LH}$ (17) |
| $\mathcal{DR}^2$ | $\mathcal{DR}^2$ (36) | $\mathcal{DR}^2$ (18) | $\mathcal{DR}^2$ (52) (36) | $\mathcal{DR}^2$ (19) | $\mathcal{DR}^2$ (52) (36) | $\mathcal{DR}^2$ (20) | $\mathcal{DR}^2$ (52) (36) |
| $\mathcal{LDR}\circ\mathcal{NDR}$ | $\mathcal{DR}^3$ (52) (54) (36) | $\mathcal{LDR}\circ\mathcal{NDR}$ (21) | $\mathcal{DR}^2$ (52) (54) (36) | $\mathcal{LDR}\circ\mathcal{NDR}$ (22) | $\mathcal{DR}^2$ (52) (54) (36) | $\mathcal{LDR}\circ\mathcal{NDR}$ (23) | $\mathcal{DR}^3$ (52) (54) (36) |
| $\mathcal{LDR}^2$ | $\mathcal{DR}^2$ (54) (36) | $\mathcal{LDR}^2\circ\mathcal{NDR}$ | $\mathcal{LDR}^2$ (37) | $\mathcal{LDR}^2$ (24) | $\mathcal{LNDR}\circ\mathcal{H}$ (55) (15) | $\mathcal{LNDR}\circ\mathcal{H}$ (55) (31) | $\mathcal{LDR}^2$ (55) (37) |
| $\mathcal{LDR}\circ\mathcal{NH}$ | $\mathcal{DR}^2$ (8) (54) | $\mathcal{LDR}\circ\mathcal{NDR}$ (10) | $\mathcal{DR}^2$ (28) (54) | $\mathcal{LDR}\circ\mathcal{NH}\circ\mathcal{C}_0$ | $\mathcal{LNDR}\circ\mathcal{H}$ (14) | $\mathcal{LDR}\circ\mathcal{NH}$ (16) | $\mathcal{LNDR}\circ\mathcal{H}$ (29) (48) (15) |
| $\mathcal{H}\circ\mathcal{NDR}$ | $\mathcal{DR}^3$ (52) (6) | $\mathcal{H}\circ\mathcal{NDR}$ (21) | $\mathcal{DR}^3$ (52) (6) | $\mathcal{H}\circ\mathcal{NDR}$ (22) | $\mathcal{DR}^3$ (52) (6) | $\mathcal{H}\circ\mathcal{NDR}$ (23) | $\mathcal{DR}^2$ (52) (6) |
| $\mathcal{LDR}^2\circ\mathcal{NDR}$ | $\mathcal{DR}^2$ (52) (54) (36) | $\mathcal{LDR}^2\circ\mathcal{NDR}$ (21) | $\mathcal{DR}^2$ (52) (54) (36) | $\mathcal{LDR}^2\circ\mathcal{NDR}$ (22) | $\mathcal{DR}^2$ (52) (54) (36) | $\mathcal{LDR}^2\circ\mathcal{NDR}$ (23) | $\mathcal{DR}^2$ (52) (54) (36) |
| $\mathcal{LNDR}\circ\mathcal{H}$ | $\mathcal{DR}^3$ (6) (32) | $\mathcal{LDR}^2\circ\mathcal{NDR}$ (31) (55) (10) | $\mathcal{DR}^2$ (27) (32) | $\mathcal{LNDR}\circ\mathcal{H}\circ\mathcal{C}_0$ | $\mathcal{LNDR}\circ\mathcal{H}$ (11) | $\mathcal{LNDR}\circ\mathcal{H}$ (12) | $\mathcal{LNDR}\circ\mathcal{H}$ (13) |

Case (a). There exists a $\mathcal{C}''\in S_1$ such that $\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n=\mathcal{C}''$, thus $\mathcal{C}=\mathcal{C}''\circ\mathcal{K}_{n+1}$. Here $\mathcal{C}$ can be given in one of the following three forms, verifying our theorem:

(i) $\mathcal{C}=\mathcal{C}''\circ\mathcal{C}_0$ if $\mathcal{C}''\in S_2$ and $\mathcal{K}_{n+1}=\mathcal{LNDR}$;

(ii) $\mathcal{C}=\mathcal{C}_1$ if $\mathcal{C}''=\mathcal{LNDR}$ and $\mathcal{K}_{n+1}=\mathcal{NH}$;

(iii) $\mathcal{C}\in S_1$ in any other cases, by Table 1.

Case (b). There exist $\mathcal{C}''\in S_2$ and $k\geqq 0$ for which

$$\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n=\mathcal{C}''\circ\mathcal{C}_k,\quad\text{so}\quad \mathcal{C}=\mathcal{C}''\circ\mathcal{C}_k\circ\mathcal{K}_{n+1}.$$

Now seven subcases detailed from (i) to (vii) can be raised proving again our theorem.

(i) $\mathcal{C}=\mathcal{DR}^2$ if $\mathcal{K}_{n+1}=\mathcal{DR}$, by (32);

(ii) $\mathcal{C}=\mathcal{C}''\circ\mathcal{NDR}\in S_1$ if $\mathcal{K}_{n+1}=\mathcal{NDR}$, by (10), (25) and table 1;

(iii) $\mathcal{C}=\begin{cases}\mathcal{LDR}^2 & \text{if } \mathcal{K}_{n+1}=\mathcal{LDR},\ k=0 \text{ and } \mathcal{C}''=\mathcal{LH},\\ & \text{by (33) and (9)}\\ & \text{if } \mathcal{K}_{n+1}=\mathcal{LDR}, \text{ and } k\geqq 1 \text{ or } \mathcal{C}''\neq\mathcal{LH}\\ \mathcal{DR}^2 & \text{since in this case } \mathcal{DR}^2\overset{(51)}{=}\mathcal{NH}\circ\mathcal{LNDR}\circ\mathcal{LH}\subseteq\\ & \subseteq\mathcal{C}''\circ\mathcal{C}_k\circ\mathcal{K}_{n+1}\subseteq\mathcal{DR}^2;\end{cases}$

(iv) $\mathcal{C}=\begin{cases}\mathcal{C}''\circ\mathcal{C}_k & \text{if } \mathcal{K}_{n+1}=\mathcal{LNDR} \text{ and } k=2m, \text{ by (26)}\\ \mathcal{C}''\circ\mathcal{C}_{k+1} & \text{if } \mathcal{K}_{n+1}=\mathcal{LNDR} \text{ and } k=2m+1;\end{cases}\quad (m\geqq 0)$

(v) $\mathcal{C}=\begin{cases}\mathcal{DR}^2 & \text{if } \mathcal{K}_{n+1}=\mathcal{H},\ \mathcal{C}''\neq\mathcal{LH} \text{ or } k\geqq 2 \text{ because}\\ & \mathcal{DR}^2\overset{(51)}{=}\mathcal{NH}\circ\mathcal{LNDR}\circ\mathcal{LH}\subseteq\mathcal{C}''\circ\mathcal{C}_k\circ\mathcal{K}_{n+1}\subseteq\mathcal{DR}^2\\ \mathcal{LNDR}\circ\mathcal{H} & \text{if } \mathcal{K}_{n+1}=\mathcal{H},\ \mathcal{C}''=\mathcal{LH} \text{ and } k=0,1 \text{ since}\\ & \text{in both cases } \mathcal{LNDR}\circ\mathcal{H}\subseteq\mathcal{C}''\circ\mathcal{C}_k\circ\mathcal{K}_{n+1}\overset{(14)}{=}\\ & \mathcal{LH}\circ\mathcal{LNDR}\circ\mathcal{H}\subseteq\mathcal{LDR}^2\circ\mathcal{H}\overset{(55)}{=}\mathcal{LNDR}\circ\\ & \circ\mathcal{LH}\circ\mathcal{H}\overset{(15)}{=}\mathcal{LNDR}\circ\mathcal{H};\end{cases}$

(vi) $\mathcal{C}=\begin{cases}\mathcal{C}''\circ\mathcal{C}_{k+1} & \text{if } \mathcal{K}_{n+1}=\mathcal{NH} \text{ and } k=2m\\ \mathcal{C}''\circ\mathcal{C}_k & \text{if } \mathcal{K}_{n+1}=\mathcal{NH} \text{ and } k=2m+1, \text{ by (16);}\end{cases}\quad (m\geqq 0)$

(vii) $\mathcal{C}=\begin{cases}\mathcal{DR}^2 & \text{if } \mathcal{K}_{n+1}=\mathcal{LH},\ \mathcal{C}''\neq\mathcal{LH} \text{ or } k\geqq 2,\\ & \text{similarly as in (v)}\\ \mathcal{LDR}^2 & \text{if } \mathcal{K}_{n+1}=\mathcal{LH},\ \mathcal{C}''=\mathcal{LH} \text{ and } k=0,\\ & \text{by (50) and (9)}\\ \mathcal{LNDR}\circ\mathcal{H} & \text{if } \mathcal{K}_{n+1}=\mathcal{LH},\ \mathcal{C}''=\mathcal{LH} \text{ and } k=1,\\ & \text{see as in (v).}\end{cases}$

Case (c). $\mathcal{K}_1\circ\ldots\circ\mathcal{K}_n=\mathcal{C}_k$ for some $k\geqq 0$, so $\mathcal{C}=\mathcal{C}_k\circ\mathcal{K}_{n+1}$. This case can be handled similarly to the case (b), the detailed proof is omitted. □

RESEARCH GROUP ON THEORY OF AUTOMATA
HUNGARIAN ACADEMY OF SCIENCES
SOMOGYI U. 7.
SZEGED, HUNGARY
H—6720

## References

[1] BAKER, B. S., Composition of top-down and bottom-up tree transductions, Inf. and Control, v. 41, 1979, pp. 186—213.
[2] ENGELFRIET, J., Bottom-up and top-down tree transformations — A comparison, Math. Syst. Theory, v. 9, 1975, pp. 198—231.
[3] GÉCSEG, F. and M. STEINBY, Tree automata, Akadémiai Kiadó, Budapest, 1984.
[4] ROUNDS, W. C., Mappings and grammars on trees, Math. Syst. Theory, v. 4, 1970, pp. 257—287.

*(Received Jan. 27, 1986)*

# The invertibility of tree transducers

IMRE NEUMÜLLER

Every finite automaton can be considered to be a finite algebra equipped with unary operations. In this setting, automata process unary polynomial symbols. This observation led to introducing tree automata by dropping the unary requirement. Basically, tree automata are finite universal algebras, and tree automata process polynomial symbols, i.e. trees. Similar generalization when applied to sequential machines leads to the concept of tree transducers. In the first section of the present paper we recall some basic definitions on tree transducers and prove a few simple propositions. The invertibility of frontier-to-root tree transducers is discussed in the second section. Namely, we give a necessary and sufficient condition describing frontier-to-root tree transducers posessing an inverse. In addition, an algorithm is given for constructing inverse transducers. Similar results are formulated in the last section for root-to-frontier tree transducers.

## 1. Notions and notations

In this section we recall concepts and results in connection with trees and forests.

**Definition 1.1.** An operation domain $F$ is a disjoint union of sets $F_n$ indexed by nonnegative integers. $F_n$ is the set of $n$-ary operational symbols. A finite operation domain is called ranked alphabet.

**Definition 1.2.** Let $X_n$ be a set of $n$ variables. An $F$-polynomial symbol over $X_n$ is called an $F$-tree over $X_n$. A set $T\subseteq T_F(X_n)$ is called an $F$-forest over $X_n$.

Let $X$ be a finite set of variables and $p\in T_F(X)$.

**Definition 1.3.** The set of all subtrees of $p$, denoted sub $(p)$, is defined as follows:

(1) if $p\in F_0\cup X$, then $\mathrm{sub}(p)=\{p\}$,

(2) if $p=f(p_1, p_2, ..., p_m)$ $(f\in F_m, m>0)$, then

$$\mathrm{sup}(p)=\{p\}\cup(\mathrm{sub}(p_i); \quad i=1, ..., m).$$

**Definition 1.4.** The root of $p$, denoted root $(p)$, is given by the following two conditions:

(1) if $p \in F_0 \cup X$, then $\operatorname{root}(p) = p$;

(2) if $p = f(p_1, \ldots, p_m)$ $(f \in F_m, m > 0)$, then $\operatorname{root}(p) = f$.

**Definition 1.5.** The height $h(p)$ of $p$ is defined by

(1) if $p \in F_0 \cup X$, then $h(p) = 0$,

(2) if $p = f(p_1, \ldots, p_m)$ $(f \in F_0, m > 0)$, then

$$h(p) = \max \{h(p_i);\ i = 1, \ldots, m\} + 1.$$

**Definition 1.6.** Let $F$ be a ranked alphabet, and $X$ a finite set of variables. The system $\mathbf{A} = (\mathfrak{A}, X, \mathscr{L}, A')$ is called an $n$-ary $F$-automaton, where

(1) $\mathfrak{A} = (A, F)$ is a finite $F$-algebra,
(2) $\mathscr{L}: X \to A$ is the initial assignment,
(3) $A' \subseteq A$ is the set of final states.

Let $\hat{\mathscr{L}}$: $T_F(X) \to \mathfrak{A}$ denote the homomorphic extension of $\mathscr{L}$, where $T_F(X)$ is now considered to be the absolutely free $F$-algebra generated by $X$. The forest recognized by $\mathbf{A}$ is defined by:

$$T(\mathbf{A}) = \{p \in T_F(X);\ p\hat{\mathscr{L}} \in A'\}.$$

A forest $T$ is called recognizable (regular) if there is an $F$-automaton $\mathbf{A}$ with $T(\mathbf{A}) = T$.

In defining tree transducers we shall make use of a set $Z = \{z_1, z_2, \ldots\}$ of auxiliary variables. We set $Z_n = \{z_1, \ldots, z_n\}$ $(n > 0)$. $Z$ is supposed to be disjoint with every other set.

**Definition 1.7.** A system $\mathbf{A} = (T_F(X_n), A, T_G(Y_m), A', P)$ is called a frontier-to-root tree transducer ($F$-transducer), where

(1) $F$ and $G$ are ranked alphabets,

(2) $A$ is a ranked set containing only unary operational symbols, the state set of $\mathbf{A}$. (It is assumed that $A$ is disjoint with all other sets in the definition of $\mathbf{A}$, expect $A'$.)

(3) $A' \subseteq A$ is the set of final states,

(4) $P$ is a finite set of rewriting rules of the following two types:

(i) $x_i \to a(q)$ $(x_i \in X_n,\ a \in A,\ q \in T_G(Y_m))$ and

(ii) $f(a_1(z_1), \ldots, a_k(z_k)) \to a(q(z_1, \ldots, z_k))$ $(f \in F_k;\ k \geqq 0;\ a_1, \ldots, a_k,\ a \in A;$ $z_1, \ldots, z_k \in Z_k;\ q(z_1, \ldots, z_k) \in T_G(Y_m \cup Z_k))$.

In what follows, if $a \in A$ and $t$ is a tree, instead of $a(t)$ we shall use the notation $at$. Accordingly, we write $AT$ for the set $AT = \{at | a \in A;\ t \in T\}$ if $T$ is any forest.

Let $\mathbf{A}$ be the above defined $F$-transducer. It is said that $\mathbf{A}$ is

— linear, if every $z_i \in Z$ occurs at most once on the right side of a rewriting rule,

— nondeleting, if in every rewriting rule, all the variables $z_i$ occurring on the left side occur on the right side, too,

— completely defined, if for every $i(\leqq n)$, $l(\geqq 0)$, $f(\in F_l)$, and $a_1, \ldots, a_l(\in A)$ there are a rewriting rule with left side $x_i$ and a rewriting rule with left the side $f(a_1 z_1, \ldots, a_l z_l)$.

We are now going to define the tree transformation induced by an $F$-transducer **A**. Let $p, q \in T_F(X_n \cup AT_G(Y_m \cup Z))$ be arbitrary trees, and **A** the tree transducer given in Definition 1.7. We say that $p$ directly derives $q$ in **A**, if $q$ can be obtained from $p$

(i) by substituting $a\bar{q}$ for an occurrence of $x_i$ in $p$ provided that $x_i \to a\bar{q}$ is a rewriting rule in $P$,

(ii) or by substituting $a\bar{q}(p_1, \ldots, p_k)$ for an occurrence of a subtree of the form $f(a_1 p_1, \ldots, a_k p_k)$, provided that $f(a_1 z_1, \ldots, a_k z_k) \to a\bar{q}(z_1, \ldots, z_k)$ is a rewriting rule in $P$. We use the notation $\underset{\mathbf{A}}{\Rightarrow}$ for direct derivation. The reflexive-transitive closure of $\underset{\mathbf{A}}{\Rightarrow}$ is denoted $\overset{*}{\underset{\mathbf{A}}{\Rightarrow}}$. If $p \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} q$ we say that $p$ derives $q$ in **A**.

**Definition 1.8.** The $F$-transformation induced by the $F$-transducer **A** is the following relation $\tau_{\mathbf{A}}$:

$$\tau_{\mathbf{A}} = \{(p, q) | p \in T_F(X_n), \quad q \in T_G(Y_m), \quad p \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} aq, \quad a \in A'\}.$$

**Definition 1.9.** The system $\mathbf{A} = (T_F(X_n), A, T_G(Y_m), A', P)$ is called a root-to-frontier tree transducer, $R$-transducer for short, if

(1) $F$, $G$ and $A$ are as in Definition 1.7,

(2) $A' \subseteq A$ is the set of initial states,

(3) $P$ is a finite set of rewriting rules of one of the following two forms:

(i) $ax_i \to q \quad (x_i \in X_n, \quad a \in A, \quad q \in T_G(Y_m))$ and

(ii) $af(z_1, \ldots, z_k) \to q \quad (f \in F_k; \quad k \geqq 0; \quad a \in A; \quad z_1, \ldots, z_k \in Z_k; \quad q \in T_G(Y_m \cup AZ_k))$.

Linear, nondeliting and completely defined $R$-transducers are defined in a way analogous to the $F$-transducer case.

To define the transformation induced by the above $R$-transducer we define the direct derivation $\underset{\mathbf{A}}{\Rightarrow}$ in **A** for trees $p, q \in T_G(AT_F(X_n \cup Z) \cup Y_m)$ as follows: $p \underset{\mathbf{A}}{\Rightarrow} q$ if and only if either

(i) $q$ is obtained from $p$ by substituting $\bar{q}$ for an occurrence of a subtree $ax_i$ in $p$ provided that $ax_i \to \bar{q}$ is a rewriting rule in $P$, or

(ii) $q$ is obtained from $p$ by substituting $\bar{q}(p_1, \ldots, p_k)$ for an occurrence of a subtree $af(p_1, \ldots, p_k) \in \mathrm{sub}(p)$ provided that $af(z_1, \ldots, z_k) \to \bar{q}$ is a rewriting rule in $P$.

The reflexive transitive closure of $\underset{\mathbf{A}}{\Rightarrow}$ is again denoted $\overset{*}{\underset{\mathbf{A}}{\Rightarrow}}$. If $p \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} q$, we say that $p$ derives $q$ in **A**.

**Definition 1.10.** The root-to-frontier tree transformation ($R$-transformation) induced by **A** is the binary relation:

$$\tau_{\mathbf{A}} = \{(p, q) | p \in T_F(X_n); \; q \in T_G(Y_m); \; ap \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} q; \; a \in A'\}.$$

**Definition 1.11.** Let A and B be $R$-transducers ($F$-transducers). It is said that A is equivalent to B if $\tau_A = \tau_B$.

**Definition 1.12.** An $F$-transducer A ($R$-transducer) is called bounded, if $\tau_A^{-1}(q)$ is a finite set for every $(p, q) \in \tau_A$.

Let A be an arbitrary $F$-transducer. A state $a \in A$ is accessible if there is a tree $p \in T_F(X_n)$ with $p \overset{*}{\underset{A}{\Rightarrow}} aq$ for some $q \in T_G(Y_m)$. In this case we also say $p$ leads to $a$. Similarly, we say that a state $a' \in A$ is accessible from a state $a \in A$ if there are $p \in T_F(X_n \cup Z_1)$ and $q \in T_G(Y_m \cup Z_1)$ with $p(az_1) \overset{*}{\underset{A}{\Rightarrow}} a'q$.

**Definition 1.13.** An $F$-transducer is called biaccessible if every state $a$ is accessible, and for every state $a$ there is a final state $a'$ such that $a'$ is accessible from $a$. (In other words, this means that every state occurs in a derivation $p \overset{*}{\underset{A}{\Rightarrow}} aq$ where $p \in T_F(X_n)$, $q \in T_G(Y_m)$ and $a \in A'$.)

It is easily seen that for every $F$-transducer A there is an equivalent biaccessible $F$-transducer provided that $\tau_A \neq \emptyset$.

Let A be an arbitrary $R$-transducer. A state $a \in A$ is called essential if there are $p \in T_F(X_n)$ and $q \in T_G(Y_m)$ with $ap \overset{*}{\underset{A}{\Rightarrow}} q$. A rewriting rule

$$af(z_1, \dots, z_k) \to r \tag{1}$$

is called essential if there are $p_1, \dots, p_k \in T_F(X_n)$ and $q \in T_G(Y_m)$ with $af(p_1, \dots, p_k) \overset{*}{\underset{A}{\Rightarrow}} q$ such that in the course of the derivation the first rule applied is (1).

**Definition 1.14.** An $R$-transducer A is called biaccessible if its states and rewriting rules are essential, further, every state occurs in a derivation $ap \overset{*}{\underset{A}{\Rightarrow}} q$ where $p \in T_F(X_n)$, $q \in T_G(Y_m)$ and $a \in A'$.

Again, it is straightforward to prove that for every $R$-transducer A there is an equivalent biaccessible $R$-transducer provided that $\tau_A \neq \emptyset$.

**Definition 1.15.** Let A be an $F$-transducer. A state $a \in A$ is said to be of $k$-type for $k = 0, \dots, \infty$, if there are exactly $k$ distinct trees leading to $a$.

**Lemma 1.16.** Let A be a biaccessible $F$-transducer. There exists a biaccessible $F$-transducer B which is equivalent to A and such that every state of B is either of 1-type or of $\infty$-type.

*Proof.* Since A is biaccessible A does not have 0-type states. If A has only 1-type or $\infty$-type states set B=A. Assume that $a \in A$ is of $k$-type with $1 < k < \infty$. There are $p_1, \dots, p_k$ ($p_i \in T_F(X_n)$; $i = 1, \dots, k$) and $q_{j1}, \dots, q_{jt_j}$ ($q_{jl} \in T_G(Y_m)$ $j = 1, \dots, k$; $l = 1, \dots, t_j$) with $p_j \overset{*}{\underset{A}{\Rightarrow}} aq_{jl}$. (The trees $p_j$ and $q_{jt}$ can be determined in an effective way.) Let $A_i = \{p_i, p_i^1, \dots, p_i^{s_i}\}$ ($i = 1, \dots, k$) denote the set of subtrees of $p_i$. In what follows, $\bar{p}_i$ and $\bar{p}_i^j$ will denote states. Take the following sets of rewriting rules $P_i$ $(i=1, \dots, k)$: (Let $r$ be an arbitrary tree in $T_G(Y_m)$.)

$$\text{if} \quad A_i = \{p_i\} \quad \text{then} \quad v \to \bar{p}_i q_{im} \in P_i \Leftrightarrow v = p_i; \quad v \in X_n \cup F_0$$
$$m = 1, \dots, t_i,$$
$$\text{if} \quad A_i \neq \{p_i\} \quad \text{then} \quad v \to \bar{p}_i^j r \in P_i \Leftrightarrow v = p_i^j; \quad v \in X_n \cup F_0;$$
$$f_h(\bar{p}_i^{j_1} z_1, \dots, \bar{p}_i^{j_h} z_h) \to \bar{p}_i^u r \in P_i \quad \text{if and only if} \quad f_h(p_i^{j_1}, \dots, p_i^{j_h}) = p_i^u$$
$$f_h(\bar{p}_i^{j_1} z_1, \dots, \bar{p}_i^{j_h} z_h) \to \bar{p}_i q_{im} \in P_i \quad \text{if and only if} \quad f_h(p_i^{j_1} \dots, p_i^{j_h}) = p_i$$
$$h > 0; \ m = 1, \dots, t_i; \ u, j_1, \dots, j_h \in \{1, \dots, s_i\}.$$

Let $P^*$ consist of all those rules of $P$ not containing an occurrence of the state $a$ as well as the rules formed in the following way: If a rule in $P$ contains an occurrence of $a$ then substitute $p_i (i=1, \dots, k)$ for $a$ in every possible way. Take the $F$-transducer $\mathbf{C}=(T_F(X_n), C, T_G(Y_m), C', P')$, where

$$C = A \cup A_1 \cup \dots \cup A_k - \{a\}$$
$$C' = \begin{cases} A', & \text{if} \quad a \notin A' \\ A' \cup \{\bar{p}_1, \dots, \bar{p}_k\} - \{a\}, & \text{if} \quad a \in A' \end{cases}$$
$$P' = P_1 \cup \dots \cup P_k \cup P^*.$$

It is easy to see that **A** is equivalent to **C** and **C** has fewer states of type $k$, $1<k<\infty$, than **A**. In a finite number of steps we arrive at the transducer **B** with the required property.

We continue by introducing a few concepts to be used later. Let **A** be an arbitrary $F$-transducer. A rewriting rule is called jumping provided that it is of the form

$$f(a_1 z_1, \dots, a_k z_k) \to a z_i \quad (0 < i \leqq k).$$

A state $a \in A$ is called

— deleting state, if there is a rule containing $az_i$ on the left side but $z_i$ does not occure on the right side,

— multiplying state, if there is a rule containing $az_i$ on the left side and $z_i$ occurs at least twice on the right side,

— jumping state, if there is a jumping rule containing $az_i$ on the left side, and the right side is of the form $bz_i$ for some $b$, i.e., the right side contains the variable corresponding to $a$ on the left side of the rule.

A chain $a_1, a_2, \dots, a_k$ $(k>0)$ of states is called a jumping cycle if for every $i=1, \dots, k$ there is a jumping rule containing $a_i$ on the left side and such that its right side is $a_{i+1} z_j$ where $z_j$ is the variable corresponding to $a_i$ on the left side of the rule. If $i=k$, $a_{i+1}=a_1$. For the sake of simplifying the treatment, if the left side of a rule contains a state $a$ of $k$-type, then the auxiliary variable corresponding to $a$ is called of $k$-type, too.

Let **A** be an arbitrary $R$-transducer. A rewriting rule is called a jumping rule provided that it is of the form $af(z_1, \dots, z_k) \to a' z_i$. A state a is said to be

— jumping state, if there is a jumping rule whose left side contains $a$. A chain $a_1, \dots, a_k$ is a jumping cycle if for every $i$ there is a jumping rule with left side containing $a_i$ and right side containing $a_{i+1}$. Again, $a_{k+1}=a_1$.

Later we shall use the following notation: $A(a_i)$ is the $R$-transducer obtained from **A** by letting $a_i$ to be the unique initial state.

The proof of the next result can be found in [3].

**Theorem 1.17.** For every linear nondeleting $F$-transducer there is an equivalent linear nondeleting $R$-transducer and conversely.

**Definition 1.18.** Let **A** be an arbitrary $F$-transducer ($R$-transducer). An $F$-transducer ($R$-transducer) **B** is called an inverse of **A**, if $\tau_A^{-1} = \tau_B$.

## 2. The invertibility of $F$-transducers

In what follows we shall always assume that the $F$-transducers to be considered are biaccessible with states 1-type or $\infty$-type. By the previous section this assumption does not restrict the generality of the treatment except for the induced transformation is the empty relation — however, in this case the inverse is obviously inducable.

First let us discuss some necessary conditions of invertibility.

**Theorem 2.1.** Let **A** be an arbitrary $F$-transducer. Then the domain of $\tau_A$ is regular and $\tau_A^{-1}$ preserves regularity.

The proof of the above result can be found in [3].

**Lemma 2.2.** Let **A** be a biaccessible $F$-transducer. If **A** is invertible then $\tau_A$ preserves regularity.

*Proof.* The statement is obvious by Theorem 2.1.

**Lemma 2.3.** Let **A** be a biaccessible $F$-transducer. If **A** is invertible then **A** is bounded.

*Proof.* If **A** is not bounded then there are an infinite number of trees mapped to the same tree $q$ under $\tau_A$. Thus, $q$ has an infinite number of images under $\tau_A^{-1}$. This contradicts the invertibility of **A**.

**Lemma 2.4.** Let **A** be an arbitrary biaccessible $F$-transducer. **A** is bounded if and only if

(1) **A** has no jumping cycle of states and
(2) **A** has no deleting state of $\infty$-type.

*Proof.* Let $\mathbf{A} = (T_F(X_n), A, T_G(Y_m), A', P)$. Assume that $a \in A$ is a deleting state of $\infty$-type. Let $r_1, r_2, \ldots$ be distinct trees in $T_F(X_n)$ with $r_i \overset{*}{\underset{A}{\Rightarrow}} aq_i$ $(i = 1, 2, \ldots)$, $q_i \in T_G(Y_m)$. As **A** is biaccessible, there are $p \in T_F(X_n)$, $q \in T_G(Y_m)$ such that $(p, q) \in \tau_A$, and in the derivation of $q$ from $p$ we go trough the state $a$ at a stage where the subtree $r$ belonging to $a$ is deleted. Let us replace the subtree $r$ in $p$ by $r_1, r_2, \ldots$; respectively. For the trees $p_1, p_2, \ldots$ obtained in this way we have $(p_i, q) \in \tau_A$.

Suppose now that $a_1, \ldots, a_k$ $(k > 0)$ is a jumping cycle of states, and let $s_1, \ldots, s_k$ be the corresponding jumping rules. Put $r_1 = z$, where $z \in Z$ is an auxiliary variable. Having defined $r_l$ $(l = 1, \ldots, k)$, form $r_{l+1}$ in the following way. Let $s_l$ be the rule

$f(a^1z_1, \ldots, a_lz, \ldots, a^uz_u) \to a_{l+1}z$. (Of course, $a_{k+1} = a_1$) Take a tree $t^i \in T_F(X_n)$ with $t^i \overset{*}{\underset{A}{\Rightarrow}} a^i q^i$ where $q^i \in T_G(Y_m)$ for every $i = 1, \ldots, u$. Put

$$r_{l+1} = f(t^1, \ldots, r_l, \ldots, t^u) \quad l = 1, \ldots, k.$$

Denote by $p^*$ the tree $r_{k+1}$. Obviously, $h(p^*) > 0$. Since A is biaccessible there are $p \in T_F(X_n)$, $q \in T_G(Y_m)$ with $(p, q) \in \tau_A$, and such that a derivation of $q$ from $p$ goes through $a_1$. Denote by $\bar{p}$ that subtree of $p$ leading to $a_1$. Put

$$\bar{p}_0 = \bar{p},$$

$$\bar{p}_{i+1} = p^* \cdot_z \bar{p}_i,$$

where $\cdot_z$ denotes the $z$-product of trees. Substitute $\bar{p}_i$ for $\bar{p}$ in $p$, and let $p_i$ $(i = 0, 1, \ldots)$ be the resulting tree. Obviously $(p_i, q) \in \tau_A$, ending the proof of the necessity.

Conversely, if both (1) and (2) are satisfied by A, then for every $(p, q) \in \tau_A$ it holds that $h(q) > (h(p) - m)/(k+1)$, where $m = \max\{h(r);\ r \overset{*}{\underset{A}{\Rightarrow}} aq,\ q \in T_G(Y_m),\ a \in A$ is a deleting state$\}$, and $k$ is the cardinality of the state set. From this, A is easily seen to be bounded.

Next we try to construct an inverse transducer by "inverting the rules", if it is possible.

**Definition 2.5.**

— The inverse of a rule $x_i \to aq$ or $f_0 \to aq\,(f_0 \in F_0)$ is a finite set of rules ensuring $q \overset{*}{\Rightarrow} ax_i$ or $q \overset{*}{\Rightarrow} af_0$.

— The inverse of a rule $f(a_1z_1, \ldots, a_kz_k) \to aq$ $k > 0$, $(q \in T_G(Y_m \cup Z_k)$ $q \neq z_i$ $(i = 1, \ldots, k)$ is a finite set of linear rules in which none of the auxiliary variables occure (the auxiliary variables are denoted e.g. by $v_1, v_2, \ldots)$, and such that these rules realize a derivation $q(a_1z_1, \ldots, a_kz_k) \overset{*}{\Rightarrow} af(r_1, \ldots, r_k)$ where the variables $z_i$ occure with the same multiplicity and with the same states as in the original rules and

$$r_i = \begin{cases} z_i & \text{if } a_i \text{ is of } \infty\text{-type} \\ p_i & \text{if } a_i \text{ is of 1-type and } p_i \text{ leads to } a_i. \end{cases}$$

— Further, we say that some occurrences of the variables $z_1, \ldots, z_i$ meet at the same vertex in a tree $q \in T_G(Y_m \cup Z)$ if $q$ has a vertex such that there is a path in $q$ from that vertex to every given occurrence of the variables $z_j\,(j = 1, \ldots, i)$ and there is no edge in $q$ belonging to two different such paths.

**Lemma 2.6.** Suppose that the states occuring in the rules are of 1-type or of $\infty$-type and none of them is an $\infty$-type deleting state. Then

(1) all the rules $x_i \to aq$, $f_0 \to aq$ $(f_0 \in F_0;\ q \in T_G(Y_m))$ are invertible.

(2) a linear rule $f(a_1z_1, \ldots, a_kz_k) \to aq(k > 0;\ q \in T_G(Y_m \cup Z_k);\ q \neq z_i;\ i = 1, \ldots, k)$ is invertible if and only if all $\infty$-type auxiliary variables in $q$ meet at same vertex,

(3) a nonlinear rule $f(a_1z_1, \ldots, a_kz_k) \to aq$ $(k > 0; q \in T_G(Y_m \cup Z_k))$ is invertible if and only if each $\infty$-type auxiliary variable in $q$ has an occurrence such that these occurrences meet at the same vertex.

*Proof.* Suppose that a rule is invertible. There is a uniquely determined vertex in $q$ such that we get back the given occurrence of the symbol $f$ during the derivation process. Of the arguments of $f$, all $\infty$-type auxiliary variables occur. Since none of the auxiliary variables $z_i$ occurs in the rules realizing the inverse of the rule, the vertex in question has an outgoing path to an occurrence of every $\infty$-type auxiliary variable and no edge belongs to more than one such path. We have proved the necessity in case of conditions (2) and (3).

(1) In the same way as in the proof of Lemma 1.16 we can construct an $F$-transducer transforming a given tree to a given tree. Let us replace in this transducer every occurrence of the final state by $a$, the resulting set of rules is the inverse of the rule given in (1).

(2) Further on the symbols $z_i (i=1, \ldots, k)$ will not be considered to be auxiliary variables. The auxiliary variables will be denoted by $v$. Denote by $S=\{s_1, \ldots, s_h\}$ the set of all those vertices of $q(z_1, \ldots, z_k)$ from which there are paths leading to $\infty$-type $z_i$'s. Let $R=\{r_1, \ldots r_l\}$ be the set of those subtrees of $q(z_1, \ldots, z_k)$ whose roots are directly attached to vertices in $S$ and which does not contain vertices of $S$. (Different occurrences are treated separately.) Let $t \in T_F(X_n)$ be an arbitrary tree. For every $j (j=1, \ldots, l)$ there is an $F$-transducer $\mathbf{B}_j$ with $\tau_{\mathbf{B}_j}=\{(r_j, t)\}$. Let $P_j$ be the set of rules of $\mathbf{B}_j$. If $z_i$ occurs in a rule then replace all occurrences of the state appearing on the right side with $a_i$. Delete those rules containing an occurrence of one of the symbols $z_i$. Denote by $P'$ the union of the sets of rules obtained in this way, and let $c_1, \ldots, c_l$ be the final states. If $S$ is empty then $R=\{q\}$. In this case put $t=f(p_1, \ldots, p_h)$ where $p_i$ leads to $a_i$, and replace $c_i$ with $a$ everywhere — the proof is done. Otherwise, let $e$ denote the vertex where the $\infty$-type $z_i$'s meet, if there is only one such $z_i$ let $e$ be the root. Let us assign to each $s_i$ a state $d_i$ $(i=1, \ldots, n)$ as follows: If $s_i$ is the root then $d_i=a$, otherwise let $d_i$ be a new state. Let $g(q_1, \ldots q_\alpha)$ be the subtree corresponding to the vertex $s_i$. We construct a rule to every vertex $s_i$ in $S$. If the vertex is different from $e$ then the rule is:

$$g(b_1 v_1, \ldots, b_\alpha v_\alpha) \to d_i v_u \quad \text{where}$$

$$b_j = \begin{cases} c_\beta, & \text{if} \quad q_j = r_\beta \\ d_k, & \text{if} \quad q_j \text{ has root } s_k \end{cases}$$

$(j=1, \ldots, \alpha)$ and $v_u$ is the auxiliary variable corresponding to the unique $d$ occurring on the left side. If the vertex coincides with $e$ then

$$g(b_1 v_1, \ldots, b_\alpha v_\alpha) \to d_i f(\bar{r}_1, \ldots, \bar{r}_k) \quad \text{where}$$

$$b_j = \begin{cases} c_\beta, & \text{if} \quad q_j = r_\beta \\ d_u, & \text{if} \quad q_u \text{ has root } s_u \end{cases}$$

$$(j = 1, \ldots, \alpha)$$

$$\bar{r}_j = \begin{cases} p_j, & \text{if} \quad a_j \text{ is of 1-type and } p_j \text{ leads to } a_j \\ v_u, & \text{if} \quad z_j \text{ is of } \infty\text{-type and occurs in } q_u \end{cases}$$

$$(j = 1, \ldots, k).$$

Take the union of all the sets of rules costructed. Obviously, this set is an inverse of the original rule.

(3) Choose an occurrence of every $\infty$-type $z_i$ in such a way that these occurrences meet at the same vertex, and consider these occurrences as $\infty$-type $z_i$'s. The proof is finished as in (2).

The proof of the next two theorems can be found in [4].

**Theorem 2.7.** Let A be an arbitrary $F$-transducer. $\tau_A$ preserves regularity if an only if A is equivalent to a linear $F$-transducer.

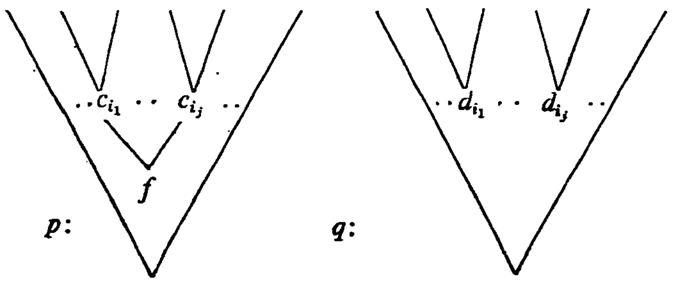
**Theorem 2.8.** Let A be a biaccessible $F$-transducer. Then A is equivalent to a linear $F$-transducer if and only if A is linear or its multiplying states are of finite type (i.e. of $k$-type with $k<\infty$).

**Lemma 2.9.** Let A be a biaccessible $F$-transducer. If A is invertible then its non-jumping rules are invertible.

*Proof.* Assume to the contrary that A is invertible and has a non-jumping rule which is not invertible. On the basis of the previous results we may assume that A is biacessible, linear, and its states are of 1-type or of $\infty$-type. Denote by **B** an inverse of A. Let

$$f(a_1 z_1, \ldots, a_k z_k) \to a\bar{q}(z_1, \ldots, z_k) \qquad (1)$$

be a non-invertible rule. By Lemma 2.6, the $\infty$-type auxiliary variables do not meet at a single vertex. Because of biaccessibility, there are $p \in T_F(X_n)$ and $q \in T_G(Y_m)$ such that $(p, q) \in \tau_A$ and the rule is used in the course of deriving $q$ from $p$. Let $e$ denote that vertex of $p$ where the rule (1) is used. Let $z_{i_1}, \ldots, z_{i_j}$ be all the $\infty$-type auxiliary variables in $q$ with corresponding $\infty$-type states $a_{i_1}, \ldots, a_{i_j}$. Obviously, $j>2$. Let $c_{i_1}, \ldots c_{i_j}$ denote those verteces in $p$ which are direct descendants of $e$ (i.e. there are edges from $e$ to them) and such that in the course of the derivation we obtain $\infty$-type states after processing the subtrees belonging to them. The auxiliary variables $z_{i_1}, \ldots,$ $\ldots, z_{i_j}$ correspond to these vertices $c_{i_1}, \ldots, c_{i_j}$. Let $d_{i_t} (t=1, \ldots, j)$ denote the root vertex of the image of the subtree belonging to $c_{i_t}$.


$\ldots c_{i_1} \cdot\cdot\ c_{i_j} \ldots$
$f$
$p$:
$q$:
$\ldots d_{i_1} \cdot\cdot\ d_{i_j} \ldots$

Since $a_{i_1}, \ldots, a_{i_j}$ are of $\infty$-type, for every $t$ there are distinct trees $p_t^u$ $(u=1, 2, \ldots; t=1, \ldots, j)$ leading to $a_{i_t}$. Replacing the subtrees belonging to the vertices $c_{i_1}, \ldots, c_{i_j}$ by $p_1^{u_1}, \ldots, p_j^{u_j}$, respectively, the trees $p_n$ $(n=1, 2, \ldots)$ can be transformed to trees $q_n$ "similar" to $q$: $q_n$ is obtained from $q$ by replacing the subtrees at $d_{i_1}, \ldots, d_{i_j}$ with images of the trees replacing the subtrees at $c_{i_1}, \ldots, c_{i_j}$ in $p$. By Lemma 2.3, the set of these trees $q_n$ is infinite, and $(q_n, p_n) \in \tau_B$ by assumption. Classify the vertices of the trees $q_n$ as follows. The vertices of the subtrees replacing the subtrees at $d_{i_1}, \ldots, d_{i_j}$ in $q$ belong to $O_1, \ldots, O_j$, resp. All other vertices form a singleton class. It is plain to see that a "similar" classification is obtained for every $n$. Since $(q_n, p_n) \in \tau_B$ for all $n$, therefore every tree $q_n$ has a unique vertex whose translation in **B** gives back that occurrence of the symbol $f$ which is the label of $e$ in $p$. The above classification is finite, therefore, there is a class containing the vertex in question for infinitely many $q_n$. With this we have designated an infinite subset of the trees $p_n$, as well. It is easily seen that there is a class such that the corresponding trees $p_n$ satisfy the following: for each $c_{i_t}$ there is an infinite number of subtrees at $c_{i_t}$. In what follows we restrict ourselves to this class. Suppose it is one of the singleton classes, i.e. a concrete vertex. If this vertex is not on the paths from the root to one of the $d_{i_t}$'s then the same tree belongs to this vertex in every $q_n$. Consequenly, **B** should translate an infinite number of trees from this very same tree, which is impossible. If the vertex is located on a path to one of the vertices $d_{i_1}, \ldots, d_{i_j}$ then, by assumption, at most $j-1$ edges lead from this vertex in the direction of $d_{i_1}, \ldots, d_{i_j}$. Since the designated trees $p_n$ are such that infinitely many independent trees are attached to each of the vertices $c_{i_1}, \ldots, c_{i_j}$, this case leads to a contradiction, too. If the class in question is one of the classes $O_1, \ldots, O_j$ then **B** is not bounded. This derives from the fact that, if e.g. $O_1$ is the particular class, then an infinite number of trees may be attached to each of the vertices $d_{i_2}, \ldots, d_{i_j}$ but the subtree belonging to this vertex is already obtained during the translation of the subtree belonging to $d_{i_1}$. It is also impossible because **B** has inverse, namely **A**.

**Lemma 2.10.** Let **A** be a biaccessible, bounded $F$-transducer such that $\tau_A$ preserves regularity. If the non-jumping rules of **A** are invertible then **A** is invertible.

*Proof.* We may assume that every state of **A** is of 1-type or of $\infty$-type and that **A** is linear. By boundedness, **A** does not contain deleting states of $\infty$-type and jumping cycles. If $P$ has a jumping rule then form all the state chains $a_1, \ldots, a_k$ $(k > 1)$ satisfying the following condition: $a_1, \ldots, a_{k-1}$ are jumping states, a unique jumping rule leads from $a_i$ to $a_{i+1}$ $(i=1, \ldots, k-1)$. In a similar way as we assigned a tree to jumping cycle in the proof of Lemma 2.4 let us assign a tree to every chain in question, as well. We do this in all possible ways. Of course, it may happen that more then one tree is assigned to the same chain (if there are more jumping rules from $a_i$ to $a_{i+1}$).

Let $a_1, \ldots, a_k$ be a state chain with corresponding tree $r$. Choose a non-jumping rule leading to $a_1$, say it is

$$f(a^1 z_1, \ldots, a^l z_l) \to a_1 q. \tag{1}$$

Of course, it can be of the form $x_i \to a_1 q$, too. Form the following formal transition rule:

$$r \cdot_z f(a^1 z_1, \ldots, a^l z_l) \to a_k q, \tag{2}$$

where $q$ is the tree appearing an the right side of (1). Take all possible formal transition rules (2). The inverse of a formal transition rule is meant a finite set of linear rules not containing the auxiliary variables $z_1, \ldots, z_l$ (the auxiliary variables are denoted by $v$) and such that they realize the derivation $q(a^1 z_1, \ldots, a^l z_l) \overset{*}{\Rightarrow} r \cdot_z f(s_1, \ldots, s_l)$, where the $z_i$'s $(i=1, \dots, l)$ occure with the same multiplicity and with the same states as in (2), and

$$s_i = \begin{cases} z_i & \text{if } a^i \text{ is of } \infty\text{-type}, \\ p_i & \text{if } a^i \text{ is of 1-type and } p_i \text{ leads to } a_i. \end{cases}$$

Obviously, we get a finite set of rules and if (1) is invertible then so is (2).

Let $\mathbf{B}=(T_G(Y_m), B, T_F(X_n), B', P')$ be the $F$-transducer where $B=A\cup\bar{A}$,

$\bar{A}$ is the set of the states which are obtained in the inversion process of non-jumping and formal rules,

$B'=A'$,

$P'$ is the set of the rules which are obtained in the inversion process of non-jumping and formal rules. It suffices to show that

$$p \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} aq \Leftrightarrow q \overset{*}{\underset{\mathbf{B}}{\Rightarrow}} ap \quad a\in A;\ p\in T_F(X_n);\ q\in T_G(Y_m).$$

Proof of $\Rightarrow$. If $h(p)=0$ the implication holds by the invertibility of the rules. We proceed by induction of $h(p)$. Suppose the claim for $h(p)<m$ and let $h(p)=m$. If the rule applied for the last time in the derivation $p \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} aq$ is not a jumping rule then $p=f(p_1, \dots, p_k)$ $(k>0)$ and

$$f(a_1z_1, \dots, a_kz_k) \to a\bar{q}\in P, \quad q = \bar{q}(q_1, \dots, q_k), \quad \bar{q} \neq z_i \quad (i = 1, \dots, k)$$

$$\text{and} \quad p_j \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} a_jq_j \quad (j = 1, \dots, k)$$

for some states $a_1, \dots, a_k\in A$. Thus,

$$p = f(p_1, \dots, p_k) \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} f(a_1q_1, \dots, a_kq_k) \underset{\mathbf{A}}{\Rightarrow} a\bar{q}(q_1, \dots, q_k) = aq.$$

By the induction hypothesis, $q_j \overset{*}{\underset{\mathbf{B}}{\Rightarrow}} a_jp_j$ $(j=1, \dots, k)$. Since all the non-jumping rules of $\mathbf{A}$ are invertible,

$$q = \bar{q}(q_1, \dots, q_k) \overset{*}{\underset{\mathbf{B}}{\Rightarrow}} \bar{q}(a_1p_1, \dots, a_kp_k) \overset{*}{\underset{\mathbf{B}}{\Rightarrow}} af(p_1, \dots, p_k) = ap.$$

If the rule applied for the last time is a jumping rule then we have "used" a formal rule at the end of the derivation. In this case the derivation can be written as

$$p = \bar{p}(f(p_1, \dots, p_k)) \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} \bar{p}(f(a_1q_1, \dots, a_kq_k)) \underset{\mathbf{A}}{\Rightarrow}$$

$$\bar{p}(a'\bar{q}(q_1, \dots, q_k)) \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} a\bar{q}(q_1, \dots, q_k) = aq,$$

where the rule applied in the second part is non-jumping and all the rules applied in the third part are jumping. In this case there is a formal rule $\bar{p}(f(a_1z_1, \dots, a_kz_k)) \to$ $\to a\bar{q}(z_1, \dots, z_k)$. Formal rules are invertible, therefore,

$$q = \bar{q}(q_1, \dots, q_k) \overset{*}{\underset{\mathbf{B}}{\Rightarrow}} \bar{q}(a_1p_1, \dots, a_kp_k) \overset{*}{\underset{\mathbf{B}}{\Rightarrow}} ap.$$

Proof of $\Leftarrow$. The proof is accomplished by induction on $h(q)$. Suppose $h(q)=0$. Then, by the construction of the rules in $P'$, either $h(p)=0$ and $p\to aq\in P$, or $p==f(p_1, \dots, p_k)$ $(k>0)$ and $f(a_1z_1, \dots, a_kz_k)\to aq\in P$; $p_j\overset{*}{\underset{A}{\Rightarrow}}a_jq_j$, $(j=1, \dots, k)$ where $a_j$ is 1-type. But then $p\overset{*}{\underset{A}{\Rightarrow}}aq$. Suppose that the proof is done for $h(q)<m$, and let $h(q)=m$. Take the graph representation of $q$. Let us mark those verteces in the graph at which the derivation gets into a state belonging to $A$. Denote it by $q^*$. Denote by $q'$ the maximal connected subgraph of $q^*$ containing the vertex which corresponds to the root of $q$ but not containing any other marked vertex. Let $\bar{q}$ denote that part of $q$ corresponding to $q'$. We have

$$q=\bar{q}(q_1, \dots, q_k)\overset{*}{\underset{B}{\Rightarrow}}\bar{q}(a_1p_1, \dots, a_kp_k)\overset{*}{\underset{B}{\Rightarrow}}a\bar{p}(p_1, \dots, p_k)=ap \quad a_1, \dots, a_k, a\in A.$$

After the first part we will not get back to a state in $A$ only at the root. The inversion of the rule was done in such a way that we introduced new states at each stage, and with the new rules, we could get to a state belonging to $A$ only at the root of the right side of the original rule. From this it follows that either $\bar{p}(a_1z_1, \dots, a_kz_k)\to\to a\bar{q}(z_1, \dots, z_k)\in P$ or there is a formal rule with left side $\bar{p}(a_1z_1, \dots, a_kz_k)$ and right side $a\bar{q}(z_1, \dots, z_k)$. This ends the proof.

The main results of this section easily follows from Lemmas 2.2, 2.3, 2.9, 2.10.

**Theorem 2.11.** A biaccessible $F$-transducer A is invertible if and only if A is bounded, the non-jumping rules of A are invertible and $\tau_A$ preserves regularity;

**Theorem 2.12.** The invertibility of $F$-transducers is decidable. There is an effective procedure for constructing inverse $F$-transducers.

## 3. The invertibility of R-transducers

Again, we only treat biaccessible $R$-transducers. The proof of the following theorem can be found in [3].

**Theorem 3.1.** Let A be an $R$-transducer. Then the domain of $\tau_A$ is regular and $\tau_A^{-1}$ preserves regularity.

**Lemma 3.2.** Let A be a biaccessible $R$-transducer. If A is invertible then $\tau_A$ preserves regularity.

*Proof.* The statement is trivial by Theorem 3.1.

**Lemma 3.3.** Let A be a biaccessible $R$-transducer. If A is invertible then A is bounded.

*Proof.* Similar to that of Lemma 2.3.

**Lemma 3.4.** Let A be an arbitrary biaccessible $R$-transducer. A is bounded if and only if:

(1) A has no jumping cycle of states,
(2) A is nondeleting.

*Proof.* The proof is similar to that of Lemma 2.4.

**Theorem 3.5.** Let **A** be an arbitrary biaccessible $R$-transducer. **A** is invertible if and only if:

(1) $\tau_{\mathbf{A}}$ preserves regularity,
(2) **A** is bounded,
(3) the rewriting rules are of one of the following three types:

(a) $ap \to q \quad a \in A;\ p \in X_n \cup F_0;\ q \in T_{G_0 \cup G_1}(Y_m)$,

(b) $af(z_1) \to q \quad a \in A;\ f \in F_1;\ q \in T_{G_1}(AZ_1)$,

(c) $af(z_1, \ldots, z_k) \to q; \quad a \in A;\ f \in F_k \quad (k > 1)$;

$$q \in T_G(\{v\}) \cdot_v g_k(r_1, \ldots, r_k), \quad \text{where} \quad g_k \in G_k;\ r_i \in T_{G_1}(AZ_k) \quad (i = 1, \ldots, k).$$

*Proof.* The necessity of the first two conditions directly comes from Lemmas 3.2 and 3.3. Suppose **B** is an inverse of **A**. Then both **A** and **B** are nondeleting. Let $(p, q) \in \tau_{\mathbf{A}}$ be arbitrary. Let $n(p)$ $(n(q))$ denote the number of those vertices of $p$ $(q)$ whose label is an operation symbol with arity at least 2. Since **A** and **B** are nondeleting and $(p, q) \in \tau_{\mathbf{A}}$, $(q, p) \in \tau_{\mathbf{B}}$, we have $n(p) = n(q)$. From this it follows that the rules of **A** are of one of the three types as indicated.

For the converse, suppose that **A** satisfies all the conditions (1), (2), (3). Observe that this assumption implies that **A** is linear and nondeleting. By Theorem 1.14, there is a linear nondeleting $F$-transducer **C** equivalent to **A**. By the assumptions and the construction given in the proof of Theorem 1.17 the auxiliary variables meet at the same vertex in right side of the rules of **C**. Therefore, we can consider all the states of **C** and all the auxiliary variables to be of $\infty$-type. Let $f(c_1 z_1, \ldots, c_l z_l) \to q$ $(l > 0)$ be a rule in **C**. Then, again by the construction given in the proof of Theorem 1.17 and our assumptions, the frontier of $q$ only consists of auxiliary variables. Thus, **C** is invertible. Let us invert the rules in such a way that every auxiliary variable is taken $\infty$-type. The inverse obtained is a linear nondeleting $F$-transducer. By Theorem 1.17, it has an equivalent linear nondeleting $R$-transducer. This ends the proof.

To obtain a complete solution we would need to describe $R$-transducers equivalent to linear $R$-transducers. For our purposes, we may confine ourselves to bounded regularity preserving $R$-transducers.

**Lemma 3.6.** Let $\mathbf{A} = (T_F(X_n), A, T_G(Y_m), A', P)$ be a biaccessible $R$-transducer. Suppose that **A** is nonlinear. Let $af(z_1, \ldots, z_l) \to q(\mathbf{a}_1 z_1^{n_1}, \ldots, \mathbf{a}_l z_l^{n_l}) \in P$; $\mathbf{a}_i \in A^{n_i}$, $(i = 1, \ldots, l)$ be a multiplying rule. Especially, $\mathbf{a}_1 = (a_1, \ldots, a_{n_1})$. Let $T = \bigcap \{\mathrm{dom}(\tau_{\mathbf{A}(a_i)}) \mid i = 1, \ldots, n_1\}$ and $T_i = \tau_{\mathbf{A}(a_i)}(T)$ $(i = 1, \ldots, n_1)$, where $\mathrm{dom}(\tau_{\mathbf{A}(a_i)})$ is the domain of the transformation $\tau_{\mathbf{A}(a_i)}$. Thus, to every multipying rule there are corresponding forests $T$ and $T_i$. If for every $T$ at most one of the associated $T_i$'s is infinite then $\tau_{\mathbf{A}}$ preserves regularity.

*Proof.* Let $A = \{a_1, \ldots, a_k\}$ be the state set of **A**. Denote by $T'$ the union of the finite $T_i$'s. $T'$ is finite, say $T' = \{q_1, \ldots, q_t\}$ $q_i \in T_G(Y_m)$, $i = 1, \ldots, t$. Denote by $T_{ij}$ $(i = 1, \ldots, k; j = 1, \ldots, t)$ the set of those trees $p \in T_F(X_n)$ with $a_i p \overset{*}{\underset{\mathbf{A}}{\Rightarrow}} q_j$. The forests $T_{ij}$ are regular, consequently, there are tree automata $\mathbf{A}_{ij} = (\mathfrak{A}_{ij}, \mathscr{L}_{ij}, A'_{ij})$

with $T_{ij}=T(\mathbf{A}_{ij})$. Let $V$ denote the set of all vectors of dimension $kt$ over the set $\{0, 1\}$. Let $H=H_0 \cup H_1 \cup \dots$ be a new ranked alphabet where $H_0=F_0$; $H_l=F_l \times V^l$ $(l>0)$. Take the $F$-transducer $\mathbf{B}=(T_F(X_n), B, T_H(X_n), B', P')$ with

$$B = A_{11} \times A_{12} \times \dots \times A_{k1} \times \dots \times A_{kt}, \quad \text{where} \quad A_{ij} \text{ is the state set of } \mathbf{A}_{ij},$$

$$B' = B$$

$P'$:

(i) $$x \to (x\mathscr{L}_{11}, \dots, x\mathscr{L}_{kt}) \times \in P', \quad x \in X_n,$$

(ii) $$f \to (f^{\mathfrak{A}_{11}}, \dots, f^{\mathfrak{A}_{kt}}) f \in P', \quad f \in F_0,$$

(iii) $$f(b_1 z_1, \dots, b_l z_l) \to b(f, v_1, \dots, v_l)(z_1, \dots, z_l) \in P'$$
$$f \in F_l; \; l > 0; \; b, b_1, \dots, b_l \in B; \; v_i \in V \quad (i = 1, \dots, l);$$

where $b=(f^{\mathfrak{A}_{11}}(b_{1,1}; \dots, b_{l,1}), \dots, f^{\mathfrak{A}_{kt}}(b_{1,kt}, \dots, b_{l,kt}))$, and let $0<\alpha \leqq k$, $0<\beta \leqq t$, $v_{ij}=1$ if and only if $b_{i,j} \in A_{\alpha\beta}$, where $j=(\alpha-1)t+\beta$. $\tau_{\mathbf{B}}$ is a linear nondeleting $R$-transformation by Theorem 1.17.

Let $\mathbf{C}=(T_H(X_n), A, T_G(Y_m), A', P'')$ where $A$ and $A'$ are the same as in $\mathbf{A}$, and $P''$ is obtained in the following way.

In case of nonlinear rules:

(1) Let $af(z_1, \dots, z_l) \to r(\mathbf{a}^1 \mathbf{z}_1, \dots, \mathbf{a}^l \mathbf{z}_l) \in P$ be nonlinear. Take all possible rules $a(f, v_1, \dots, v_l)(z_1, \dots, z_l) \to \bar{r}$, where $v_i \in V$ $(i=1, \dots, l)$ and $\bar{r}$ is obtained from $r$ as follows: Let $0<\alpha \leqq k$, $0<\beta \leqq t$. If $v_{ij}=1$ $(j=(\alpha-1)t+\beta)$ and $z_i$ occurs in $r$ with state $a_\alpha$ $(i=1, \dots, l)$, then substitute $q_\beta$ for $a_\alpha z_i$ in $r$. Let $P''$ contain those rules $a(f, v_1, \dots, v_l)(z_1, \dots, z_l) \to \bar{r}$ which are linear and such that $\bar{r}$ contains an auxiliary variable if and only if it occurs in $r$.

In case of linear rules:

(2) $af(z_1, \dots, z_l) \to r \in P$ if and only if

$$a(f, v_1, \dots, v_l)(z_1, \dots, z_l) \to r \in P''; \quad v_i \in V \quad (i = 1, \dots, l).$$

(3) $ap \to r \in P$ if and only if $ap \to r \in P''$; $p \in T_{F_0}(X_n)$. We have $\tau_{\mathbf{A}}=\tau_{\mathbf{B}} \circ \tau_{\mathbf{C}}$, yielding that $\tau_{\mathbf{A}}$ preserves regularity.

**Corollary 3.7.** If $\mathbf{A}$ is nondeleting then so is $\mathbf{C}$. It is known that the composition of linear nondeleting $R$-transformations is a linear $R$-transformation. Thus, $\tau_{\mathbf{A}}$ is a linear $R$-transformation if $\mathbf{A}$ is nondeleting, further, one can effectively construct an equivalent linear $R$-transducer.

**Lemma 3.8.** Let $\mathbf{A}$ be a biaccessible bounded nonlinear $R$-transducer. Let a nonlinear rule of $\mathbf{A}$ be $af(z_1, \dots, z_l) \to \bar{q}(\mathbf{a}_1 z_1^{u}, \dots, \mathbf{a}_l z_l^{p_l})$. Suppose that this rule multiplies $z_1$, i.e. $u>1$. Put $\mathbf{a}_1=(a_1, \dots; a_u)$. Denote by $T$ the forest $T=\bigcap \{\mathrm{dom}\,(\tau_{\mathbf{A}(a_i)});\ i=1, \dots, u\}$. If $\tau_{\mathbf{A}}$ preserves regularity then $T$ is finite.

*Proof.* Since $\mathbf{A}$ is bounded so are $\mathbf{A}(a_i)$ and $\mathbf{A}(a)$ $(i=1, \dots, u)$. If $\tau_{\mathbf{A}}$ preserves regularity then $\tau_{\mathbf{A}(a_i)}$ preserves regularity as well. Therefore, it is enough to deal with the transducer $\mathbf{A}(a)$ instead of $\mathbf{A}$. Suppose that $T$ is infinite although the assumptions are fulfilled. Since $T$ is a finite intersection of domains, $T$ is regular. Let $s\in T_F(X_n\cup Z_1)$ be a tree in which $z_1$ occurs exactly once. Then

$$s^i = \underbrace{s\cdot_{z_1}s\cdot_{z_1}\dots\cdot_{z_1}s}_{i\text{-times}}.$$

Since $T$ is infinite and regular, there is an $r\in T$ such that $r=r_1(r_2(r_3))$ where $r_3\in T_F(X_n)$, $r_1, r_2\in T_F(X_n\cup Z_1)$, $h(r_2)>0$ and for every $k$, $r^{(k)}=r_1(r_2^k(r_3))\in T$. Then $T'=\{r^{(k)}; k=1, 2, \dots\}$ is an infinite regular subset of $T$. Since $T'\subseteq T$, $T_i=\tau_{\mathbf{A}(a_i)}(T')\neq\emptyset$ $(i=1, \dots, u)$, and by the boundedness, $T_i$ is infinite. Since $\tau_{\mathbf{A}}$ preserves regularity by supposition, all the forests $T_i$ are regular. Let $p_2, \dots, p_l\in T_F(X_n)$ be trees with $a_{ij}p_i\overset{*}{\underset{\mathbf{A}}{\Rightarrow}}q_{ij}$ $i=2, \dots, l$ and $j=1, \dots, n_i$. Let $p=f(z_1, p_2, \dots, p_l)$. It holds that $ap\overset{*}{\underset{\mathbf{A}}{\Rightarrow}}\bar{q}(a_1z_1, \dots, a_uz_1)$. Set $T''=p\cdot_{z_1}T'$ $(=\{p(r_1(r_2^k(r_3))), k=1, 2, \dots\})$. $T''$ is regular, and so is $K=\tau_{\mathbf{A}}(T'')$. Let $\mathbf{B}$ be a tree automaton recognizing $K$. Since all the forests $T_i$ are regular, there is a $q\in K$, $q=\bar{q}(a_1, s_2, \dots, s_u)$, with $s_i\in T_i$ $(i=1, \dots, u)$ such that $h(s_1)$, $h(s_l)$ are greater than the number of states of $\mathbf{B}$. Then $s_i$ $(i=1, 2)$ can be written in the form $s_{i_1}(s_{i_2}(s_{i_3}))$ such that $h(s_{i_2})>0$ and $t_i^{(k)}=s_{i_1}(s_{i_2}^k(s_{i_3}))\in T_i$ for every $k$. We have $q=\bar{q}(t_1^{(k)}, t_2^{(l)}, s_3, \dots, s_u)\in K$. Let $n_1$ denote the number of states of $\mathbf{A}$, and $n_2$ the maximum of the heights of the trees occurring on the right side of the rules in $P$. Choose $t_1^{(k)}$ and $t_2^{(l)}$ as follows:

$$h(t_1^{(k)}) > n_2h(p(r))+1,$$

$$h(t_2^{(l)}) > n_2(n_1+1)h(t_1^{(k)})+n_2(n_1+1)h(\bar{q}(z_1, z_2, s_3, \dots, s_u)).$$

Let $q^*=\bar{q}(t_1^{(k)}, t_2^{(l)}, s_3, \dots, s_u)$, where $t_1^{(k)}, t_2^{(l)}$ are the trees given above. It is obvious that $q^*\in K$. There is a $p_n\in T''$ $(p_n=p(r_1(r_2^n(r_3))))$ with $ap_n\overset{*}{\underset{\mathbf{A}}{\Rightarrow}}q^*$. Denote by $d_1$ and $d_2$ the root of $t_1^{(k)}$ and $t_2^{(l)}$, resp. There are vertices $c_1, c_2$ in $p_n$ such that $d_i$ $(i=1, 2)$ is obtained when the translation process arrives at $c_i$. The lengths of the paths from the root to $c_1$ and $c_2$ cannot exeed $(n_1+1)h(\bar{q}(z_1; z_2, s_3, \dots, s_u))$ because $\mathbf{A}$ does not have jumping cycles. Let $\bar{r}_i$ $(i=1, 2)$ denote the subtree belonging to $c_i$. The subtree $r_3$ cannot contain $c_1$ and $c_2$, or even, $c_1$ and $c_2$ are located on the path to the root of $r_3$. (The reason is the height of the trees $t_1^{(k)}$, $t_2^{(l)}$.) Thus, either $\bar{r}_1$ is a subtree of $\bar{r}_2$ or conversely.

Suppose first that $\bar{r}_1\in\operatorname{sub}(\bar{r}_2)$. Then

$$h(\bar{r}_2) < h(\bar{r}_1)+(n_1+1)h(\bar{q}(z_1, z_2, s_3, \dots, s_u))$$

because $\bar{r}_1$ is a subtree of $\bar{r}_2$ and that part of $\bar{r}_2$ not contained by $\bar{r}_1$ cannot be higher than $(n_1+1)h(\bar{q}(z_1, z_2, s_3, \dots, s_u))$. Since $\bar{r}_1$ is translated to $t_1^{(k)}$ and $\bar{r}_1$ is a subtree of $\bar{r}_2$, an upper bound for the height of the trees that can be translated from $\bar{r}_2$ is

$$n_2(n_1+1)(h(t_1^{(k)})+h(\bar{q}(z_1, z_2, s_3, \dots, s_u))).$$

However, this is impossible by the choice of the trees $t_1^{(k)}$ and $t_2^{(l)}$.

If $\bar{r}_2\in\operatorname{sub}(\bar{r}_1)$ then $h(\bar{r}_2)<h(\bar{r}_1)$. Thus, a tree translated from $\bar{r}_1$ is at least as high as a tree translated from $\bar{r}_2$. Since $t_2^{(l)}$ is translated from $\bar{r}_2$, $h(\bar{r}_2)>h(t_2^{(l)})/n_2$, and a tree translated from $\bar{r}_1$ is at least $h(\bar{r}_2)/(n_1+1)$ high. It follows that the trees translated from $\bar{r}_1$ are at least $h(t_1^{\prime(k)})+h(\bar{q}(z_1, z_2, s_3,\dots, s_u))$ high. This contradicts the choice of $t_1^{(k)}, t_2^{(l)}$.

From our result we immediately obtain.

**Theorem 3.9.** It is decidable for a bounded $R$-transducer **A** if **A** preserves regularity.

**Theorem 3.10.** It is decdiable if an $R$-transducer is invertible, and the inverse, if exists, can be effectively constructed.

Further results on regularity preserving $R$-transducers can be found e.g. in [1].

DEPT. OF COMPUTER SCIENCE
A. JÓZSEF UNIVERSITY
ARADI V. TERE 1
SZEGED, HUNGARY
H—6720

## References

[1] ÉSIK, Z., Decidability results concerning tree transducers II., Acta Cybernet., 6 (1983), 303—314.
[2] GÉCSEG, F., M. Steinby Algebraic theory of tree automata I., Matematikai lapok 26 (1975), 169—207, in Hungarian.
[3] GÉCSEG, F., M. Steinby, Algebraic theory of tree automata II., Matematikai lapok 27 (1976), 283—336, in Hungarian.
[4] GÉCSEG, F., Tree transformation preserving recognizability, Colloquia Mathematica Societatis János Bolyai 28. Finite Algebra and Multiple-Valued Logic, 251—273.

*(Received May 6, 1986)*

# Syntactic pattern recognition in the HLP/PAS system

T. GYIMÓTHY AND J. TOCZKI

## Abstract

In this paper a syntactic pattern recognition application of the HLP/PAS system is presented. The system has originally been developed for compiler generation. It can generate both one-pass and multi-pass compilers from attribute grammar specifications. The generated compilers use LL (1) or LALR (1) parsing methods. However, in many cases, patterns can be described only with ambiguous grammars. For this reason the HLP/PAS system was extended with a backtrack parser generator. The generated backtrack parsers use the LL (1) parsing tables to eliminate some of the unnecessary backtracks. Another characteristic of these parsers is that the parsing can be controled by the evaluated attributes. As an illustration, an attribute grammar description is presented for normal ECG waveforms.
*key words:* attribute grammars, syntactic pattern recognition, attribute evaluation.

## 1. Introduction

So far there have been several attempts for describing patterns by attribute grammars [8], [9], [11]. It is not surprising as both the context-free and the context-sensitive characteristics of the patterns can be described by attribute grammars. The numerical data of the patterns can conveniently be computed by semantic rules so attribute grammars can create a connection between syntactic and statistic methods of the pattern recognition.

While there are several complete compiler generator systems based on attribute grammars [4], [7], to our knowledge, there is no such a system for pattern recognition tasks. In a complete system a metalanguage is needed for the specification of the attribute grammars. It is practical if the primitives of the patterns can be described as lexical tokens in this metalanguage. The parser of the metalanguage has to check the formal correctness of the specifications e.g. the consistent using of the attributes. The system must generate a parser and an attribute evaluator, too. In contrary to the usual complier generators, in a pattern recognition system, the construction of backtrack parsers is needed. Therefore, the HLP/PAS system [5], [10], which has originally been developed for compiler generation, was extended with a backtrack parser generator. In this paper we give a description of this extended system. In more detail, section 2 gives a short description of the original system, section 3 contains the specification of the ECG grammar. In section 4 the structure of the generated backtrack parsers is described, while section 5 contains some observation about further research of this topics. Finally we give a short summary of the paper.

## 2. The HLP/PAS system

As it was already mentioned, the HLP/PAS system was originally developed for compiler generation. There are two metalanguages in the system for the lexical and syntactic-semantic descriptions of grammars. The lexical units (tokens) can be defined by regular expressions in the lexical metalanguage. In programming languages the usual tokens are identifiers, numbers etc. In the pattern descriptions the "primitives" [2] are the lexical units. The system generates finite automata to recognize these tokens. The generated lexical analyzer is a procedure of the complete compiler. The specification of an attribute grammar can be described in the syntactic-semantic metalanguage of the system. The semantic assignments in the description of an attribute grammar are Pascal-like expressions and procedure callings as the generated compilers are complete Pascal programs. An attribute grammar definition in the HLP/PAS system begins with the declaration of these procedures. After this the names and the types of the synthesized and inherited attributes can be defined. Both standard Pascal and user defined types can be used as the types of these attributes. Then the nonterminal declaration part follows, in which the nonterminals and the names of the attributes associated with them are described. After this the tokens and the terminals of the grammar are defined. Finally, in the last part of the specification, the syntactic rules and the semantic assignments are described. There are conditional statements which can be associated with the rules of the grammar. These statements can be applied to send messages during the compilation by means of evaluted attributes and are also used to control the generated backtrack parsers (see 4). The code generator statements generate the target code in the constructed compilers. Of course these statements also use the evaluated attributes. The evaluation time of a conditional statement is determined only by attribute dependencies while the evaluation sequence of the code-generator statements can be prescribed by the user. The system contains a simple error-recovery method which can be influenced by the definition of the grammar. If a set of terminal or token symbols (SKIP-set) is connected to a nonterminal and there is a syntactic error in the "subtree" rooted in this nonterminal during the parsing then the parser reads the input until it finds an element of the SKIP set. This symbol will be the next input symbol and the corresponding subtree is deleted (Panic method). The parser of the metalanguage always checks the formal correctness of a specification e.g. the name conflicts, the existence of superflous nonterminals, the consistence of the attribute assignments etc. The system can automatically generate so called copy rules for the simple transport of attribute values if the assignment is determined unambiguously. Finally, from a correct specification the parser of the metalanguage constructs files for other moduls of the system. As we mentioned earlier, both one-pass and multi-pass compliers can be generated. The one-pass compilers use LL (1) parsing method and L-attribute evaluation strategy [1]. In the multi-pass compilers LALR (1) parsing method and a modified version, MOAG [4] of the OAG [6] attribute evaluation method are applied.

## 3. An attribute grammar for normal ECG waveforms

On the basis of [9] an attribute grammar is presented for the description of normal ECG waveforms in the HLP/PAS system. This grammar is used to illustrate the backtrack parsing in the system. The first step in a description of a pattern is to determine the set of the primitives. These primitives are the terminal symbols of the grammar. First an ECG waveform is approximated with line segments [3]. The line segments are partitioned into pieces nearly of the same size (these are the primitives). This partition is carried out by using a UNIT segment (see Figure 1). A slope symbol is associated with each primitive as follows:

| | | | |
|---|---|---|---|
| if | $v_H < \varphi_P \leqq v_S$ | then | SP |
| if | $v_S < \varphi_P \leqq v_I$ | then | IP |
| if | $\varphi_P > v_I$ | then | LP |
| if | $-v_H > \varphi_P \geqq -v_S$ | then | SN |
| if | $-v_S > \varphi_P \geqq -v_I$ | then | IN |
| if | $\varphi_P < -v_I$ | then | LN |
| if | $-v_H < \varphi_P \leqq v_H$ | then | HP |

were $\varphi_P$ is the angle of the line segment $S_P$ with the horizontal axis and $v_H$, $v_S$, $v_I$ are predefined constant angles. Each primitive in a segment has the same size and if the UNIT not too large then there is not large difference between the size of the primitives of different segments. Moreover each primitive has a duration which is the projection of the primitive to the time axis (Figure 1.).

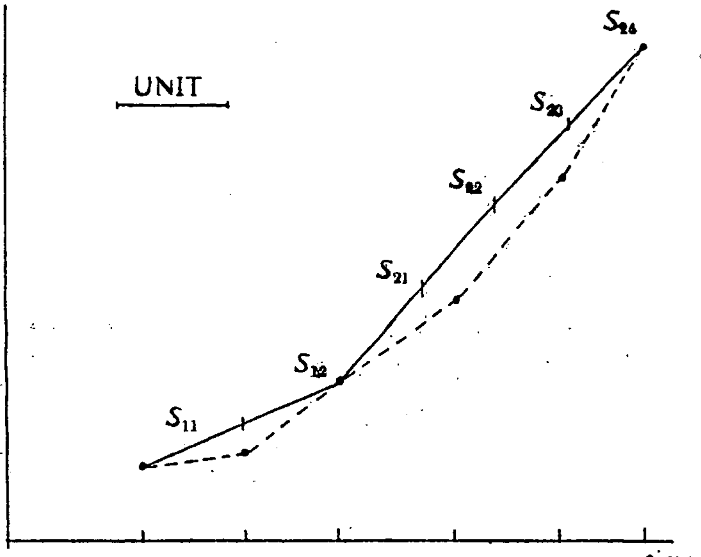


*Figure 1*

In Figure 1 the dashed lines indicate the pieces of a waveform and the solid lines are the segments. We can see that the duration of the primitives $S_{11}$, $S_{12}$ is 1 and 3/4 of the primitives $S_{21}$, $S_{22}$, $S_{23}$, $S_{24}$. If the $v_H=10°$, $v_S=30°$ angle constants are used then the waveform can be coded as follows

(SP, 1)(SP, 1)(IP, 0.75)(IP, 0.75)(IP, 0.75)(IP, 0.75).

In Appendix A a grammar is given for the description of normal ECG. In the description $X^n$ denotes $X...Xn$ times. The grammar is ambiguous. For example consider the rules

$$11.\ T \rightarrow FGH;\ 12.\ F \rightarrow K^4|K^3|K^2;\ 13.\ G \rightarrow I^3|I^2|I|\varepsilon;$$

$$16.\ K \rightarrow \neq IP \neq DIG| \neq SP \neq DIG| \neq HP \neq DIG \quad \text{and}$$

$$17.\ I \rightarrow \neq HP \neq DIG| \neq SP \neq DIG| \neq SN \neq DIG;$$

Starting from the nonterminal T both the

$$T \rightarrow FGH \rightarrow K^4GH \rightarrow K^4IH \ldots \quad \text{and the}$$

$$T \rightarrow FGH \rightarrow K^3GH \rightarrow K^3I^2H \ldots$$

derivation leads to the $(\neq HP \neq DIG)^5$ string. The grammar in Appendix A is augmented with attributes and semantic assignments. These assignments compute the durations of the cardiac cycles from that of the primitives and determine the maximal durations of cycles (maxdur, mindur). An ECG is normal if $\frac{\text{maxdur-mindur}}{\text{maxdur}} > 0.1$. In Appendix B the description of the augmented ECG grammar is presented in the metalanguage of the HLP/PAS system. The description does not contain the complete grammar, only the most important parts of the specification are given. Of course using more attributes in the description further characterictics of ECG waveforms can be analysed.

## 4. The generated backtrack parsers

As it was already mentioned, the one-pass compiler generator part of the HLP/PAS was extended with a backtrack parser generator. First the structure of the one-pass compilers generated originally is outlined. For each nonterminal of the grammar a Pascal procedure is constructed. The inherited and synthesized attributes of a nonterminal are the input and output parameters of the procedure corresponding to this nonterminal. For example consider the following rules:

$$X_0 \rightarrow X_{11}X_{12}\ldots X_{1n_1}$$
$$\vdots$$
$$X_0 \rightarrow X_{q1}X_{q2}\ldots X_{qn_q}$$

The structure of the generated procedure is:

```
    procedure $X_0\$(I(X_0)$; var $S(X_0))$;
      record $X_{11}$ declaration of $A(X_{11})$; end
        ⋮
      record $X_{qn_q}$ declaration of $A(X_{qn_q})$; end
      begin
      if $SY\$ \in S_1$ then begin
                  eval $(I(X_{11}))$; $X_{11}$ $\$(I(X_{11})$; $S(X_{11}))$;
                  ⋮
                  eval $(I(X_{1n_1}))$; $X_{1n_1}$ $\$(I(X_{1n_1})$; $S(X_{1n_1}))$; end else
                  ⋮
      if $SY\$ \in S_q$ then begin
                  eval $(I(X_{q1}))$; $X_{q1}$ $\$(I(X_{q1})$; $S(X_{q1}))$;
                  ⋮
                  eval $(I(X_{qn_q}))$; $X_{qn_q}$ $\$(I(X_{qn_q})$; $S(X_{qn_q}))$;
        end else error;
    eval $(S(X_0))$;
end of procedure $X_0\$$;
```

where $I(X_{ij})$, $S(X_{ij})$, $A(X_{ij})$ denote the inherited, synthesized and the all attributes of the nonterminal $X_{ij}$, respectively. For each different right-hand side nonterminal a record structure is generated. The variable $SY\$$ contains the current input symbol. The corresponding alternative is determined by the condition $SY\$ \in S_i$, where $S_i = \text{FIRST}_1(X_{i1}, \ldots, X_{in_i}) \oplus_1 \text{FOLLOW}_1(X_0)$. In the blocks of the alternativies, eval $(I(X_{ij}))$ denotes the evalution of the inherited attributes of the nonterminal $X_{ij}$. The places of the conditional and code generator statements in the alternativies are determined by attribute dependencies and the prescriptions of the user (see 2. section). The callings of the lexical procedure are also in the blocks of the alternatives. Instead of building the parse tree, only recursive procedure callings are executed during the parsing.

In the backtrack version of the generated compilers the instances of the procedures in a calling sequence are numbered (see Figure 2.).

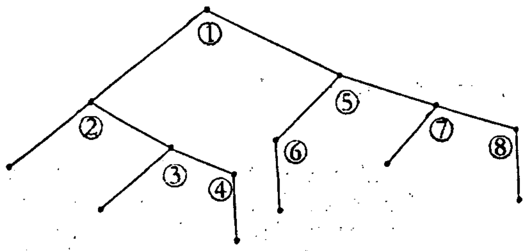


*Figure 2*

The greatest sequence number of the instances is stored in the global variable NUM$. In each procedure there is a local variable (NUMX) to store the number of the actual instance in the calling sequence. The TRUE value of the global Boolean variable BTRACK denotes that the parser is in backtracking mode. During the parsing a global stack is handled. The $I$-th element of this stack gives the number of the alternative chosen in the $I$-th instance. A flag denotes if another alternative with greater number can be chosen in this instance. Of course only those alternatives are considered for which the condition $SY\$\in S_i$ is true. In the global variable LPOINT the number of an instance is stored. In backtracking mode the new alternative will be chosen from this instance. Finally in each procedure there is a pointer (PT) to denote the position of the current input symbol at the entry of the corresponding instance. If there is an error in the $K$-th instance then BTRACK=TRUE and this instance is terminated. In the recursive calling structure the procedure instances are terminated until the condition NUMX>LPOINT is true. If NUMX<LPOINT then using the NUMX, NUMX+1, ..., LPOINT−1 elements of the stack and the pointers PT the necessary part of the parsing is reconstructed. In the instance indicated by LPOINT the new alternative is chosen. The assignments NUM$=NUMX, BTRACK=FALSE are executed and in the variable LPOINT the new backtracking point is stored. In [8] a backtrack parser was presented for pattern recognition. The main advance of the parser presented in this paper against that of [8] is that using the LL (1) conditions a lot of useless backtracks can be eliminated. Of course there is a cost of the computation of the LL (1) tables but this computation happens only ones in meta-compiling time.

To illustrate the backtrack parser consider the following structure of the ECG grammar: $ST\rightarrow I^{10}|I^9|I^8|I^7|I^6$.

It can be described with the following three rules:

```
  i) ST=I_LIST;
     DO
     I_LIST.length:=0;
     END
 ii) I_LIST=I  I_LIST;
     DO
     dur:=I.dur+I_LIST.dur;
     I_LIST.length:=length+1;
     COND
     if I_LIST.length>10 then BACKTRACK;
     END
iii) I_LIST=I;
     COND
     if length<6 then BACKTRACK;
     END
```

The inherited attribute length is used to count the I elements. The backtrack is controled by this attribute. For example if the rule ii) was applied ten times then a backtrack is executed for the nonterminal I_LIST and the alternative iii) is chosen instead of ii). On the other hand if in the rule iii) the condition length <6 is true then after several backtracks the instance of the nonterminal ST is terminated in backtracking mode. These redundant steps can also be eliminated if the number of I elements is stored in a synthesized attribute of ST and the condition length $<6$ is applied in the rule i). This solution can be seen in Appendix C.

## 5. Further research

The backtrack parsers presented in this paper use L-attribute evaluation method. This method can be applied to languages the elements of which depend on their left-hand side environments. It often holds in the case of programming languages but not in the case of pattern descriptions. A subpattern usually depends on both its left- and right hand side environments. Hence multi-pass attribute evaluators are needed. In such type parsers, attributed parsing trees are constructed to store the value of the evaluated attributes and the structure of the parsing. As we mentioned it earlier, in many cases the patterns can conveniently be described only by ambiguous grammars. Therefore the development of a multi-pass, backtrack parser generator in the HLP/PAS system would be needful. Because such type parsers work usually very slowly, in our opinion, a combination of the pass-directed and the dinamic attribute evaluation strategies is needed. When backtrack, some attribute values have to recompute. In these cases the appliement of the dinamic attribute evaluation method is efficient. Only those attributes must be recomputed the values of which are changed during the backtrack. In the other part of the grammar (and usually it is the larger part) a pass-directed evaluation method can be used e.g. MOAG [4].

## 6. Conclusions

In this paper a syntactic pattern recognition system was presented. The input of the system is a complete description of a pattern by attribute grammar. From this specification the recognizer of the pattern is generated. In the description of patterns ambiguous grammars can also be used. The generated parsers use the LL (1) tables so a lot of redundant backtracks can be eliminated. Further characteristic of the generated parsers is that the parsing can be influenced by the evaluated attributes. Calling the start symbol of an ambiguous grammar repeatedly the all possible derivations of the grammar can be constructed for a given input. The complete system was implemented on Pascal language on IBM—370 and IBM XT compatible computers.

## Acknowledgements

We wish to thank Árpád Makay and Zoltán Fülöp for their constructive comments on this paper.

## Appendix A

1. S=NORMAL_ECG
2. NORMAL_ECG=CARDIAC_CYCLE NORMAL_ECG
3. NORMAL_ECG=R
4. CARDIAC_CYCLE=RS ST T TP P PR Q

5. R=C D
6. RS=C D E
7. C= ≠LP≠ DIG C|≠LP≠DIG
8. D= ≠LM≠ DIG D|≠LM≠ DIG
9. E= ≠LP≠ DIG E|≠IP≠ DIG E|≠LP≠ DIG|≠IP≠ DIG|$\varepsilon$
10. ST=$I^{10}|I^{9}|I^{8}|I^{7}|I^{6}$
11. T=F G H
12. F=$K^{4}|K^{3}|K^{2}$
13. G=$I^{3}|I^{2}|I|\varepsilon$
14. H=$M^{4}|M^{3}|M^{2}|M|\varepsilon$
15. M= ≠IM≠ DIG|≠SM≠ DIG
16. K= ≠IP≠ DIG|≠SP≠ DIG|≠HP≠DIG
17. I= ≠HP≠ DIG|≠SP≠ DIG|≠SM≠DIG
18. TP=$I^{14}|I^{13}|I^{12}|I^{11}|I^{10}|I^{9}|I^{8}$
19. P=T
20. PR=$I^{4}|I^{3}|I^{2}|I|\varepsilon$
30. Q=$L^{3}|L^{2}|L|\varepsilon$
31. L= ≠IM≠DIG|≠LM≠DIG
32. DIG=NUMBER

## Appendix B

```
ATTRIBUTE GRAMMAR ECG
(* B+  BACKTRACK OPTION IS ON *)
PASCAL DECLARATIONS ARE
  PROCEDURE BF (a, b: INTEGER; VAR c: BOOLEAN);
  BEGIN
  IF (a-b)/a> 0.1 THEN c:=TRUE ELSE c:=FALSE;
  END;
  PROCEDURE MAXF (VAR a: INTEGER; c, b: INTEGER);
  BEGIN IF b> c THEN a:=b ELSE a:=c; END;
  PROCEDURE MIN (VAR a: INTEGER; b, c: INTEGER);
  BEGIN
  IF b>c THEN a:=c ELSE a:=b;
  END;
SYNTHESIZED ATTRIBUTES ARE
  maxdur, mindur, dur :INTEGER;
  fl: BOOLEAN; val:INTEGER;
INHERITED ATTRIBUTES ARE
  length: INTEGER;
NONTERMINALS ARE
  ECG HAS fl;
  NORMAL_ECG HAS maxdur, mindur;
  CARDIAC_CYCLE, R, RS, ST,T,TP, P, PR, Q, C, D, E, DIG, F, G, H HAVE
  dur;
  LP_PAIR, LM_PAIR, IP_PAIR, IM_PAIR, HP_PAIR, SP_PAIR, SM-PAIR
    HAVE dur;
  I_SET, K_SET, M_SET, L_SET HAVE dur;
  I1_LIST, I2_LIST, I3_LIST, I4_LIST, L_LIST, K_LIST, M_LIST HAVE
    length, dur;
  TOKENS ARE
  NUMBER HAS val;
  TERMINALS ARE
    "LP", "LM", "IP", "IM", "HP", "SP", "SM";
PRODUCTIONS ARE
ECG= NORMAL_ECG;
  DO
  fl< -BF(NORMAL_ECG.maxdur, NORMAL_ECG.mindur, fl);
  END
NORMAL_ECG=CARDIAC_CYCLE NORMAL_ECG;
  DO
    maxdur< -MAXF(maxdur, CARDIAC_CYCLE.dur, NORMAL_ECG.
    maxdur);
    mindur< -MINF (mindur, CARDIAC_CYCLE.dur, NORMAL_ECG.
    mindur);
  END
NORMAL_ECG=R;
  DO
     maxdur :=0;
     mindur :=0;
  END
    CARDIAC_CYCLE=RS  ST  T TP  P PR Q;
  DO
    dur := RS.dur+ST.dur+T.dur+TP.dur+P.dur+PR.dur+Q.dur;
  END
      ⋮
```

## Appendix C

```
 i) ST=I_LIST
      DO
      I_LIST.length:=0;
      COND
      IF I_LIST.slength <6 THEN BACKTRACK;
      END

ii) I_LIST=I I_LIST;
      DO
      dur:= I.dur+I_LIST.dur;
      I_LIST.length:=length+1;
      slength=I_LIST.slength+1;
      COND
      IF I_LIST.length > 10 THEN BACKTRACK;
      END
```

```
iii) I_LIST=I;
     DO
     slength:=1;
     END
```

RESEARCH GROUP ON THEORY OF AUTOMATA
HUNGARIAN ACADEMY OF SCIENCES
SOMOGYI U. 7.
SZEGED, HUNGARY
H—6720

## References

[1] Bochmann, G. V., Semantic evaluation from left to right, Commun. Ass. Comput. Mach., vol 19, pp. 55—62, Feb. 1976.
[2] Fu, K. S., (ed), Syntactic pattern recognition, applications, Springer-Verlag, New York/Berlin 1977.
[3] Gritzali, F. and G. Papakonstantinou, A fast piecewise linear approximation algorithm, Signal Processing, vol. 5, pp. 221—227, 1983.
[4] Gyimóthy, T., E. Simon and Á. Makay, An implementation of the HLP, Acta Cybernetica, Tom 6, Fasc. 3, pp. 315—327.
[5] Gyimóthy, T., Kocsis, F., Makay, Á., Simon, E. and Toczki, J., The compiler generator HLP/PAS, Computer and Automation Institute, Hungarian Academy of Sciences, Report, to be published.
[6] Kastens, U., Ordered Attribute Grammars, Acta Informatica 13, 229—256, 1980.
[7] Koskimies, K. and Paakki, J., HLP84-Semantic metalanguage and its implementation, University of Helsinki, Report C—1983—69.
[8] Papakonstantinou, G., An interpreter of attribute grammars and its application to the waveform analysis, IEEE Transactions on Software Engineering, vol. SE-7, No. 3, pp. 279—283, May 1981.
[9] Skordalakis, E. and Papakonstantinou, G., Toward an attribute grammar for the description of ECG waveforms, 7-th International Conference on Pattern Recognition, 1984.
[10] Toczki, J., et al., On the Pascal implementation of the HLP, Proc. of 4th Hungarian Computer Science Conference, Győr, 1985, 12 pp.
[11] You, K. C. and Fu, K. S., A syntactic approach to shape recognition using attributed grammars, IEEE Transactions on Systems, Man, and Cybernetics, vol. SMC-9, No. 6, pp. 334—345, 1979.

*(Received May 6, 1986)*

# Software Specification Methods and Attribute Grammars

GUENTER RIEDEWALD, PETER FORBRIG

## 1. Introduction

An important problem of modern computer science is the development of quality software. The necessary design of large systems in the 1980s requires other methods than the design of smaller systems in the 1960s and becomes more and more an engineering problem.

There are many different methods to cope with this problem. These methods range for example from data driven program development by Jackson [Jac 75] to information hiding or data encapsulation by Parnas [Par 72].

Computer science calls for programming development systems, which support the programming development process by the computer itself.

After a short suvery of the fundamentals of software specification we want to illustrate the use of attribute grammars in commercial data processing. It will be presented how the well known methods of data driven programming and data encapsulation, usually classified as contrary concepts, can be combined by using attribute grammars with abstract data types.

Such attribute grammars represent specifications in a clearly readable form. The grammatical definition formalism is used only to the necessary extend. Many implementation details are encapsulated in abstract data types.

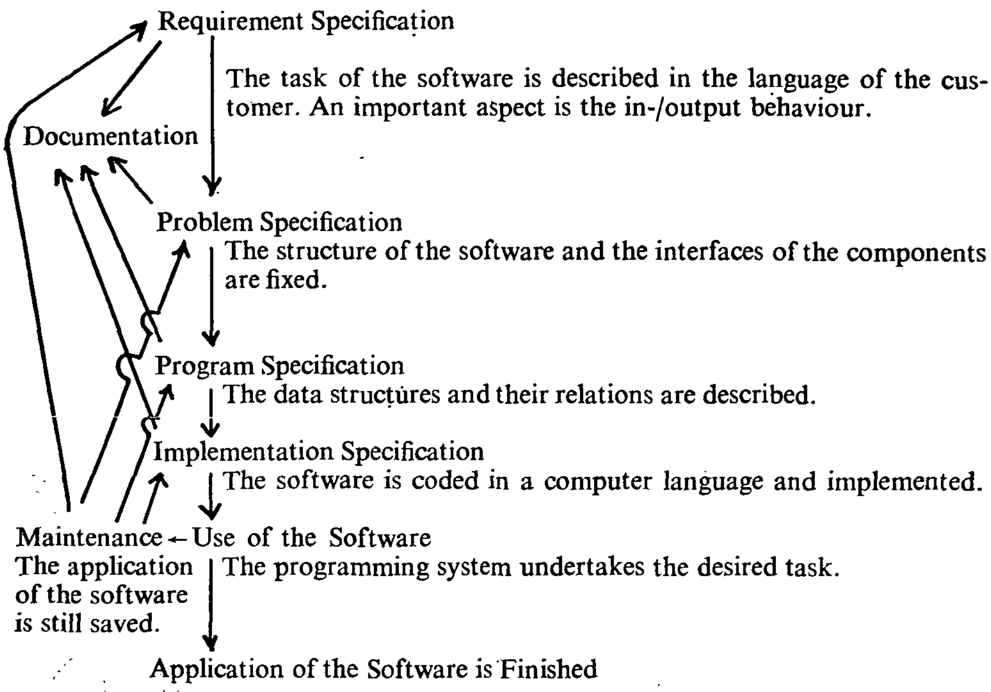
## 2. Software Specification

### 2.1. Life Cycle Model

We will use the life cycle model for demonstrating the problems of software design in different stages of its development. Our point of view is demonstrated in Figure 1.

The arrows in Figure 1 represent possible relations between components of the life cycle.

The first problem in software design is to define the exact task of a programming system. In all other phases of development the corresponding specifications have to be compared and verified with this requirement specification.


Requirement Specification
Documentation
Problem Specification
Program Specification
Implementation Specification
Maintenance
Use of the Software
Application of the Software is Finished

The task of the software is described in the language of the customer. An important aspect is the in-/output behaviour.

The structure of the software and the interfaces of the components are fixed.

The data structures and their relations are described.

The software is coded in a computer language and implemented.

The application of the software is still saved.

The programming system undertakes the desired task.

*Figure 1.* Life Cycle Model of Software

Figure 1 also demonstrates that the development of software is not finished after its implementation. Maintenance of software is of utmost importance. Experiences in software engineering have shown, that maintenance costs can be higher than all earlier development costs.

If programming systems are not developed in a good manner according to maintenance, the application of the software becomes more and more complicated.

According to Figure 1 software maintenance can influence all specifications of a project. The simplest modifications may have an influence on implementation only. More complex changes are even more difficult and it is possible, that not all specifications are updated in the right way.

In this case different descriptions contain contradictions and the application of the project is very difficult.

The same problems may occur during adaptation of software for further customers, which is necessary from the economic point of view.

Therefore, modern software engineering requires methods proving the different descriptions of software and tools generating one specification from the other automatically or in a dialogue with the sotware engineer.

## 2.2. Specification Methods

### 2.2.1. Fundamentals of Specification Methods

Nowadays, some methods well known from semantic definition of programming languages can be found in software specification reports. This is not surprising because an exact semantic definition of a programming language will also define the semantics of all its programs. Therefore, it is logical to use well known sematic definition methods for describing software.

Beside specification in natural languages, algebraic, logical and denotational specifications are the fundamentals of many methods.

To illustrate the principles of these methods we will use a simple example, the modified telegram problem. The original problem was already studied in [Heh 83], [Jac 75], [Noo 75] and [Rec 84].

### 2.2.2. Natural Language Specification

Specifications in natural languages are easily understood, whereas formal languages are more difficult to understand. But natural languages have the disadvantage, that they are ambiguous. Therefore it is very difficult to write down a precise specification in a natural language.

*Natural Language Specification of a Simple Telegram Problem*

A program is required to process a stream of telegrams. This stream is available as a sequence of words and spaces. The stream is terminated by the occurrence of the empty telegram. Each telegram is delimited by the symbol "$\ast\ast\ast\ast$". The telegrams are to be processed to determine for each telegram the number of words with more than twelve and the number of words with less than twelve characters. The telegrams together with the statistics have to be stored on an outputfile by eliminating all but one space between the words. The longest possible word has twenty characters. For simplicity telegram streams containing telegrams with words longer than twenty characters are omitted.

### 2.2.3. Logical Specification

Algebraic specifications conceive programs as being abstract data types and therefore heterogeneous universal algebras. According to the static character of algebras the specification of a program means to define the structure of input and output data and the ralations between them. For this purpose sorts representing data types, operation symbols describing the "rough" structure of data and auxiliary operation symbols describing "details" of data are introduced.

Properties of operations assigned to operation symbols in concrete algebras, and therefore properties of data, are described by axioms (systems of term equations).

Auxiliary operation symbols and axioms are also used for describing the dependencies between input and output data.

We can get a shorter description using a method known from the definition of programming languages (see e.g. [Rie 85]). Then the input telegram stream is considered as a program. The concrete semantic meaning is the output telegram stream. In this case the structure of the input telegram streams is defined only. By axioms all input streams are grouped into equivalence classes. Each class represents the abstract semantic meaning of the elements of the class. Therefore the axioms must be such that a class contains all input telegram streams differing only by the number of spaces between the words. Furthermore a class must be defined containing all erroneous input telegram streams.

Fundamental work was done by Goguen and Thatcher [Gog 74] and also by Guttag [Gut 75]. The shown approaches are not the only ones. Other possible algebraic approaches are proposed for example by Guttag and Horning [Gut 78] using routines or by Mallgren [Mal 80] using event algebras.

*Algebraic Specification of the Simple Telegram Problem Sorts*:

| | | |
|---|---|---|
| $C$ | = | Character Data Type |
| $W$ | = | Word Data Type |
| $WS$ | = | Word Sequence Data Type |
| $T$ | = | Input Telegram Data Type |
| $TS$ | = | Input Telegram Sequence Data Type |
| $S$ | = | Input Telegram Stream Data Type |
| $OT$ | = | Output Telegram Data Type |
| $OTS$ | = | Output Telegram Sequence Data Type |
| $OS$ | = | Output Telegram Stream Data Type |
| $P$ | = | Program Data Type |
| $N$ | = | Integer Data Type |
| $B$ | = | Boolean Data Type |

*Operation Symbols*

Formally only operation symbols are defined here. But our comments will already give an interpretation to increase the readability of the definitions.

char: $\rightarrow$ char
{char $\in \{A, \ldots, Z, a, \ldots, z, 0, \ldots, 9\}$ constructs the corresponding character.}

dig: $\rightarrow$ dig
{dig $\in \{0, \ldots, 9\}$}

bool: $\rightarrow$ bool
{bool $\in$ {true, false}}

sword: $C \rightarrow W$
{Constructs a simple word consisting of one character only.}

word: $W \times C \rightarrow W$
{Constructs a word by concatenating a word and a character.}

sseq: $W \rightarrow WS$
{Constructs a simple word sequence consisting of one word only.}

seq: $WS \times W \rightarrow WS$
{Constructs a word sequence by concatenating a word sequence, a space and a word.}

eseq: $WS \rightarrow WS$
{Constructs a word sequence by appending one space to a word sequence.}

tel: $WS \rightarrow T$
{Constructs from a word sequence a telegram by appending "****".}

stseq: $T \rightarrow TS$
{Constructs a simple telegram sequence consisting of one telegram only.}

tseq: $TS \times T \rightarrow TS$
{Constructs a telegram sequence by appending one telegram to a telegram sequence.}

stream: $TS \rightarrow S$
{Constructs from a telegram stream an input stream by appending "****", the empty telegram.}

outtel: $WS \times N \times N \rightarrow OT$
{Constructs an output telegram by composing a word sequence with the integers for short and long words.}

sotseq: $OT \rightarrow OTS$
{Constructs a simple output telegram sequence consisting only of one output telegram.}

outseq: $OTS \times OT \rightarrow OTS$
{Constructs an output telegram sequence by appending one output telegram to an output telegram sequence.}

outstream: $OTS \rightarrow OS$
{Constructs from an output telegram sequence an output telegram stream.}

prog: $S \times OS \rightarrow P$
{Composes an input telegram stream and an output telegram stream to a program.}

length: $W \rightarrow N$
{For a given word the number of characters is delivered.}

succ: $N \rightarrow N$
add: $N \times N \rightarrow N$
eq: $N \times N \rightarrow B$
le: $N \times N \rightarrow B$
{These are the well known successor, addition, equal and less or equal operators.}

*Auxiliary operation symbols*

isword: $W \rightarrow B$
isws: $WS \rightarrow B$
istel: $T \rightarrow B$
ists: $TS \rightarrow B$
iss: $S \rightarrow B$

isotel: $OT \to B$
isots: $OTS \to B$
isos: $OS \to B$
isprog: $P \to B$
{These operators deliver the value true if the corresponding word, word-sequence, telegram, telegram sequence, telegram stream, output telegram, output telegram sequence, output telegram stream or program are well formed; otherwise false is delivered.}
isderts: $TS \times OTS \to B$
{If the output telegram stream is derivable from the telegram stream true is delivered otherwise false}
isderws: $WS \times WS \to B$
{If the first wordsequence is derivable from the second one then true otherwise false is delivered.}
is: $W \times W \to B$
{If the two words are identical the operation delivers true otherwise false.}
isshort: $W \to B$
{If the number of characters of the word is less or equal twelve then true otherwise false is delivered.}

*Axioms:*

{$X_S$ means a variable of sort $S$.}
isprog (stream ($X_{TS}$), outstream ($X_{OTS}$))= iss (stream ($X_S$)$_T$)
& isos (outstream ($X_{OTS}$))
& isderts ($X_{TS}$, $X_{OTS}$)
{A program is well formed if the input telegram stream and the output telegram stream are well formed and the output telegram sequence is derivable from the input telegram sequence.}
iss (stream ($X_{TS}$))= its ($X_{TS}$)
{A telegram stream is well formed if the corresponding telegram sequence is well formed.}
ists (tseq ($X_{TS}$, $X_T$))= ists ($X_{TS}$) & istel ($X_T$)
ists (stseq ($X_T$))= istel ($X_T$)
{A telegram sequence is well formed if it consists of a telegram sequence and a telegram and both are well formed. It is also well formed if it consists of one well formed telegram only.}
istel (tel ($X_{WS}$))= isws ($X_{WS}$)
{A telegram is well formed if the corresponding word sequence is well formed.}
isws (eseq ($X_{WS}$))= isws ($X_{WS}$)
isws (seq ($X_{WS}$, $X_W$))= isws ($X_{WS}$) & isword ($X_W$)
isws (sseq ($X_W$)= isword ($X_W$)
{A word sequence is well formed if it consists of a well formed word sequence followed by a space. It is also well formed if it consists of a word sequence and a word and both are well formed.
If it consists of a word only and this word is well formed the word sequence is well formed too.}

isword $(X_W)$ = le (length $(X_W)$, succ$^{2O}$ (0))

{A word is well formed if its length is less than or equal to twenty.}

length (sword $(X_C)$) = succ (0)

length (word $(X_W, X_C)$ = add (length $(X_W,)$ succ (0))

{The length of the simple word consisting only of one character is one and the length of a word consisting of a word followed by a character is the length of this word plus one.}

isos (outstream $(X_{OTS})$) = isots $(X_{OTS})$

{An output telegram stream is well formed if the corresponding output telegram sequence is well formed.}

isots (sotseq $(X_{OT})$) = isotel $(X_{OT})$

isots (otseq $(X_{OTS}, X_{OT})$) = isots $(X_{OTS})$ & isotel $(X_{OT})$

{An output telegram stream is well formed if it consists of one well formed output telegram only. It is also well formed if it consists of an output telegram sequence and an output telegram and both are well formed.}

isotel (outtel (sseq $(X_W,)$ $X_N$, $Y_N$) = isshort $(X_W)$ & eq ($X$, succ (0))
& isword $(X_W)$ & eq $(Y_N, 0)$

{If the output telegram consists of one short word only then the number of short words is one and the number of long words is zero.}

isotel (outtel (sseq $(X_W)$, $X_N$, $Y_N$) = ($\neg$isshort $(X_W)$) & isword $(X_W)$
& eq$(X_N, 0)$ & eq m $(Y_N$, succ (0))

{If the output telegram consists only of one word, which is a well formed word but not a short word, then the mumber of short words is zero and the number of long words is one.}

isotel (outtel (seq $(X_{WS}, X_W)$, add $(X_N$, succ (0)); $Y_N$) =
isotel (outtel $(X_{WS}, X_N, Y_N)$ & isshort $(X_W)$ & isword $(X_W)$

{If an output telegram consists of a word sequence and a short word and the telegram of the word sequence is a well formed output telegram then the number of short words of the whole telegram is equal to the number of short words of this telegram plus one. The number of long words is the same in both telegrams.

isotel (outtel (seq $(X_{WS}, X_W)$, $X_N$, add $(Y_N$, succ (0))) =
isotel (outtel $(X_{WS}, X_N, Y_N)$ & ($\neg$isshort $(X_W)$)
& isword $(X_W)$

isotel (outtel (eseq $(X_{WS})$, $X_N$, $Y_N$)) = false

{An output telegram cannot contain more than one space between the words.}

isotel (outtel (seq $(X_{WS}, X_W)$, 0, 0)) = false

{An output telegram composed of a word sequence and a word cannot have zero short and zero long words.}

isshort $(X_W)$ = le (length $(X_W)$, succ$^{12}$ (0))

{A word is short if it has less than or equal to twelve characters.}

isderts (stseq (tel $(X_{WS})$), sotseq (outtel $(Y_{WS}, X_N, Y_N)$) =
isderws $(X_{WS}, Y_{WS})$

isderts (tseq $(X_{TS}$, tel $(X_{WS})$), otseq $(X_{OTS}$, outtel $(Y_{WS}, X_N, Y_N)$)) =
isderts $(X_{TS}, X_{OTS})$ & isderws $(X_{WS}, Y_{WS})$

isderts (tseq $(X_{TS}, X_T)$, sotseq $(X_{OT})$) = false

isderts (stseq $(X_T)$, otseq $(X_{OTS}, X_{OT})$) = false

{An output telegram stream is derivable from a telegram stream if both consists of one telegram only and the corresponding word sequences are derivable. It is also derivable if both streams consist of a telegram stream and a word sequence and they are derivable correspondingly. But an output stream is not derivable from an input stream if one of them is a sequence and the other one is a simple sequence.}

isderws (sseq ($X_W$), sseq ($Y_W$))= is ($X_W$, $Y_W$)
isderws (seq ($X_{WS}$, $X_W$), seq ($Y_{WS}$, $Y_W$))= isderws ($X_{WS}$, $Y_{WS}$)
& is ($X_W$, $Y_W$)
isderws (eseq ($X_{WS}$), $Y_{WS}$)= isderws ($X_{WS}$, $Y_{WS}$)
isderws (seq ($X_{WS}$, $X_W$), sseq ($Y_W$)= false
isderws (sseq ($X_W$), seq ($Y_{WS}$, $Y_W$)= false

{An output word sequence is derivable from an input word sequence if both consist of one word only and these words are identical. It is also derivable if both sequences consist of a word sequence and a word and the word sequences are derivable and the words are identical. If an output word sequence is derivable from an input word sequence then the output sequence is also derivable from the input word sequence followed by a space. If one sequence is a simple sequence and the other is a sequence then the output sequence is not derivable from the input sequence.}

Because of readability the axioms for "is" are omitted. They can be built straightforward. "eq", "le", "succ" and "and" are assumed to be standard operations.

### 2.2.3. Logical Specification

Logical specifications are based on predicate calculus. Well known approaches are the axiomatic approach introduced by Hoare [Hoa 69] to define the semantics of programming languages and the logical programming using the programming language PROLOG [Kow 74].

*Hoare's axiomatic approach*

To define the semantics of a programming language Hoare uses specifications. A specification is a string of the form

$$\{A\}p\{B\},$$

where $p$ is a part of a program, and $A$ and $B$ are formulas which can be interpreted as assertions. Therefore the above specification could be read as:

"If $A$ is valid before execution of $p$ and $p$ is finished then $B$ is valid after execution of $p$ (partial correctness)."

To specify a programming language one needs a finite system of specifications consisting of axioms and rules of inference.

In the case of our simple telegram problem we consider a stream of input telegrams as a program. The semantics of this program is defined by the corresponding stream of output telegrams. Such a "program" consists of elements from the set

$$C = \{"A", ..., "Z", "a", ..., "z", "0", ..., "9", "\#", "****"\}$$

(# represents one space,
**** represents the end of a telegram)

and of the (invisible) concatenation operator which concatenates a part of an input stream with an element into a new part of an input stream.

For the definition of the semantics we need some auxiliary variables:

| | |
|---|---|
| output | — Part of the output stream corresponding to the treated part of the input stream, |
| length | — Actual number of characters in the actually treated word, |
| short | — Actual number of short words in the actually treated telegram, |
| long | — Actual number of long words in the actually treated telegram. |
| telno | — Number of already treated telegrams. |

length $= -1$ means the last telegram of the input stream was treated. length $= 0$ means the end of a word was treated. The quintuple (output, length, short, long, telno) represents "evaluation" states of our "programs". That means a component of a given input stream of telegrams transforms a given quadruple into a new one. Particularly, a given input stream of telegrams transforms the quintuple (empty, 0, 0, 0, 0) into a quintuple with the sought output telegram stream as its first component.

In the following system of axioms and rules of inference we omit axioms and rules for formulas supposing that all formulas are well defined.

Axioms:

$$\{A\}c\{B\}$$

$$A = \text{output} = o \,\&\, \text{length} = l \,\&\, l \geqq 0$$

$$B = \text{output} = o.c \,\&\, \text{length} = 1+1$$

$$c \in \{"A", ..., "Z", "a", ..., "z", "0", ..., "9"\}$$

The last telegram has not been treated. Therefore the output stream is concatenated with the new element $c$. ("." means concatenation)

$$\{A1\}\#\{B1\}$$

$$\{A2\}\#\{B2\}$$

$$\{A3\}\#\{B3\}$$

$A1 = 12 < \text{length} \leqq 20 \,\&\, \text{long} = l \,\&\, \text{output} = o$
$B1 = \text{length} = 0 \,\&\, \text{long} = l+1 \,\&\, \text{output} = o.\#$
$A2 = 0 < \text{length} \leqq 12 \,\&\, \text{short} = s \,\&\, \text{output} = o$
$B2 = \text{length} = 0 \,\&\, \text{short} = s+1 \,\&\, \text{output} = o.\#$
$A3 = \text{length} = 0$
$B3 = \text{length} = 0$

These axioms count the number of short and long words in the actually treated telegram. The last axiom secures that only one space occurs between words of the output stream.

$$\{A1\}****\{B1\}$$

$$\{A2\}****\{B2\}$$

$A1$ = (short > 0 or long > 0) & length = 0 & output = $o$
& telno = $t$
$B1$ = short = 0 & long = 0 & telno = $t+1$
& output = $o$. ✻✻✻✻. # .short. # .long. #
$A2$ = short = 0 & long = 0 & length = 0 & output = 0
& telno notequal 0
$B2$ = output = $o$. ✻✻✻✻. # .0. # .0. # & length = $-1$

In the first axiom the actual telegram is finished. For this telegram the number of short and long words is concatenated to the output stream.

In the second axiom the last telegram is treated and output contains the output telegram stream.

Rule of inference:

$$\frac{\{P\}p1\{Q1\},\ \{Q1\}p2\{Q\}}{\{P\}p1\,p2\{Q\}},$$

where $p1$ is a part of an input stream and

$$p2\in\{''A'', \ldots, ''Z'', ''a'', \ldots, ''z'', ''0'', \ldots, ''9'', ''\#'', ''****''\}.$$

This rule enables the composition of specifications and thereby the construction of specifications for input streams of telegrams.

## *Specification of the simple telegram problem using predicate calculus*

For the definition of the simple telegram problem we use now a finite system of Horn clauses (see e.g. [Loy 84]). A Horn clause is a string of the form

$$B \leftarrow A_1, \ldots, A_n, \qquad (*)$$

where $B$, $A_1, \ldots, A_n$ are atoms.

An atom consists of a $n$-ary predicate symbol followed by a list of $n$ terms inserted in paranthesis. Let $x_1, \ldots, x_K$ be the only variables occurring in the terms of $B, A_1, \ldots, A_n$. Then (*) means

$$(Vx_1, \ldots, x_k)(A_1 \& \ldots \& A_n \Rightarrow B).$$

By usual interpretation of formulas we get:
For all values of the variables $x_1, \ldots, x_k$ such that all $\bar{A}_i$ are valid $\bar{B}$ is valid too ($\bar{A}_i$ and $\bar{B}$ are assertions arising from $A_i$ and $B$ respectively).

For the definition of facts one uses a special kind of Horn clauses:

$$B \leftarrow .$$

Horn clauses for the simple telegram problem (For simplicity we omit the definition of some clauses referring to natural numbers and the concatenation operation in terms.):

To achieve better readability of the clauses we will first give an interpretation of the atoms.

| | |
|---|---|
| sum $(X, Y, Z)$ | — $Z$ is the sum of $X$ and $Y$. |
| greater $(X, Y)$ | — $X$ is greater than $Y$. |
| lessequal $(X, Y)$ | — $X$ is less than or equal to $Y$. |
| character $(X)$ | — $X$ is a character. |
| word $(X, L)$ | — $X$ is a word of length $L$. |
| telegram $(X, Y)$ | — $Y$ is the output telegram corresponding to the input telegram $X$. |
| telegramstream $(X, Y)$ [telstr $(X, Y)$] | — $Y$ is the output telegram stream corresponding to the input telegram steream. |

```
character (X)  ←.          X∈{"A", ..., "Z", "a", ..., "z", "0", ..., "9"}
word (X, 1)    ←  character (X).
word (XY, L1)  ←  character (X), word (Y, L), sum (L, 1, L1).
telegram ( #X, Y)←telegram (X, Y).
telegram (X#Y, X#Z#S#L1#) ← word (X, L0),
                             tel (Y, Z#S#L#),
                             greater (L0, 12),
                             lessequal (L0, 20),
                             sum (L, 1, L1).
telegram (X#Y, X#Z#S1#L#) ← word (X, L0),
                             tel (Y, Z#S#L#),
                             lessequal (L0, 12),
                             sum (S, 1, S1).
tel (****, **** #0# 0#)←.
tel (X, Y)←telegram (X, Y).
telegramstr (****, **** #0#0#)←.
telegramstr (XI, YO)     ← telegram (X, Y),
                           telegramstr (I, O).
telegramstream (XI, YO)← telegram (X, Y)
                           telegramstr (I, O).
```

For a given input telegram stream $Ti$ the corresponding output telegram stream is determined (if it exists) beginning from the goal telegramstream $(Ti, Y)$. This is done by constructing a prove for the goal with the Horn clauses. The variables of the Horn clauses are suitably substituted. The determined value of the variable $Y$ in the goal is the sought output telegram stream.

## 2.2.5. Denotational Specification

The objective of denotational specification is to think of programs as being functions which transform input values into output values. However unlike a program which specifies how to compute the function, the denotational specifications merely indicate which function the program should compute.

The fundamentals of this theory were developed by Scott and Strachey [Sco 71] to define the semantics of programming languages. They were further developed by Stoy [Sto 77].

To specify our simple telegram problem we will use the meta-language of the Vienna Development Method. (see e.g. [Bjø 78])

Analogously to the axiomatic approach the starting point is to consider an input telegram stream as a program. The semantic meaning of this program is the corresponding output telegram stream. Now, the denotational approach requires the definition of syntactical and semantical domains and the definition of functions determining the semantic meaning of program parts. Furthermore, we need some functors for "pasting together" semantic functions.

*Denotational Specification of the Simple Telegram Problem Syntactic Domains:*

Program = Prog.
Prog::Stream Endword.
Stream = Tel | Telseq.
Tel::Wordseq Endword.
Telseq::Stream Tel.
Wordseq = Word | Words.
Word::Charstr Spaces.
Words::Wordseq Word.
Charstr = Character | Characters.
Character = *A*|...|*Z*|*a*|...|*z*|0|...|9.
Characters::Charstr Character.
Spaces = Space|Spaceseq.
Space = #.
Spaceseq::Spaceseq Space.
Endword = ✳✳✳✳.

Sematic Domains:
OUTTELSTREAM::OUTTEL*
OUTTEL::WORD* END SPACE INT SPACE INT SPACE
WORD::CHARACTER* SPACE
CHARACTER = {*A*, ..., *Z*, *a*, ..., *z*, 0, ..., 9}
INT = {0, 1, 2, 3, ...}
SPACE = #
END = ✳✳✳✳✳

Elaboration Functions:

| | | | |
|---|---|---|---|
| **type:** | eval-program: | Program | → OUTTELSTREAM |
| **type:** | eval-orog: | Prog | → OUTTELSTREAM |
| **type:** | eval-stream: | Stream | → OUTTELSTREAM |
| **type:** | eval-tel: | Tel | → OUTTEL |
| **type:** | eval-wordseq: | Wordseq | → OUTTEL |
| **type:** | eval-word: | Word | → INT×INT |
| **type:** | eval-charstream: | Charstream | → INT |
| **type:** | eval-characters: | Characters | → INT |

eval-program $(p) \overline{\wedge}$ eval-prog $(p)$

{The result of the function eval-program $(p)$ is the result of the function eval-prog $(p)$.}

```
eval-prog (mk-prog (s, e)) ≜
                              let v1 = eval-stream (s)
                                  v2 = # * * * * #0#0#
                              in  v1.v2
```

{A program $p$ consists of a stream $s$ and an end word $e$. The result of eval-prog ($p$) is equivalent to the result of eval-stream ($s$) concatenated by the empty telegram.}

```
eval-stream (st) ≜
    cases st: mk-tel (ws, e)    → eval-wordseq (ws),
              mk-telseq (s, t)  → let v1 = eval-stream (s),
                                      v2 = eval-tel (t)
                                  in  v1.v2
```

{If the stream $st$ consists of a word sequence $ws$ and the end word $e$ the result of eval-stream ($st$) is the result of eval-wordseq ($ws$).
But if $st$ consists of a stream $s$ and a telegram $t$ eval-stream ($st$) is equal to the concatenation of the results of eval-stream ($s$) and eval-tel ($t$).}

```
eval-tel (mk-tel (ws, e)) ≜ eval-wordseq (ws)
```

{A telegram consists of a word sequence $ws$ and an end word $e$. The result of eval-tel ($t$) is equal to the result of eval-wordseq ($ws$).}

```
eval-wordseq (ws) ≜
    cases ws: mk-word (w)        → let (x, y) = eval-word (w)
                                     in w * * * * #x#y#
              mk-words (sw, w)   → let
                                     (u, v) = eval-word (w),
                                   x * * * * #y#z# = eval-wordseq (sw)
                                   in xw * * * * #u+y#v+z#
```

{If the word sequence $ws$ consists of one word $w$ only the result of eval-wordseq ($ws$) is the composition of $w$ and the result of eval-word ($w$).
If $ws$ consists of a wordsequnce $sw$ and a word $w$ the result of eval-wordseq ($ws$) is a composition of the results of eval-word ($w$) and eval-wordseq ($sw$).}

```
eval-word (mk-charstr (cs, s)) ≜
                              if eval-charstr (cs) ≦ 12
                                then (1, 0)
                                else
                                if eval-charstr (cs) ≦ 20
                                  then (0, 1)
                                  else undefined
```

{A word consists of a character stream $cs$ and a space sequence $s$. The result of eval-word ($w$) is equal to (1, 0) or (0, 1) depending on the result of eval-chararstr ($cs$).}

eval-charstr $(cs) \underline{\Delta}$

> **cases** $cs$: mk-character $(c) \rightarrow 1$,
> mk-characters $(c) \rightarrow$ eval-characters $(c)$
>
> {If the character stream $cs$ consists of a character $c$ the result of eval-charstr $(cs)$ is equal to one.
> If $cs$ consists of characters $c$ then the result of eval-charstr $(cs)$ is equal to the result of eval-characters $(c)$.}

eval-characters (mk-characters $(cs, c)) \underline{\Delta}$

```
                    let $x$=eval-charstr ($cs$)
                      in $x+1$
```

{Characters $ca$ consist of a character stream $cs$ and a character $c$. The result of eval-characters $(ca)$ is equal to the result of eval-charstr $(cs)$ plus one.}

### 2.2.6. Relations Between Fundamentals of Problem, Program and Implementation Specification

A selection of some references related to the topic is given in Figure 2.

According to the methods for problem and program specification there are many tools for implementation specification, which are also influenced by mathematics. Figure 2 also presents a classification of some of these tools.

There is no fixed way from program specification to programming. The software engineer can choose all programming methods for every specification.

But modern methodologies of software engineering try to unify the descriptions during the whole life cycle. Especially the implementation specification attains a higher level of abstraction and has the form of program or problem specification.

There is a good success in logical programming (e.g. PROLOG), in programming by grammars (e.g. CDL, HFP) and in programming on a very high level (e.g. MODEL).

All modern methods have in common, that the software engineer is not confronted with all implementation details. In many cases he does not know them.

He can concentrate upon the main design principles. The details are generated automatically (artificial intelligence) or are already implemented (abstract data types).

In our opinion programming by grammar will be more important in future. Watt and Madsen [Wat 81] have shown for example, that algebraic, logical and denotational specification can be expressed by extended attribute grammars.

### 2.2.7. Grammatical Specification of the simplified telegram problem

First, we will informally introduce the notion of attribute grammars on the basis of grammars of syntactic functions [Rie 83].
An attribute grammar is a contextfree grammar

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

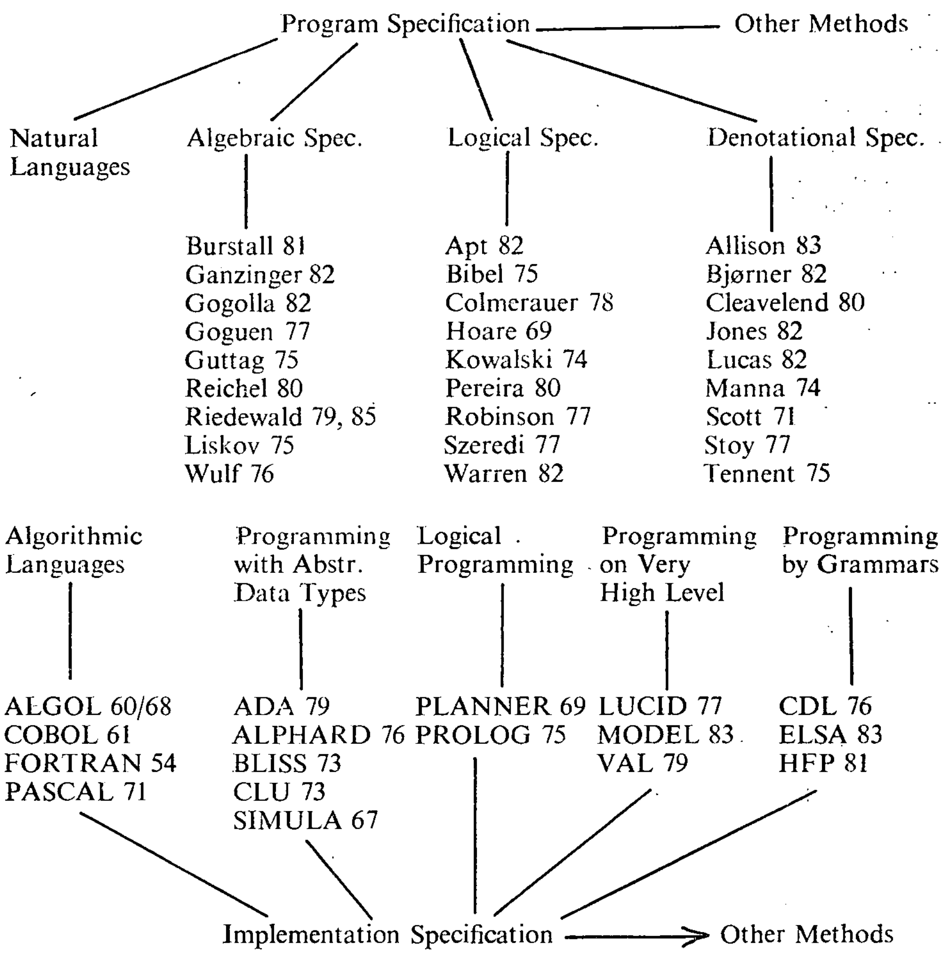


*Figure 2.* Classification of Problem, Program and Implementation Specification Methods and Tools

$V$-vocabulary, $T$-set of terminals, $N$-set of nonterminals
$V = N \cup T$, $S$-start element, $P$-set of production rules
augmented with parameters, auxiliary syntactic functions and semantic functions. Auxiliary syntactic functions are necessary to describe the static semantic.

The rules have the form

$$f(p_1^f, \ldots, p_{n_f}^f) ::= f_1(p_1^{f_1}, \ldots, p_{n_{f_1}}^{f_1}) \ldots f_r(p_1^{f_r}, \ldots, p_{n_{f_r}}^{f_r})$$

$$H_1(p_1^{H_1}, \ldots, p_{n_{H_1}}^{H_1}) \ldots H_k(p_1^{H_k}, \ldots, p_{n_{H_k}}^{H_k}).$$

$$f \in N, f_1, \ldots, f_r \in V,$$

$H_1, \ldots, H_k \in$ {auxiliary syntactic functions} $\cup$ {semantic functions}, $p_y^x \in$ set of parameters, $x, y, r, k \in$ set of integers.

The telegram problem can be specified in different ways by a grammar. The following two methods are possible:

1. The input and output telegram streams are described by parameters. The start symbol has the input telegram stream as input parameter and delivers the output telegram stream as the value of the output parameter.
2. The input telegram stream is described by a context-free grammar and this grammar is augmented by parameters and functions in such a way that the start element of the grammar delivers the output telegram stream as the value of the parameter of of the start symbol.

The first method would result in a grammar very similar to our specification using Horn clauses. Therefore we will omit this example here. The interested reader will very easily get such a grammar.

Let us demonstrate the second method in full detail.

## *Grammatical Specification of the Simple Telegram Problem (using the second method)*

### I. *Semantic functions*

— CAT2 (↓*S*1, ↓*S*2, ↑*S*)
This function concatenates *S*2 to *S*1 and delivers *S*.
— CAT3 (↓*S*1, ↓*S*2, ↓*S*3, ↑*S*)
This function delivers *S*1. ✳✳✳✳. # .*S*2. #*S*3. # in *S*.
— COUNT (↓*L*, ↓*Long*1, ↓*Short*1, ↑*Long*, ↑*Short*)

```
IF L<12 THEN Long:=Long1;  Short:=Short1+1
             ELSE Long:=Long1+1; Short:=Short1
FI
```

— ADD (↓*A*, ↓*B*, ↑*C*)
$C := A + B$

### II. *Auxiliary Syntactic functions*

— OVERLENGTH (↓*L*)
The actions of the parser are influenced by this function. The application of the corresponding rule is possible if *L* is less or equal to 20 only.

### III. *Production rules*

```
1. Program (↑Outstream)::=Telegramstream (↑Out)
                "#" Endsymbol
                CAT3 (↓Out,↓ "0", ↓"0", ↑Outstream).
2. Telegramstream (↑O)::=Telegram (↑O).
```

```
 3. Telegramstream (↑O)::=Telegramstream (↑O1)
                          Telegram (↑O2)
                          CAT2 (↓O1, ↓O2, ↑O).
 4. Telegram (↑O)::=  Wordsequence (↑O1, ↑Short, ↑Long)
                      Endsymbol
                      CAT3 (↓O1, ↓Short, ↓Long, ↑O).
 5. Wordsequence (↑O, ↑Short, ↑Long)::=
             Word (↑O, ↑Length)
             OVERLENGTH (↓Length)
             COUNT (↓Length, ↓"0", ↓"0", ↑Short, ↑Long).
 6. Wordsequence (↑O, ↑Short, ↑Long)::=
             Wordsequence (↑O1, ↑Short1, ↑Long1)
             Word (↑O2, ↑Length)
             OVERLENGTH (↓Length)
             COUNT (↓Length, ↓Short1, ↓Long1,
                                   ↑Short,  ↑Long
             CAT2 (↓O1, ↓O2, ↑O).
 7. Word (↑Word, ↑L)::=  Charactersequence (↑Word, ↑L)
                         Spacesequence.
 8. Charactersequence (↑C, ↑"1")::=Character (↑C).
 9. Charactersequence (↑Char, ↑Length)::=
             Charactersequence (↑Char1, ↑Length1)
             Character (↑Char 2)
             CAT2 (↓Char1, ↓Char2, ↑Char)
             ADD (↓Length1, ↓"1", ↑Length).
10. Character (↑"A")::="A".
     ...
35. Character (↑"Z")::="Z".
36. Character (↑"a")::="a".
     ...
51. Character (↑"z")::="z".
52. Character (↑"0")::="0".
     ...
62. Character (↑"9")::="9".
63 Endsymbol::="*****"..
64. Spacesequence::="#".
65. Spacesequence::=Spacesequence "#".
```

### 2.3. Programming with Production Rules

The relations of methods, which were developed independently for using production rules in programming, are the result of current research.

Figure 3 shows some interesting relations between attribute grammars and logical programming.

The world wide interest in logical programming and the relations of Figure 3 support our opinion to study applications of attribute grammars in software engineering.

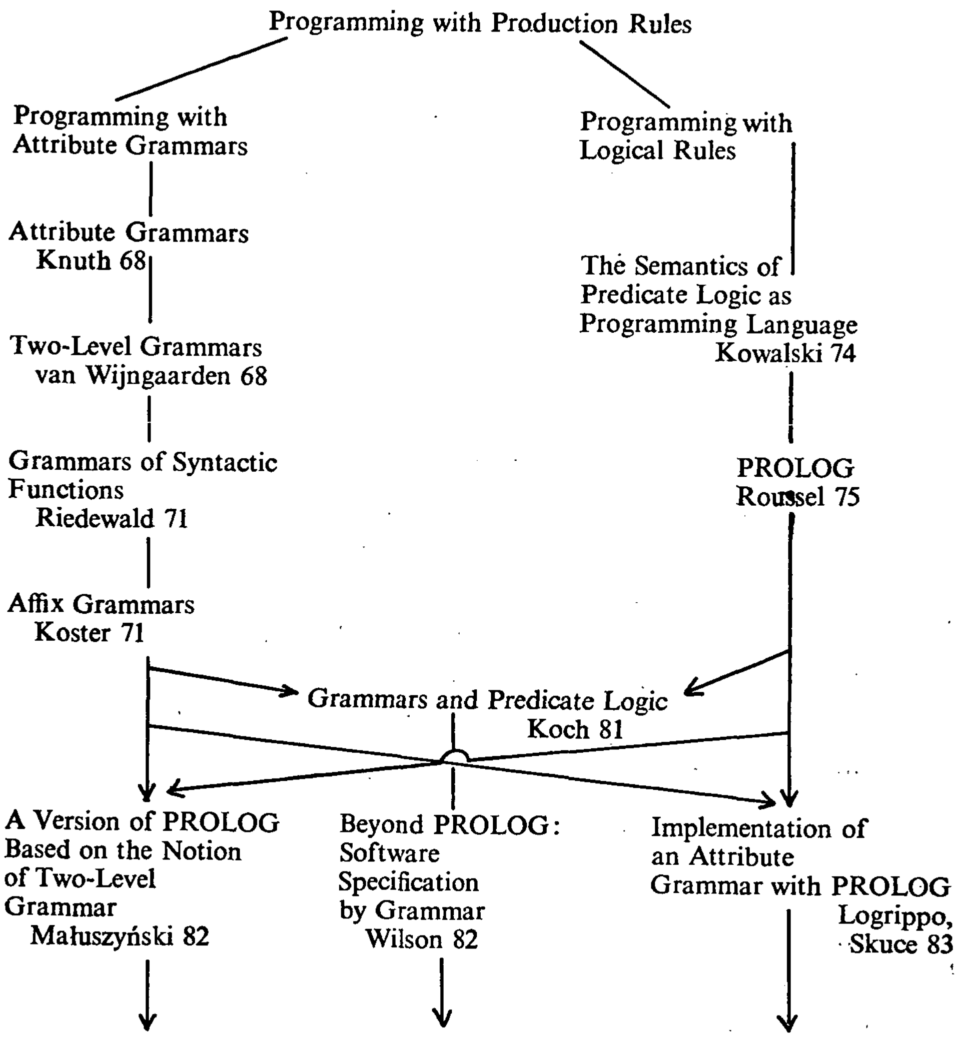

Programming with Production Rules
Programming with Attribute Grammars
Programming with Logical Rules
Attribute Grammars Knuth 68
The Semantics of Predicate Logic as Programming Language Kowalski 74
Two-Level Grammars van Wijngaarden 68
Grammars of Syntactic Functions Riedewald 71
PROLOG Roussel 75
Affix Grammars Koster 71
Grammars and Predicate Logic Koch 81
A Version of PROLOG Based on the Notion of Two-Level Grammar Małuszyński 82
Beyond PROLOG: Software Specification by Grammar Wilson 82
Implementation of an Attribute Grammar with PROLOG Logrippo, Skuce 83

*Figure 3.* Some Relations Between Programming Methods Using Production Rules

## 2.4. Some Software Development Methods and Tools with Different Use of Data and Information

We want to discuss some other arguments supporting the application of attribute grammars in software development. Let us first have a look at some methods and tools supporting the use of data and information in different kinds. Figure 4 is an attempt to classify some methods and tools. Such tools can also be used in another way, of course, but they were mainly designed for this purpose.

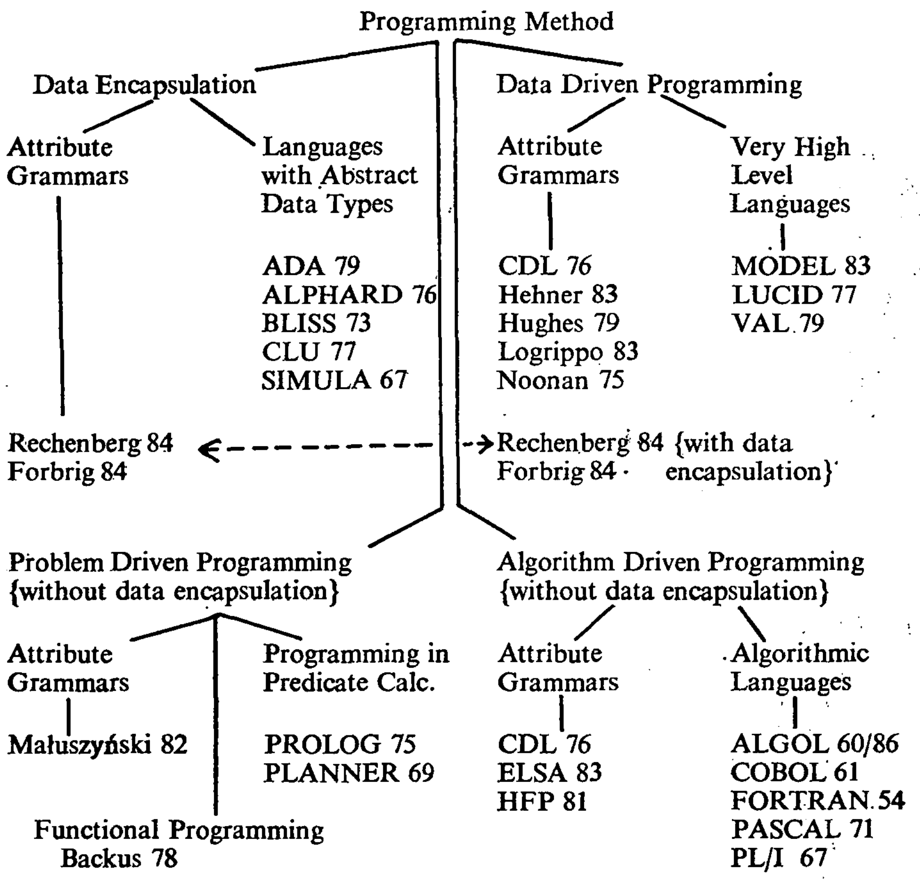


*Figure 4.* Classification of some Programming Methods and Tools

The following results can be obtained from Figure 4:

1. Attribute grammars are applied in all classified fields.
2. The programming language CDL can be used in a data driven and algorithm driven way.
3. Attribute grammars can be combined with data encapsulation.
4. Attribute grammars can be used to combine data driven programming and data encapsulation.

The method of data driven programming by attribute grammars with abstract data types was discussed in Rostock in [For 84 a]. Some similar results can be found in [Rec 84]. Rechenberg proposes attribute grammars mainly as a tool for program specification. The implementation is suggested by top-down programming.

We will mainly use attribute grammars as input for a translator writing system. Examples of data driven data processing using attribute grammars with abstract data types will be presented in the following section.

## 2.5. Data Driven Programming with Abstract Data Types

### 2.5.1. Attribute Grammars and Abstract Data Types

We will extend the definition of an attribute grammar by abstract data types. The context-free grammar is not only augemented with parameters, semantic functions and syntactic functions but also with functions of abstract data types.

That means

$H_1, \ldots, H_k \in$ {semantic functions} $\cup$ {syntactic functions} $\cup$ {functions of abstract data types}.

These grammars are more effective to implement, because not all information has to be transfered a long way via parameters.

In our opinion attribute grammars with abstract data types are better to read and write. They can better be maintained.

### 2.5.2. Examples

#### 2.5.2.1. Grammatical Specification of the Simple Telegram Problem

I. *Abstract data types*

∗ File of output telegram stream with statistics

— OPEN-OUTFILE, CLOSE-OUTFILE

These functions open and close the file.

— OUTWORD (↓*Word*, ↓*Length*)

The "*Word*" with given length is stored on the file.

— OUTCOUNT (↓*Short*, ↓*Long*)

The number of long and short records of the current telegram are stored on the file according to the specification. (∗∗∗∗. # .*Short*. # *Long*. #)

II. *Semantic functions*

We use the semantic functions of our example in section 2.2.7.

III. *Aixiliary syntactic functions*

We use the syntactic functions of the example in section 2.2.7.

IV. *Production rules*

```
1. Program::=Begin  Stream  CLOSE-OUTFILE.
2. Begin::=OPEN-OUTFILE.
3. Stream::=Telegramstream. "#" Endsymbol
                 OUTCOUNT(↓"0", ↓"0").
4. Telegramstream::=Telegram.
5. Telegramstream::=Telegramstream  Telegram.
```

```
 6. Telegram::=Wordsequence (↑Short, ↑Long)
               Endsymbol
               OUTCOUNT (↓Short, ↓Long).
 7. Wordsequence (↑Short, ↑Long)::=Word (↑Word, ↑L)
               OVERLENGTH (↓L)
               COUNT (↓L, ↓"0", ↓"0", ↑Short, ↑Long)
               OUTWORD (↓Word, ↓L).
 8.Wordsequence (↑Short, ↑Long)::=
    Wordsequence (↑Short1, ↑Long1)
               Word (↑Word, ↑L)
               OVERLENGTH (↓L)
               COUNT (↓L, ↓Short1, ↓Long1, ↑Short, ↑Long)
               OUTWORD (↓Word, ↓L).
 9. Word (↑Word, ↑L)::=Charactersequence (↑Word, ↑L)
               Spacesequence.
10. Charactersequence (↑C, ↑"1")::=Character (↑C).
11. Charactersequence (↑C, ↑L)::=
                              Charactersequence (↑C1, ↑L1)
                              Character (↑C2)
                              CAT2 (↓C1, ↓C2, ↑C)
                              ADD (↓L1, ↓"1", ↑L).
12. Character (↑"A")::="A".
    ...
37. Character (↑"Z")::="Z".
38. Character (↑"a")::="a".
    ...
53. Character (↑"z")::="z".
54. Character (↑"0") :="0".
    ...
64. Character (↑"9")::="9".
65. Endsymbol::="****".
66. Spacesequence ::=" # ".
67. Spacesequence ::=Spacesequence " # ".
```

If standard technices are used as subprograms for lexical analysis only the first eight production rules of the grammar are necessary.

### 2.5.2.2. Grammatical Specification of a Very Little Commercial Project

The following task has to be fulfilled by a computer: A special master file contains data of all wage-earners of an enterprise. Another file contains monthly data of working time and wages. These two files have to be used to produce pay slips, to remit the money through the bank and to report about working time. There is a lot of possibilities of monthly data. Therefore, every item has a key and the file contains only items different from zero. According to the four kinds of taxes used in the GDR the total sum on the pay slip is broken up into four groups. This very little commercial project can be descibed by the following attribute grammar.

I. *Abstract data types*

a) Master file with the functions:
- OPEN-MASTER-FILE, CLOSE-MASTER-FILE
  These functions open and close the file.
- MASTER-DATA ($\downarrow$*Number*, $\uparrow$*Group*, $\uparrow$*Place*, $\uparrow$*Bank*)
  This function delivers for a given number of a wage-earner his number of the bank account, his working place and his group of professional classification. The master file data of this worker are prepared open for other functions.
- MASTER-WAGES ($\uparrow$*Money*)
  For the current wage-earner the money per hour is delivered from the master file.

b) File of working time statistics:
- OPEN-TIME, CLOSE-TIME
- TIME-BEGIN ($\downarrow$*Group*, $\downarrow$*Place*)
  For a given group and working place the entry of statistical data is prepared.
- TIME-KEY ($\downarrow$*Key*, $\downarrow$*Hours*)
  For a group and working place fixed above the given hours are added according to the key.

c) File of data for the bank:
- OPEN-BANK, CLOSE-BANK
- BANK-REMIT ($\downarrow$*Bank*, $\downarrow$*Amount*)
  The amount is transfered to the given bank account.

d) File of pay slips:
- OPEN-PAY-SLIP, CLOSE-PAY-SLIP
- PAY-SLIP-BEGIN ($\downarrow$*Number*, $\downarrow$*Place*)
  An entry of data is prepared for number and place.
- PAY-SLIP ($\downarrow$*Am*1, $\downarrow$*Am*2, $\downarrow$*Am*3, $\downarrow$*Am*4, $\downarrow$*Sum*)
  The given data are entered on the file.

II. *Semantic functions*

- ADD2 ($\downarrow S1$, $\downarrow S2$, $\uparrow Sum$)
  $Sum := S1 + S2$
- ADD ($\downarrow S1$, $\downarrow S2$, $\downarrow S3$, $\downarrow S4$, $\uparrow Sum$)
  $Sum := S1 + S2 + S3 + S4$
- MULT ($\downarrow F1$, $\downarrow F2$, $\uparrow Product$)
  $Product := F1 * F2$
- DEC ($\downarrow D1$, $\downarrow D2$, $\downarrow D3$, $\downarrow D4$, $\downarrow D5$, $\uparrow Value$)
  $Value := (((D1 * 10 + D2) * 10 + D3) * 10 + D4) * 10 + D5$

III. *Auxiliary syntactic functions*

Auxiliary syntactic functions are not necessary for our example.

IV. *Production rules*

1. Project-run ::= Begin Records CLOSE-MASTER-FILE
CLOSE-TIME CLOSE-BANK CLOSE-PAY-SLIP.
2. Begin ::= OPEN-MASTER-FILE OPEN-TIME OPEN-BANK
OPEN-PAY-SLIP.
3. Records ::= Record.
4. Records ::= Records Record.
5. Record ::= Head (↑*Bank*) Items (↑*Amount*)
BANK-REMIT (↓*Bank*, ↓*Amount*).
6. Head (↑*Bank*) ::= "NO" Earn-No (↑*Number*)
MASTER-DATA (↓*Number*, ↑*Group*, ↑*Place*, ↑*Bank*)
TIME-BEGIN (↓*Group*, ↓*Place*)
PAY-SLIP-BEGIN (↓*Number*, ↓*Place*).
7. Items (↑*Amount*) ::= Amount1 (↑*Am*1) Amount2 (↑*Am*2)
Amount3 (↑*Am*3) Amount4 (↑*Am*4)
ADD4(↓*Am*1, ↓*Am*2, ↓*Am*3, ↓*Am*4, ↑*Sum*)
PAY-SLIP (↓*Am*1, ↓*Am*2, ↓*Am*3, ↓*Am*4,
↓*Sum*).
8. Amount1 (↑*Am*1) ::= Amounts1 (↑*Am*1).
9. Amount1 (↑"0") ::= .
10. Amounts1 (↑*Am*1) ::= *Am*1 (↑*Am*1).
11. Amounts1 (↑*Am*1) ::= Amounts1 (↑*Am*2) *Am*1 (↑*Am*3)
ADD2 (↓*Am*2, ↓*Am*3, ↑*Am*1).
12. Am1 (↑*Am*1) ::= "H01" Hours (↑*H*) "M01" Money (↑*M*)
MULT (↓*H*, ↓*M*, ↑*Am*1) TIME-KEY (↓"01", ↓*H*).
13. Am1 (↑*Am*1) ::= "H01" Hours (↑*H*) MASTER-WAGES (↑*M*)
MULT (↓*H*, ↓*M*, ↑*Am*1) TIME-KEY (↓"01", ↓*H*).
14. Am1 (↑*Am*1) ::= "FM01" Money (↑*Am*1).
15. Am1 (↑*Am*1) ::= "FM02" Money (↑*Am*1).
16. Earn-No (↑*Number*) ::= Number5 (↑*Number*).
17. Hours (↑*H*) ::= Number5 (↑*H*).
18. Money (↑*M*) ::= Number5 (↑*M*).
19. Number5 (↑*V*) ::= Digit (↓*D*1) Digit (↓*D*2) Digit (↓*D*3)
Digit (↓*D*4) Digit (↓*D*5)
DEC (↓*D*1, ↓*D*2, ↓*D*3, ↓*D*4, ↓*D*5, ↑*V*).
20. Digit (↑"0") ::= "0".

...

30. Digit (↑"9")= ::"9".

With respect to simplicity the rules of Amount2, Amount3 and Amount4 were omitted. They can be formulated similarly to the rules of Amount1.

According to rule 1 and 2 the project run consists of opening all abstract data types, interpreting a sequence of records and closing all abstract data types.

Every record has head data and items (rule 5).

The head data consist of key "NO" followed by the number of a wage-earner (rule 6). With the help of this number, data are obtained from the master file and the entry of data for statistics and the pay slip are prepared.

The items consist of four groups (rule 7). Every group can be a sequence of data (rule 11). The empty sequence is possible (rule 9).

If there are data about hours and money, multiplication is performed and the hours are reported for statistics (rule 11).

If there are only hours the money per hour is taken from the master file (rule 13). It is also possible to get money per month (rule 14, 15).

Everybody familiar with attribute grammars can easily get this information from the grammar. Therefore, it is an exact document of the project and it supports the implementation.

## 3. Summary

After a short survey of the fundamentals of software engineering we have discussed some classifications of methods and tools. As a result, the combination of data driven programming and data encapsulation, usually classified as contrary concepts, was developed by using attribute grammars.

This method was demonstrated by a very little commercial data processing system. The advantages of the method presented can be summarized as follows.

1. Attribute grammars are a good document for design and implementation.
2. Modularization is supported.
3. Maintenance can be performed relative easily and locally.
4. Syntactic analysis of data is automated and the software engineer can concentrate upon the main principles of his system.
5. Grammars can already be tested at very early development phases and the completeness of the system can be checked.
6. Simulations can be performed without total implementation of all functions.
7. Developed projects are broken up into many parts in a natural manner, which can run in parallel.
8. Functions have not to be designed in the same manner. A system of existing modules can be composed by using this method.
9. The method supports the use and design of so called knowledge bases (e.g. as abstract data types).
10. Syntactic analysis algorithm in translator writing systems will be much more effective than most hand written algorithms.
11. Automatic error recovery methods can be used (e.g. [For 84 b]).

Of course, this method is not intended to be applied to all problems of software engineering. The application of data driven programming, however, is very well supported by a grammar. We think this method to be useful especially in the field of commercial data processing.

Only a short list of references can be given here. A more complete list with about 300 references related to the topic of software specification can be obtained from the authors.

WILHEM PICK UNIVERSITAET ROSTOCK
SEKTION INFORMATIK
DDR 2500 ROSTOCK
ALBERT EINSTEIN STRASSE 21

## References

[Aba 82] Abaffy, J., Krafft, W., XHELF: An Aid for Developing Computer Programs, Proc. of the Conference on System Theoretical Aspects in Computer Science, Hungary, 1982.

[Bac 78] Backus, J., Can Programming be Liberated from the von Neumann Style? Comm. of the ACM 21 (8) 1978.

[Bjø 78] Bjørner, D., Jones, C. B., The Vienna Development Method: The Meta-Language, Lecture Notes in Computer Science, Vol. 61, 1978.

[Cle 80] Cleaveland, J. C., Mathematical Specification, SIGPLAN Notices, 15 (12) 1980.

[Clo 81] Clocksin, W. F., Mellish, C. S., Programming in PROLOG, Springer Verlag, 1981.

[Col 81] Coleman, D., Hughes, W., Powell, M. S., A Method for the Syntax Directed Design of Multiprograms, IEEE Trans. on Software Eng., 7 (2) 1981.

[Des 83] Despeyroux, J., An Algebraic Specification of a PASCAL Complier, SIGPLAN Notices, 18 (12) 1983.

[Fib 84] Fiby, R., Molnar, S., Weigl, I., Is the Idealised Logic Programming Feasible? Proc. IMYCS 84, Smolenice, CSSR, 1984.

[For 84a] Forbrig, P., Attributierte Grammatiken und Softwarespezifikation, Seminar attr. Gr., Rostock 84.

[For 84b] Forbrig, P., A New Error Recovery Method for Optimized LR Parsers, Proc. IMYCS 84, Smolenice.

[For 85] Forbrig, P., Kombination der datengesteuerten Programmierung nach Jackson mit der Methode der Datenabstraktion nach Parnas, Rostock, Rep. 7/85.

[Gog 74] Goguen, J. A., Thatcher, J. W., Initial Algebra Semantics, IEEE Symp. on Switching and Autom. 74.

[Gut 75] Guttag, J. V., The Specifications and Application to Programming of Abstract Data Types, University of Toronto, Report CSRG-59, 1975.

[Gut 78] Guttag, J. V., Horning, J. J., The Algebraic Specification of Abstract Data Types, Acta Informatica 10 (1) 1978.

[Gyi 83] Gyimóthy, T., Simon, E., Makay, A., An Implementation of HLP, Acta Cybernetica, 3 (6) 1983.

[Heh 83] Hehner, E. C. R., Silverberg, B. A., Programming with Grammars: An Exercise in Methodology-Directed language Design The Computer J. 26 (3) 1983.

[Hug 79] Hughes, J. W., A Formalization and Explication of the Michael Jackson Method of Program Design, Software Practice & Experience, 9 (2) 1979.

[Hoa 69] Hoare, C. A. R., An Axiomatic Basis for Computer Programming, Comm. of the ACM, 12 (10) 1969.

[Jac 75] Jackson, M., Principles of Program Design, Academic Press, 1975.

[Kat 81] Katayama, T., HFP: A Hierarchical and Functional Programming Based on Attribute Grammars, Proc. 5th Int. Conf. on Software Engineering, 1981.

[Knu 82] Knuth, E., Neuhold, E. J., Specification and Design of Software Systems, Proc. of the Conference of Operating Systems, Hungary, 1982.

[Kow 74] Kowalski, R. A., Predicate Logic as a Programming Language, Information Processing 74, North Holland, 1974.

[Lae 84] Laemmel, U., Specification of Dialogue Systems Using Attributed Grammars, Proc. IMYCS 84.

[Lin 83] Lindsey, C. H., ELSA — An Extensible Programming System, IFIP — TC 2, Dresden, 1983.

[Log 83] Logrippo, L., Skuce, D. R., File Structures, Program Structures and Attribute Grammars, IEEE Trans. on Software Engineering, 9 (3) 1983.

[Loy 84] Loyd, J. W., Foundations of Logic Programming, Springer Verlag, Heidelberg—New York—Tokio 1984.

[Mad 80] Madsen, O. L., On Defining Semantics by Means of Extended Attribute Grammars, Lecture Notes in Computer Science, Vol. 94, 1980, p. 259—300.

[Mal 80] Mallgren, W. R., Formal Specification of Graphical Data Types, University of Washington, Technical Reprt No. 80—04—04, 1980.

[Mał 82] Małuszynski, J., Nilsson, J. F. A Version of PROLOG Based on the Notion of Two-Level Grammars, International PROLOG Workshop, Linkoeping, 1982.

[Noo 75] Noonan, R. E., Structured Programming and Formal Specification, IEEE Trans. on S. Eng., 1 (4) 1975.

[Par 72] PARNAS, D. L., On Criteria to be Used in Decomposing Systems into Modules, Comm. ACM, 15 (12) 1972.

[Rec 84] RECHENBERG, P., Attributierte Grammatiken als Methode der Softwaretechnik, El. Rechenanl., 26 (3) 84.

[Rie 83] RIEDEWALD, G., MALUSZYNSKI, J., DEMBINSKI, P., Formale Beschreibung von Programmiersprachen, Akademie Verlag Berlin, 1983. also Oldenbourg Verlag, Wien—Muenchen, 1983.

[Rie 85] RIEDEWALD, G., Ein Modell fuer Programmiersprachen und Compiler auf der Basis universeller Algebren, Elektronischen Informationsverarbeitung und Kybernetik, 21 (3) 1985.

[Sco 71] SCOTT, D., STRACHEY, C., Towards a Mathematical Semantics for Computer Languages, Proc. of the Symp. on Computer and Automata 1971.

[Sto 77] STOY, J. E., Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory, MIT Press 1977.

[Sze 77] SZEREDI, P., PROLOG: A Very High Level Language Based on Predicate Logic, Proc. 2nd Hungarian Computer Science Conference, 1977.

[Wat 79] WATT, D. A., MADSEN, O. L., Extended Attribute Grammars, Aarhus Univ., Rep. DAIMI PB-105, 1979.

[Watt 83] WATT, D. A., MADSEN, O. L., Extended Attribute Grammars, The Computer Journal, 26 (2) 1983.

[Wil 82] WILSON, W. W., Beyond PROLOG: Software Specification by Grammars, SIGPLAN Notices, 17 (9) 1982.

*(Received Dec. 18, 1985)*

# Bibliographie

**Barbara Liskov, John Guttag: Abstraction and Specification in Program Development.** XVI+469 pages, The MIT Press, McGraw-Hill Book Company, 1986.

Abstraction is a key concept in modern programming methodology. This book provides the first unified treatment of techniques of abstraction and specification in program development. Three kinds of abstractions, procedural, iteration and data abstraction are discussed in the book. The authors place particular emphasis on the use of data abstraction to produce highly modular programs. Various aspects of data abstraction used in program construction are dealt with: requirement analysis, design, specification, verification and implementation. One chapter is devoted to present a language for writing formal specifications. We believe that all students of programming should acquire at least a reading knowledge of formal specifications and that such knowledge can be a great help in writing informal specifications.

There are many examples of abstractions throughout the text. Most sample implementations in the book are written in CLU, a programming language which was designed to support the methodology of this book. Sufficient material is included, however, to allow the reader to work in Pascal as well. Complete reference manual of the CLU language is included in the appendix.

This very clearly written book can be recommended as a text for both an undergraduete laboratory course and a graduete-level course for professional programers and analysts.

Gy. Horváth

**F. Gécseg: Products of Automata (EATCS Monographs on Theoretical Computer Science, vol. 7),** VIII+107 pages, Springer-Verlag, Berlin—Heidelberg—New York—Tokyo, 1986.

Both theoretical and practical considerations motivate the representation of objects as certain compositions of simpler ones. In the theory of automata this observation led to the concepts of products and complete systems of automata. Although numerous concepts of products are known, there are only a few results as regards them. One of the exceptions is the hierarchy of $\alpha_i$-products introduced by the author 10 years ago. This hierarchy contains a product for any non-negative integer $i$. In an $\alpha_i$-product the index set of the component automata is linearly ordered and the input of each automaton may only depend ont he states of those automata preceding it and on the states of the next $i-1$ automata, including itself The monograph consisting of five chapters deals with $\alpha_i$-products and gives a systematic summary of results concerning this hierarchy.

The first chapter contains the necessary concepts and results from the theories of automata and universal algebras.

Chapter 2 deals with homomorphic representations of automata with respect to the $\alpha_i$-products. After studying certain special questions concerning homomorphic representations by $\alpha_0$-and $\alpha_1$-product, a deep characterization of homomorphically complete systems is given with respect to the $\alpha_i$-products, where $i\geqq 2$. Using this characterization, it is shown that from $i=2$ the $\alpha_i$-product is homomorphically equivalent to the general product, while for $i=0, 1, 2$ the hierarchy is proper. Afterwards, those automata are determined which are simple in the sense that whenever they can be represented homomorphically by an $\alpha_i$-product then a single-factor $\alpha_i$-power of one of its components represents them homomorphically. This chapter ends with a decision problem.

Chapter 3 is devoted to isomorphic representations. It contains a description of isomorphically complete systems with respect to the $\alpha_i$-products. It is shown that as regards isomorphic representation the $\alpha_i$-products form a proper hierarchy, and from $i=1$ they are equivalent to each other with respect to isomorphic completeness.

Chapter 4 concerns with isomorphic and homomorphic simulations with respect to a generalization of $\alpha_i$-products. Comleteness results are proved as regards both isomorphic and homomorphic simulations. The chapter ends with some camparison results.

Chapter 5 deals with infinite products and representations of automaton mappings in finite lengths. It is shown that in this representation the $\alpha_0$-product is as powerful as the general product.

The monograph is very well-written. It contains a rich and deep material, which is well-arranged. The proofs are clear. It may be recommended as an excellent summary of results concerning $\alpha_i$-products. It may be very useful for researchers and graduates with interest in automata or switching theory and may also be useful for everybody working in computer design and universal algebra.

B. Imreh

**K. Nickel (Editor): Interval Mathematics 1985, (Lecture Notes in Computer Science Vol. 212),** VI+227 pages, Springer-Verlag, Berlin—Heidelberg—New York—Tokio, 1986.

This book provides a collection of selected papers of the International Symposium on Interval Mathematics held in Freiburg i. Br. (Germany) from September 23 to 26, 1985.

Interval arithmetic is a new programming tool for automatic estimation and control of computational errors caused by approximation of real numbers by machine representable numbers. Interval analysis, interval algebra and interval topology, i.e. the interval mathematics are engaged in elaboration of the theoretical base of interval arithmetic. Lately more and more papers are published on diverse aspects of interval mathematics.

The contens of the book are the following: H. Engels, D. an May: Interpolation of an Interval-Valued Function for Arbitrarily Distributed Nodes; H. Fischer: Acceptable Solutions of Linear Interval Integral Equations; Y. Fujii, K. Ichida, M. Ozasa: Maximization of Multivariable Functions Using Interval Analysis; E. Gardenes, H. Mielgo, A. Trepat: Modal Intervals: Reason and Ground Semantics; J. Garloff: Convergent Bounds for the Range of Multivariate Polynomials; T. Giec: On an Interval Comutational Method for Finding the Reachable Set in Time-Optimal Control Problems; H. Kolacz: On the Optimality of Inclusion Algorithms; R. Krawczyk: Interval Operators and Fixed Intervals; F. Krückeberg: Arbitrary Accuracy with Variable Precision Artithmetic; S. Markov, R. Angelov: An Interval Method for Systems of ODE; A. Neumaier: Linear Interval Equations; K. Nickel: How to Fight the Wrapping Effect; M. S. Petkovic, Z. M. Mitrovic, L. D. Petkovic: Arithmetic of Circular Rings; L. B. Rall: Improved Interval Bounds for Ranges of Functions; J. Rohn: Inner Solutions of Linear Interval Systems; K. D. Schmidt: Embedding Theorems for Cones and Applications to Classes of Convex Sets Occuring in Interval Mathematics; Shen Zuhe: Interval Test and Existence Theorem; P. Thieler: Technical Calculations by Means of Interval Mathematics; You Zhaoyong, Xu Zongben, Liu Kunkun: Generalized Theory and Some Specializations of the Region Contraction Algorithm I — Ball Operation.

The book covers various fields of interval analysis, the majority of papers deal with numerical problems. This well edited interesting volume presents the state of the art in interval mathematics. It is recommended for those people interested in the latest results of the field.

T. Csendes

**G. Rozenberg, A. Salomaa (Editors): The book of L,** XV+471 pages, Springer-Verlag, Berlin—Heidelberg—New York—Tokyo, 1986.

The book is dedicated to Artistid Lindenmayer who introduced language-theoretic models in biology referred to as L systems. It contains about 40 articles showing a continuous interest in the topic. Most of them are up-to-date research papers concerning different classes of L systems (e.g. 0L, D0L, DT0L) from formal language theoretical point of view.

"A 0L scheme is a pair $(X, \sigma)$ with $X$ a finite alphabet ad $\sigma$ a finite substitution of $X$ into the free monoid $X^*$. It is deterministic (a D0L scheme) if $\sigma(a)$ is a singleton set for each $a \in X$, and in this case $\sigma$ can be considered an endomorphism of $X^*$, ... A 0L system is a triple $(X, \sigma, \omega)$ such that

$(X, \sigma)$ is a 0L scheme and $\omega \in X^*$ is the axiom of the system. For a 0L system $G=(X, \sigma, \omega)$ one considers the languages

$$L_i(G) = \begin{cases} \{\omega\} & \text{if } i = 0 \\ \{\sigma^i(\omega)\} & \text{if } i > 0. \end{cases}$$

The language of $G$ is the set $L(G)= \bigcup_{i=0}^{\infty} L_i(G)$." (H. Jürgensen, D. E. Metthews)

People interested in applications of L systems find articles in developmental biology, transplantation and software technology.

Á. Makay

**Shuji Tasaka: Performance Analysis of Multiple Access Protocols,** XX+263 pages, The MIT Press, 1986.

This book is included in the Computer Systems Series, Research Reports and Notes, edited by Herb Schwetman. The objective of this book is to describe a unified approach to the performance evaluation problem by means of an approximate analytical technique called equilibrium point analysis. Six multiple access protocols of satellite networks and two of local area networks are analyzed. The book is divided into three parts.

Part I (chapters 1—2) gives the foundations, These are the principles of various multiple access protocols, the states of the performance evaluation and a description of the equilibrium point analysis.

Part II (chapters 3—9) is the main part of the book. It studies the performance (throughput, and average message delay) of S-ALOHA, R-ALOHA, ALOHA-Reservation, TDMA-Reservation SRUC and TDMA protocols of satellite networks. A performance comparison of the six multiple access protocols is given.

Part III (chapters 10—11) deals with CSMA—CD and BRAM protocols of local area networks and compares their performances.

The book is clearly written, many figures help in understanding the models and their behaviors; it can be recommended to readers interested in the performance of protocols, as researchers in computer science, system analysts and network designers.

M. Bohus

A SZERKESZTŐ BIZOTTSÁG CÍME:
6720 SZEGED
SOMOGYI U. 7.

EDITORIAL OFFICE:
6720 SZEGED
SOMOGYI U. 7.
HUNGARY

**Information for authors**

Acta Cybernetica publishes only original papers in the field of computer sciences mainly in English, but also in French, German or Russian. Authors should submit two copies of manuscripts to the Editorial Board. The manuscript must be typed double-spaced on one side of the paper only. Footnotes should be avoided and the number of figures should be as small as possible. For the form of references, see one of the articles previously published in the journal. A list of special symbols used in the manuscript should be supplied by the authors.

A galley proof will be sent to the authors. The first-named autho‹ will receive 50 reprints free of charge.

## INDEX—TARTALOM

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Gécseg Ferenc
A kézirat a nyomdába érkezett: 1986. július 1.
Megjelenés: 1987 április
Terjedelem: 10,5 (A/5) ív
Készült monószedéssel, íves magasnyomással
az MSZ 5601 és az MSZ 5602—55 szabvány szerint
86-2846 — Szegedi Nyomda — F. v.: Surányi Tibor igazgató