Tomus 5. Fasciculus 1.

ACTA CYBERNETICA

FORUM CENTRALE PUBLICATIONUM
CYBERNETICARUM HUNGARICUM

Szeged, 1980
Curat: Universitas Szegediensis de Attila József nominata

5. kötet 1. füzet

ACTA CYBERNETICA

A HAZAI KIBERNETIKAI KUTATÁSOK
KÖZPONTI PUBLIKÁCIÓS FÓRUMA

Szeged, 1980. december
A Szegedi József Attila Tudományegyetem gondozásában

Decidability results concerning tree transducers I

By Z. Ésik

A tree transducer is called functional if its induced transformation is a partial mapping. We show that the functionality of tree transducers is decidable. Consequently, the equivalence problem for deterministic tree transducers is also decidable. The latter result was independently achieved by Z. Zachar in [12] for bottom-up tree transducers and a restricted class of top-down tree transducers. The solvability of the equivalence problem for generalized deterministic sequential machines is known from [2] and [4]. It was proved in [11] that this positive result can not be generalized for arbitrary, i.e. generalized nondeterministic, sequential machines. Therefore, the equivalence problem for nondeterministic tree transducers is undecidable.

Our result can be used to minimize deterministic tree transducers in an effective manner. However, the minimal realizations of a deterministic tree transducer are not isomorphic. We investigate conditions assuring the uniqueness (up to isomorphism) of minimal realizations in certain classes of tree transducers.

Part of the results of the present paper have been announced in [8]. The terminology is used in the sense of [5].

1. Notions and notations

By a *type* $F=\bigcup_{n<\omega} F_n$ we mean a finite type such that $F_0\neq\emptyset$. For the type F, $v(F)=\max\{n\,|\,F_n\neq\emptyset\}$. An *F-algebra* is a system $\mathbf{A}=(A, \{(f)_\mathbf{A}\,|\,f\in F\})$, or shortly, (A, F), where for every nonnegative integer n and $f\in F_n$ $(f)_\mathbf{A}$: $A^n\to A$ is the realization of the n-ary operational symbol f.

Let Y be an arbitrary set. Then $\mathbf{T}_{F,Y}=(T_{F,Y}, F)$ denotes the *free F-algebra generated by Y*. The elements of $T_{F,Y}$ are called *trees* and they can be obtained by induction as follows: $T_{F,Y}$ is the smallest set satisfying

(i) $F_0, Y\subseteq T_{F,Y}$,

(ii) if $n>0$, $f\in F_n$, $t_1, \ldots, t_n\in T_{F,Y}$ then $f(p_1, \ldots, p_n)\in T_{F,Y}$.

In particular, if $Y=X_n$, the set of the first n variables $x_1, \ldots, x_n$ for a nonnegative integer n, $T_{F,Y}$ is denoted by $T_{F,n}$ and $T_{F,0}$ is written T_F. Each *n-ary tree* $p\in T_{F,n}$ induces a mapping $(p)_{\mathbf{A}}: A^n\to A$ in an F-algebra $\mathbf{A}$. If $\mathbf{A}$ is the free algebra $\mathbf{T}_{F,Y}$ then $(p)_{\mathbf{A}}(t_1, \ldots, t_n)=p(t_1, \ldots, t_n)$, i.e. the tree obtained by substituting t_i for x_i $(i=1, \ldots, n)$ in p.

The *depth* (dp), *rank* (rn) and *frontier* (fr) of trees are defined as usually. For a tree $p\in T_{F,Y}$ we have

(i) $\mathrm{dp}(p)=0, \quad \mathrm{rn}(p)=1, \quad \mathrm{fr}(p)=p \quad \text{if} \quad p\in Y,$

(ii) $\mathrm{dp}(p)=0, \quad \mathrm{rn}(p)=1, \quad \mathrm{fr}(p)=\lambda \quad \text{if} \quad p\in F_0,$

(iii) $\mathrm{dp}(p)=1+\max\{\mathrm{dp}(p_i)\mid i=1, \ldots, n\}, \quad \mathrm{rn}(p)=1+\sum_{i=1}^{n}\mathrm{rn}(p_i),$
$\mathrm{fr}(p)=\mathrm{fr}(p_1)\ldots\mathrm{fr}(p_n) \quad \text{if} \quad p=f(p_1, \ldots, p_n), \quad f\in F_n,$

$p_1, \ldots, p_n\in T_{F,Y}$ and $n>0$. Here λ denotes the empty string.

In connection with the elements of $T_{F,n}(n\geqq 0)$ we shall also use the concept of *path*. For an arbitrary $i(1\leqq i\leqq n)$ and $p\in T_{F,n}$ $\mathrm{path}_i(p)$ is given by

(i) $\mathrm{path}_i(p)=\{\lambda\} \quad \text{if} \quad p=x_i,$

(ii) $\mathrm{path}_i(p)=\emptyset \quad \text{if} \quad p\in F_0\cup X_n-\{x_i\},$

(iii) $\mathrm{path}_i(p)=\{jw\mid w\in \mathrm{path}_i(p_j),\ 1\leqq j\leqq m\} \quad \text{if} \quad p=f(p_1, \ldots, p_m),$

$m>0$, $f\in F_m, p_1, \ldots, p_m\in T_{F,n}$. If $\mathrm{path}_i(p)$ is a singleton then it is identified with its unique element. For $w\in\mathrm{path}_i(p)$ we denote by $|w|$ the *length* of w. $\mathrm{path}(p)=\bigcup_{i=1}^{n}\mathrm{path}_i(p)$. For arbitrary two strings v and w v/w denotes the *derivative of v with respect to w*, i.e. $v/w=u$ if and only if $v=wu$.

Further on we shall often use *vector notations* to simplify the treatment. Vectors, except possibly the one dimensional ones, are always denoted by boldfaced letters. For each k dimensional vector $\mathbf{a}\in A^k$ $(k\geqq 0)$ and i $(1\leqq i\leqq k)$ a_i denotes the ith component of $\mathbf{a}$. Conversely, if $a\in A$ then $\mathbf{a}^k\in A^k$ is the k dimensional vector whose each component is equal to a. The product $\mathbf{ab}$ of the k dimensional vectors $\mathbf{a}$ and $\mathbf{b}$ is defined by $\mathbf{ab}=(a_1b_1, \ldots, a_kb_k)$ where a_ib_i are short notations for (a_i, b_i) $(i=1, \ldots, k)$. For the vectors of trees $\mathbf{p}\in T_{F,n}^k$ and $\mathbf{q}\in T_{F,m}^n$ we denote by $\mathbf{p}(\mathbf{q})$ the vector $(p_1(\mathbf{q}), \ldots, p_k(\mathbf{q}))$.

According to the function fr one can distinguish the subset $\hat{T}_{F,n}$ of $T_{F,n}$. This consists of those elements of $T_{F,n}$ whose frontier is a *permutation* of the variables in X_n. We may extend this definition to vectors as follows: $\hat{T}_{F,n}^k=\{\mathbf{p}\in T_{F,n}^k\mid \mathrm{fr}(p_1)\ldots\mathrm{fr}(p_k)$ is a permutation of $X_n\}$. Observe that $\hat{T}_{F,n}^k$ is not the kth power of $\hat{T}_{F,n}$.

We now turn to the definition of tree transducers. Following [5] a *top-down tree transducer* is a system $\mathbf{A}=(F, A, G, A_0, \Sigma)$, where F and G are types, A is a finite, nonvoid set, the set of states, $A_0\subseteq A$ is the set of initial states, finally, Σ is a finite set of top-down rewriting rules. A *top-down rule* has the form $af\to p$ — or equivalently $af(x_1, \ldots, x_n)\to p$, where $n\geqq 0, a\in A, f\in F_n, p\in T_{G,A\times X_n}$. A *bottom-up tree transducer* $\mathbf{A}=(F, A, G, A_0, \Sigma)$ has a similar structure except A_0 is called the set of final states and Σ contains bottom-up rewriting rules. A typical

bottom-up rewriting rule is of form $f(a_1x_1, \ldots, a_nx_n) \to ap$ where $n \geqq 0$, $f \in F_n$, $p \in T_{G,n}$, $a, a_1, \ldots, a_n \in A$. By a tree transducer we mean a top-down or bottom-up transducer.

Take an arbitrary tree transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)$ and let Y be an arbitrary set. Σ can be used to define a binary relation $\overset{*}{\Rightarrow}_{\mathbf{A},Y}$ on $T_{G,A\times T_{F,Y}}$ in the top-down case and on the set $T_{F,A\times T_{G,Y}}$ in the bottom-up case. It is called *derivation* and its exact definition can be found in [5]. If there is no danger of confusion $\mathbf{A}$ is omitted in $\overset{*}{\Rightarrow}_{\mathbf{A},Y}$. It can be seen that if $Y_1 \subseteq Y_2$ and $p, q \in T_{G,A\times T_{F,Y_1}}$ then $p \overset{*}{\Rightarrow}_{Y_1} q$ if and only if $p \overset{*}{\Rightarrow}_{Y_2} q$. Similar equivalence is valid in the bottom-up case. Thus we may omit Y in $\overset{*}{\Rightarrow}_{Y}$.

Again take the tree transducer $\mathbf{A}$. This induces a *transformation* $\tau_{\mathbf{A}} \subseteq T_F \times T_G$:

$$\tau_{\mathbf{A}} = \{(p, q) \mid \exists a_0 \in A_0 \;\; a_0 p \overset{*}{\Rightarrow} q\}$$

in the top-down case, and

$$\tau_{\mathbf{A}} = \{(p, q) \mid \exists a_0 \in A_0 \;\; p \overset{*}{\Rightarrow} a_0 q\}$$

for bottom-up $\mathbf{A}$. If $\tau_{\mathbf{A}}$ is a (partial) function $\mathbf{A}$ is called *functional*. This is always the case if $\mathbf{A}$ is *deterministic*, i.e. different rules have different left sides, moreover, A_0 is a singleton in the top-down case. Two tree transducers are called *equivalent* if their induced transformations coinside. For a tree transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)$ and a state $a \in A$ we denote by $\mathbf{A}(a)$ the transducer $\mathbf{A}(a)=(F, A, G, \{a\}, \Sigma)$.

The domain of the transformation $\tau_{\mathbf{A}}$ is denoted by $\operatorname{dom} \tau_{\mathbf{A}}$. It is a *regular subset* of T_F, i.e. a regular forest. Regular forests are exactly the forests recognized by tree automata. A *tree automaton* is a system $\mathbf{B}=(F, B, B_0)$ with (B, F) a finite F-algebra which is denoted by $\mathbf{B}$ too, $B_0 \subseteq B$ is the set of final states. The *forest recognized by* $\mathbf{B}$ is determined by $T(\mathbf{B})=\{p \in T_F \mid (p)_{\mathbf{B}} \in B_0\}$.

Sometimes we need to restrict a top-down tree transducer to a regular forest. If $\mathbf{A}=(F, A, G, A_0, \Sigma)$ is a top-down tree transducer and $T \subseteq T_F$ is a regular forest then the system $\mathbf{B}=(F, T, A, G, A_0, \Sigma)$ is called a *regularly restricted top-down tree transducer*. Its induced transformation is $\tau_{\mathbf{B}}=\{(p, q) \in \tau_{\mathbf{A}} \mid p \in T\}$. A similar but more general concept is the concept of *top-down tree transducer with regular look-ahead* introduced in [6]. A top-down tree transducer with regular look-ahead is a system $\mathbf{A}=(F, A, G, A_0, \Sigma)$ where F, A, G, A_0 are the same as for top-down tree transducers and Σ is a finite set of rules

$$(af(x_1, \ldots, x_n) \to p;\; R_1, \ldots, R_n)$$

where $af(x_1, \ldots, x_n) \to p$ is a top-down rewriting rule, i.e. $a \in A$, $f \in F_n$ $(n \geqq 0)$, $p \in T_{G,A\times X_n}$, and $R_i \subseteq T_F$ $(1 \leqq i \leqq n)$ are regular forests. The regular forests R_i are used to restrict the applicability of the coressponding top-down rule $af(x_1, \ldots, x_n) \to p$. The rule $(af(x_1, \ldots, x_n) \to p;\; R_1, \ldots, R_n)$ can be applied for a subtree of a tree in $T_{G,A\times T_{F,Y}}$ if and only if it is of form $af(p_1, \ldots, p_n)$ with $p_i \in R_i$ for each i $(1 \leqq i \leqq n)$. Apart from this derivation is defined as for top-down transducers. The *induced transformation* is the relation $\tau_{\mathbf{A}}=\{(p, q) \mid a_0 p \overset{*}{\Rightarrow} q$ for some $a_0 \in A_0\}$. Again, if it is a function $\mathbf{A}$ is called *functional*. It is known that every functional bottom-up or top-down tree transducer is equivalent to some deterministic top-down transducer with regular look-ahead (cf. [7]).

2. The decidability of functionality of tree transducers

First we show that the decision of functionality of bottom-up transducers is reducible to the decision of functionality of regularly restricted top-down ones.

Let $\mathbf{A}=(F, A, G, A_0, \Sigma)$ be an arbitrary bottom-up transducer. Define the top-down transducer with regular look-ahead $\mathbf{A}'$ as follows: $\mathbf{A}'=(F, A, G, A_0, \Sigma')$ where

$$\Sigma' = \{(af \to p(a_1x_1, \ldots, a_nx_n);\ R_1, \ldots, R_n) \mid f(a_1x_1, \ldots, a_nx_n) \to ap \in \Sigma,$$
$$R_i = \operatorname{dom} \tau_{\mathbf{A}(a_i)}\ (i = 1, \ldots, n)\}.$$

Lemma 1. $\mathbf{A}$ is functional if and only if $\mathbf{A}'$ is functional.

Proof. It is obvious that $\tau_{\mathbf{A}} \subseteq \tau_{\mathbf{A}'}$. Therefore if $\mathbf{A}'$ is functional then $\mathbf{A}$ is functional, too. To prove the converse first we show that if $ap \overset{*}{\Rightarrow}_{\mathbf{A}'} q$ and $a'p \overset{*}{\Rightarrow}_{\mathbf{A}'} q'$ where $a, a' \in A$, $p \in T_F$, $q, q' \in T_G$ and $q \neq q'$ then there exist different trees $r, r' \in T_G$ such that $p \overset{*}{\Rightarrow}_{\mathbf{A}} br$ and $p \overset{*}{\Rightarrow}_{\mathbf{A}} b'r'$ are also satisfied for certain choise of states b, b' with $\{b, b'\} \subseteq \{a, a'\}$. We shall prove this by induction on p. The basis, $p \in F_0$, is immediate. Suppose now that $p=f(p_1, \ldots, p_n)$ where $n>0$, $f \in F_n$, $p_1, \ldots, p_n \in T_F$. Since $ap \overset{*}{\Rightarrow} q$ and $a'p \overset{*}{\Rightarrow} q'$ there exist rules $f(a_1x_1, \ldots, a_nx_n) \to aq_0$, $f(a_1'x_1, \ldots, a_n'x_n) \to a'q_0' \in \Sigma$ with $p_i \in \operatorname{dom} \tau_{\mathbf{A}(a_i)} \cap \operatorname{dom} \tau_{\mathbf{A}(a_i')}$ and satisfying $q_0(a_1p_1, \ldots, a_np_n) \overset{*}{\Rightarrow} q$ and $q_0'(a_1'p_1, \ldots, a_n'p_n) \overset{*}{\Rightarrow} q'$, respectively. We distinguish two cases.

Firstly assume that for each $i \in \{1, \ldots, n\}$ if x_i appears in $\operatorname{fr}(q_0)$ then there exists exactly one tree $q_i \in T_G$ with $a_ip_i \overset{*}{\Rightarrow} q_i$. Then also $p_i \overset{*}{\Rightarrow} a_iq_i$. This and $p_i \in \operatorname{dom} \tau_{\mathbf{A}(a_i)}$ $(i=1, \ldots, n)$ yield $p \overset{*}{\Rightarrow} aq$. Similarly, we get $p \overset{*}{\Rightarrow} a'q'$ if, for each x_i occuring in $\operatorname{fr}(q_0')$, there is only one tree in T_G which can be derived from $a_i'p_i$. This proves our assertion in the first case.

Secondly assume that there is an integer $i \in \{1, \ldots, n\}$ such that x_i appears in $fr(q_0)$ and there are different trees $q_i, q_i' \in T_G$ with $a_ip_i \overset{*}{\Rightarrow} q_i$ and $a_ip_i \overset{*}{\Rightarrow} q_i'$, respectively. Then, by the induction hypothesis, there exist trees $r_i \neq r_i' \in T_G$ satisfying both $p_i \overset{*}{\Rightarrow} a_ir_i$ and $p_i \overset{*}{\Rightarrow} a_ir_i'$. For each index $j (j \neq i)$ choose $r_j \in T_G$ in such a way that we have $p_j \overset{*}{\Rightarrow} a_jr_j$. This can be done by $p_j \in \operatorname{dom} \tau_{\mathbf{A}(a_j)}$. Now let $r=q_0(r_1, \ldots, r_n)$, $r'=q_0(r_1, \ldots, r_{i-1}, r_i', r_{i+1}, \ldots, r_n)$. $r \neq r'$ because $r_i \neq r_i'$. On the other hand $p \overset{*}{\Rightarrow} ar$ and $p \overset{*}{\Rightarrow} ar'$.

Now assume that $\mathbf{A}'$ is not functional. Then there exist trees $p \in T_F$, $q \neq q' \in T_G$ and initial states $a_0, a_0' \in A_0$ such that both $a_0p \overset{*}{\Rightarrow}_{\mathbf{A}'} q$ and $a_0'p \overset{*}{\Rightarrow}_{\mathbf{A}'} q'$ are satisfied. By the previous considerations it follows that there are different trees $r, r' \in T_G$ with $p \overset{*}{\Rightarrow}_{\mathbf{A}} b_0r$ and $p \overset{*}{\Rightarrow}_{\mathbf{A}} b_0'r'$ where each of the states b_0 and b_0' denotes either a_0 or a_0'. This means that both (p, r) and (p, r') are in $\tau_{\mathbf{A}}$, i.e. $\mathbf{A}$ is not functional.

Lemma 2. The decision of functionality of bottom-up tree transducers is reducible to the decision of functionality of regularly restricted top-down ones.

Proof. Let $\mathbf{A}$ be an arbitrary bottom-up transducer and $\mathbf{A}'$ the top-down transducer with regular look-ahead constructed in the previous lemma. We know that $\mathbf{A}$ is functional if and only if $\mathbf{A}'$ is functional. By Theorem 2.6 in [6] we have

$\tau_{A'}=\tau\circ\tau_{\mathbf{B}}$ where τ is a deterministic bottom-up relabeling, i.e. a transformation induced by a special deterministic bottom-up transducer, and $\mathbf{B}$ is a top-down transducer. Since τ is a function $\mathbf{A}'$ is functional if and only if $\mathbf{B}$ restricted to the regular forest τ (dom $\tau_{A'}$) is functional. Note that dom $\tau_{\mathbf{A}}=$dom $\tau_{\mathbf{A}'}$.. As one can construct the transducers $\mathbf{A}'$ and $\mathbf{B}$ in an effective manner this proves Lemma 2.

Now let us fix an arbitrary regularly restricted top-down tree transducer $\mathbf{A}=(F, T, A, G, A_0, \Sigma)$ and a tree automaton $\mathbf{B}=(F, B, B_0)$ recognizing T. Set

$$P=\{p\in T\mid \exists q\neq q'\in T_G\ \ (p, q), (p, q')\in\tau_{\mathbf{A}}\}.$$

In the next five lemmas we shall present five reduction rules. Each reduction rule produces a smaller tree $p'\in P$ for a tree $p\in T$ if it can be applied for p.

Lemma 3. Let $p_1, p_2\in\hat{T}_{F,1}, p_3\in T_F,\quad n_1, n_1', n_2, n_2'\geqq 0,\quad q_1\in\hat{T}_{G,n_1}, q_1'\in\hat{T}_{G,n_1'}$, $\mathbf{q}_2\in\hat{T}^{n_1}_{G,n_2}$, $\mathbf{q}_2'\in\hat{T}^{n_1'}_{G,n_2}$, $\mathbf{q}_3\in T^{n_2}_G$, $\mathbf{q}_3'\in T^{n_2'}_G$, $a_0, a_0'\in A_0$, $\mathbf{a}_i\in A^{n_i}$, $\mathbf{a}_i'\in A^{n_i'}$ $(i=1, 2)$. Let us denote by A_i and A_i' the sets $A_i=\{a_{i,j}\mid 1\leqq j\leqq n_i\}$ and $A_i'=\{a_{i,j}'\mid 1\leqq j\leqq n_i'\}$ $(i=1, 2)$ respectively. Assume that each of the following conditions is satisfied:

(i) $p_1(p_2(p_3))\in T$,

(ii) $a_0p_1\overset{*}{\Rightarrow}q_1(\mathbf{a}_1\mathbf{x}_1^{n_1}),\quad a_0'p_1\overset{*}{\Rightarrow}q_1'(\mathbf{a}_1'\mathbf{x}_1^{n_1'})$,

(iii) $\mathbf{a}_1\mathbf{p}_2^{n_1}\overset{*}{\Rightarrow}\mathbf{q}_2(\mathbf{a}_2\mathbf{x}_1^{n_2}),\quad \mathbf{a}_1'\mathbf{p}_2^{n_1'}\overset{*}{\Rightarrow}\mathbf{q}_2'(\mathbf{a}_2'\mathbf{x}_1^{n_2'})$,

(iv) $\mathbf{a}_2\mathbf{p}_3^{n_2}\overset{*}{\Rightarrow}\mathbf{q}_3,\quad \mathbf{a}_2'\mathbf{p}_3^{n_2'}\overset{*}{\Rightarrow}\mathbf{q}_3'$,

(v) $(p_3)_{\mathbf{B}}=(p_2(p_3))_{\mathbf{B}},\quad A_1\subseteqq A_2,\quad A_1'\subseteqq A_2'$,

(vi) $q_1(\mathbf{r})\neq q_1'(\mathbf{r}')$ holds for any $\mathbf{r}\in T_G^{n_1}$ and $\mathbf{r}'\in T_G^{n_1'}$.

Then $p_1(p_3)\in P$.

Proof. First note that our assumptions imply the condition $p_1(p_2(p_3))\in P$. From now on let $[n]$ denote the set of the first n positive integers for every $n\geqq 0$. Thus [0] is the empty set. Let $\varphi\colon[n_1]\to[n_2]$ and $\varphi'\colon[n_1']\to[n_2']$ be mappings with $a_{1,i}=a_{2,\varphi(i)}$ $(i\in[n_1])$ and $a_{1,i}'=a_{2,\varphi'(i)}'$ $(i\in[n_2])$, respectively. Obviously we have $\mathbf{a}_1\mathbf{p}_3^{n_1}\overset{*}{\Rightarrow}\mathbf{r}$ and $\mathbf{a}_1'\mathbf{p}_3^{n_1'}\overset{*}{\Rightarrow}\mathbf{r}'$ where $\mathbf{r}=(q_{3,\varphi(1)}, \dots, q_{3,\varphi(n_1)})$, $\mathbf{r}'=(q_{3,\varphi'(1)}', \dots, q_{3,\varphi'(m_1)}')$. By (ii) this implies that $a_0p_1(p_3)\overset{*}{\Rightarrow}q_1(\mathbf{r})$ and $a_0'p_1(p_3)\overset{*}{\Rightarrow}q_1'(\mathbf{r}')$. On the other hand $q_1(\mathbf{r})\neq q_1'(\mathbf{r}')$ by our assumption (vi). Furthermore, $p_1(p_3)\in T$ holds by (v). Hence $p_1(p_3)\in P$.

Lemma 4. Let $p_1\in\hat{T}_{F,1}, p_2\in T_F,\ n, n'>0,\ q_1\in\hat{T}_{G,n},\ q_1'\in\hat{T}_{G,n'},\ \mathbf{q}_2\in T_G^n,\ \mathbf{q}_2'\in T_G^{n'}$, $a_0, a_0'\in A_0, \mathbf{a}\in A^n, \mathbf{a}'\in A^{n'}$. Let $|A|$ and $|B|$ denote the cardinality of A and B, respectively and let $\|A\|=2^{|A|}$, $K=\max\{\mathrm{dp}(q)\mid\exists a\in A, p\in T_{F,X}\ \ ap\to q\in\Sigma\}$. Assume that the following conditions are valid:

(i) $p_1(p_2)\in T$,

(ii) $a_0p_1\overset{*}{\Rightarrow}q_1(\mathbf{a}\mathbf{x}_1^n),\quad a_0'p_1\overset{*}{\Rightarrow}q_1'(\mathbf{a}'\mathbf{x}_1^{n'})$,

(iii) $\mathbf{a}p_2^n \overset{*}{\Rightarrow} \mathbf{q}_2$, $\mathbf{a}'p_2^{n'} \overset{*}{\Rightarrow} \mathbf{q}_2'$,

(iv) $\mathrm{path}_1(q_1)$ is a prefix of $\mathrm{path}_1(q_1')$,

$$|\mathrm{path}_1(q_1')| - |\mathrm{path}_1(q_1)| > \|A\|^2|B|K, \quad \mathrm{dp}(p_2) \geqq \|A\|^2|B|.$$

Then there is a tree $r \in T_F$ such that $p_1(r) \in P$ and $\mathrm{rn}(r) < \mathrm{rn}(p_2)$.

Proof. Let R be the forest defined by

$$R = \{r \in T_F \mid p_1(r) \in T, \quad \mathrm{rn}(r) \leqq \mathrm{rn}(p_2), \quad \exists \mathbf{s} \in T_G^n, \mathbf{s}' \in T_G^{n'} \quad \mathbf{a}r^n \overset{*}{\Rightarrow} \mathbf{s}, \quad \mathbf{a}'\mathbf{r}^{n'} \overset{*}{\Rightarrow} \mathbf{s}'\}.$$

Since $p_2 \in R$ R is nonvoid. Let r be an element of R with minimal rank. We shall show that $p_1(r) \in P$ and $\mathrm{dp}(r) < \|A\|^2 B$.

Assume that the condition $\mathrm{dp}(r) < \|A\|^2 B$ does not hold. In this case there exist

$$r_1, r_2 \in \hat{T}_{F,1}, \ r_3 \in T_F, \ m_1, m_1', m_2, m_2' \geqq 0, \ \mathbf{s}_1 \in \hat{T}_{G,m_1}^n, \ \mathbf{s}_1' \in \hat{T}_{G,m_1'}^{n'},$$

$$\mathbf{s}_2 \in \hat{T}_{G,m_2}^{m_1}, \ \mathbf{s}_2' \in \hat{T}_{G,m_2'}^{m_1'}, \ \mathbf{s}_3 \in \hat{T}_G^{m_2}, \ \mathbf{s}_3' \in \hat{T}_G^{m_2'}, \ \mathbf{b}_i \in A^{m_i}, \ \mathbf{b}_i' \in A^{m_i'} \quad (i = 1, 2)$$

such that each of the following five conditions is satisfied:

(1) $r = r_1(r_2(r_3)), \quad r_2 \neq x_1$,

(2) $\mathbf{a}\mathbf{r}_1^n \overset{*}{\Rightarrow} \mathbf{s}_1(\mathbf{b}_1\mathbf{x}_1^{m_1}), \quad \mathbf{a}'\mathbf{r}_1^{n'} \overset{*}{\Rightarrow} \mathbf{s}_1'(\mathbf{b}_1'\mathbf{x}_1^{m_1'})$,

(3) $\mathbf{b}_1\mathbf{r}_2^{m_1} \overset{*}{\Rightarrow} \mathbf{s}_2(\mathbf{b}_2\mathbf{x}_1^{m_2}), \ \mathbf{b}_1'\mathbf{r}_2^{m_1'} \overset{*}{\Rightarrow} \mathbf{s}_2'(\mathbf{b}_2'\mathbf{x}_1^{m_2'})$,

(4) $\mathbf{b}_2\mathbf{r}_3^{m_2} \overset{*}{\Rightarrow} \mathbf{s}_3, \ \mathbf{b}_2'\mathbf{r}_3^{m_2'} \overset{*}{\Rightarrow} \mathbf{s}_3'$,

(5) $(r_3)_B = (r_2(r_3))_B, \quad B_1 \subseteq B_2, \quad B_1' \subseteq B_2'$, where

$$B_i = \{b_{i,j} \mid 1 \leqq j \leqq m_i\}, \ B_i' = \{b_{i,j}' \mid 1 \leqq j \leqq m_i'\} \quad (i = 1, 2).$$

Now let φ: $[m_1] \to [m_2]$, φ': $[m_1'] \to [m_2']$ be mappings satisfying the equalities $b_{1,i} = b_{2,\varphi(i)}$ $(i \in [m_1])$, $b_{1,i}' = b_{2,\varphi'(i)}'$ $(i \in [m_1'])$. It is immediate that $\mathbf{a}\mathbf{r}_1(\mathbf{r}_3)^n \overset{*}{\Rightarrow} \overset{*}{\Rightarrow} \mathbf{s}_1(s_{3,\varphi(1)}, \ldots, s_{3,\varphi(m_1)})$ and $\mathbf{a}'\mathbf{r}_1(\mathbf{r}_3)^{n'} \overset{*}{\Rightarrow} \mathbf{s}_1'(s_{3,\varphi'(1)}', \ldots, s_{3,\varphi'(m_1')}')$. This, together with $(r_1(r_3))_B = (r)_B$ yields that $r_1(r_3) \in R$, which is a contradiction because $\mathrm{rn}(r_1(r_3)) < \mathrm{rn}(r)$.

Therefore, $\mathrm{dp}(r) < \|A\|^2|B|$. This implies that for every $\mathbf{s} \in T_G^n$ and $\mathbf{s}' \in T_G^{n'}$ if the derivations $\mathbf{a}r^n \overset{*}{\Rightarrow} \mathbf{s}$ and $\mathbf{a}'\mathbf{r}^{n'} \overset{*}{\Rightarrow} \mathbf{s}'$ exist then $\mathrm{dp}(s_1), \mathrm{dp}(s_1') \leqq \|A\||B|K$, thus, by (iv), $p_1(r) \in P$. Since r was of minimal rank this ends the proof of Lemma 4.

Lemma 5. Let $p_1, p_2, p_3 \in \hat{T}_{F,1}, p_4 \in T_F, n_i, n_i', m_i \geqq 0$ $(i = 1, 2, 3)$, $q_1 \in \hat{T}_{G,n_1+1}$, $q_1' \in \hat{T}_{G,n_1'+1}$, $r_1 \in \hat{T}_{G,m_1}$, $\mathbf{q}_2 \in \hat{T}_{G,n_2}^{n_1}$, $\mathbf{q}_2' \in \hat{T}_{G,n_2'}^{n_1'}$, $\mathbf{r}_2 \in \hat{T}_{G,m_2}^{m_1}$, $\mathbf{q}_3 \in \hat{T}_{G,n_3}^{n_2}$, $\mathbf{q}_3' \in \hat{T}_{G,n_3'}^{n_2'}$, $\mathbf{r}_3 \in \hat{T}_{G,m_3}^{m_2}$, $\mathbf{q}_4 \in T_G^{n_3}$, $\mathbf{q}_4' \in T_G^{n_3'}$, $\mathbf{r}_4 \in T_G^{m_3}$, $a_0, a_0' \in A_0$, $a \in A$, $\mathbf{a}_i \in A^{n_i}$, $\mathbf{a}_i' \in A^{n_i'}$, $\mathbf{b}_i \in A^{m_i}$ $(i = 1, 2, 3)$. Finally, let $v \in T_G$ and $v' = r_1(\mathbf{r}_2(\mathbf{r}_3(\mathbf{r}_4)))$. Denote by A_i, A_i' and B_i $(i = 1, 2, 3)$ the sets of components of $\mathbf{a}_i$, $\mathbf{a}_i'$ and $\mathbf{b}_i$, respectively. Assume that the following conditions are satisfied:

(i) $p_1(p_2(p_3(p_4)))\in T$,

(ii) $a_0 p_1 \overset{*}{\Rightarrow} q_1(ax_1, \mathbf{a}_1\mathbf{x}_1^{n_1})$, $a_0' p_1 \overset{*}{\Rightarrow} q_1'(r_1(\mathbf{b}_1\mathbf{x}_1^{m_1}), \mathbf{a}_1'\mathbf{x}_1^{n_1'})$,

(iii) $\mathbf{a}_1\mathbf{p}_2^{n_1} \overset{*}{\Rightarrow} \mathbf{q}_2(\mathbf{a}_2\mathbf{x}_1^{n_2})$, $\mathbf{a}_1'\mathbf{p}_2^{n_1'} \overset{*}{\Rightarrow} \mathbf{q}_2'(\mathbf{a}_2'\mathbf{x}_1^{n_2'})$, $ap_2 \overset{*}{\Rightarrow} ax_1$, $\mathbf{b}_1\mathbf{p}_2^{m_1} \overset{*}{\Rightarrow} \mathbf{r}_2(\mathbf{b}_2\mathbf{x}_1^{m_2})$,

(iv) $\mathbf{a}_2\mathbf{p}_3^{n_2} \overset{*}{\Rightarrow} \mathbf{q}_3(\mathbf{a}_3\mathbf{x}_1^{n_3})$, $\mathbf{a}_2'\mathbf{p}_3^{n_2'} \overset{*}{\Rightarrow} \mathbf{q}_3'(\mathbf{a}_3'\mathbf{x}_1^{n_3'})$, $ap_3 \overset{*}{\Rightarrow} ax_1$, $\mathbf{b}_2\mathbf{p}_3^{m_2} \overset{*}{\Rightarrow} \mathbf{r}_3(\mathbf{b}_3\mathbf{x}_1^{m_3})$,

(v) $\mathbf{a}_3\mathbf{p}_4^{n_3} \overset{*}{\Rightarrow} \mathbf{q}_4$, $\mathbf{a}_3'\mathbf{p}_4^{n_3'} \overset{*}{\Rightarrow} \mathbf{q}_4'$, $ap_4 \overset{*}{\Rightarrow} v$, $\mathbf{b}_3\mathbf{p}_4^{m_3} \overset{*}{\Rightarrow} \mathbf{r}_4$,

(vi) $(p_4)_B = (p_3(p_4))_B = (p_2(p_3(p_4)))_B$,

$A_1 \subseteq A_2 \subseteq A_3$, $A_1' \subseteq A_2' \subseteq A_3'$, $B_1 = B_2 \subseteq B_3$,

(vii) $v \neq v'$, $\mathrm{path}_1(q_1) = \mathrm{path}_1(q_1')$.

Then at least one of the trees $p_1(p_2(p_4))$, $p_1(p_3(p_4))$ and $p_1(p_4)$ is in P.

Proof. First observe that by the assumptions of the lemma it follows that $p_1(p_2(p_3(p_4)))\in P$.

Let $\varphi_i\colon [n_i]\to[n_{i+1}]$, $\varphi_i'\colon [n_i']\to[n_{i+1}']$ and $\psi_i\colon [m_i]\to[m_{i+1}]$ $(i=1, 2)$ be mappings such that we have $a_{i,j}=a_{i+1,\varphi_i(j)}$ $(i=1, 2, j\in[n_i])$, $a_{i,j}'=a_{i+1,\varphi_i'(j)}'$ $(i=1, 2, j\in[n_i'])$, $b_{i,j}=b_{i+1,\psi_i(j)}$ $(i=1, 2, j\in[m_i])$. Furthermore, let $\varphi_3=\varphi_1\circ\varphi_2$, $\varphi_3'=\varphi_1'\circ\varphi_2'$, $\psi_3=\psi_1\circ\psi_2$.

Let us introduce the following notations:

$$\mathbf{s}_1 = (q_{3,\varphi_1(1)}, \ldots, q_{3,\varphi_1(n_1)})(\mathbf{q}_4),$$

$$\mathbf{s}_1' = (q_{3,\varphi_1'(1)}', \ldots, q_{3,\varphi_1'(n_1')}')(\mathbf{q}_4'),$$

$$\mathbf{t}_1 = (r_{3,\psi_1(1)}, \ldots, r_{3,\psi_1(m_1)})(\mathbf{r}_4),$$

$$\mathbf{s}_2 = \mathbf{q}_2(q_{4,\varphi_2(1)}, \ldots, q_{4,\varphi_2(n_2)}),$$

$$\mathbf{s}_2' = \mathbf{q}_2'(q_{4,\varphi_2'(1)}', \ldots, q_{4,\varphi_2'(n_2')}'),$$

$$\mathbf{t}_2 = \mathbf{r}_2(r_{4,\psi_2(1)}, \ldots, r_{4,\psi_2(m_2)}),$$

$$\mathbf{s}_3 = (q_{4,\varphi_3(1)}, \ldots, q_{4,\varphi_3(n_1)}),$$

$$\mathbf{s}_3' = (q_{4,\varphi_3'(1)}', \ldots, q_{4,\varphi_3'(n_1')}'),$$

$$\mathbf{t}_3 = (r_{4,\psi_3(1)}, \ldots, r_{4,\psi_3(m_1)}).$$

It is easy to check that each of the following derivations is valid: $a_0p_1(p_3(p_4))\overset{*}{\Rightarrow}$ $\overset{*}{\Rightarrow}q_1(v, \mathbf{s}_1)$, $a_0'p_1(p_3(p_4))\overset{*}{\Rightarrow}q_1'(r_1(\mathbf{t}_1), \mathbf{s}_1')$, $a_0p_1(p_2(p_4))\overset{*}{\Rightarrow}q_1(v, \mathbf{s}_2)$, $a_0'p_1(p_2(p_4))\overset{*}{\Rightarrow}$ $\overset{*}{\Rightarrow}q_1'(r_1(\mathbf{t}_2), \mathbf{s}_2')$, $a_0p_1(p_4)\overset{*}{\Rightarrow}q_1(v, \mathbf{s}_3)$, $a_0'p_1(p_4)\overset{*}{\Rightarrow}q_1'(r_1(\mathbf{t}_3), \mathbf{s}_3')$. On the other hand $p_1(p_3(p_4))$, $p_1(p_2(p_4))$, $p_1(p_4)\in T$.

Assume that $p_1(p_2(p_4))\notin P$. Then, by (vii), it follows that $m_1, m_2, m_3>0$ and there is an integer $i\in[m_2]$ with $r_{3,i}(\mathbf{r}_4)\neq r_{4,\psi_2(i)}$. Without loss of generality we may assume that this integer i is in the range of ψ_1, i.e. there exist $j\in[m_1]$ satisfying

$\psi_1(j)=i$. Now suppose that neither $p_1(p_3(p_4))$ nor $p_1(p_4)$ is in P. Then $r_1(\mathbf{t}_1)=r_1(\mathbf{t}_2)=r_1(\mathbf{t}_3)$ $(=v)$. But this is impossible because $t_{1,j}\neq t_{3,j}$.

Note that Lemma 5 remains valid even if $A_2'\subseteq A_3'$ and $B_2\subseteq B_3$ are replaced by $A_2'\cup B_2\subseteq A_3'\cup B_3$.

The proof of the next lemma is similar to the previous one.

Lemma 6. Let $p_1, p_2, p_3\in\hat{T}_{F,1}$, $p_4\in T_F$, $n_i, n_i', m_i\geqq 0$ $(i=1,2,3)$, $q_1\in\hat{T}_{G,n_1+1}$, $q_1'\in\hat{T}_{G,n_1'+1}$, $r_1\in\hat{T}_{G,m_1}$, $\mathbf{q}_2\in\hat{T}^{n_1}_{G,n_2}$, $\mathbf{q}_2'\in\hat{T}^{n_1'}_{G,n_2'}$, $\mathbf{r}_2\in\hat{T}^{m_1}_{G,m_2}$, $\mathbf{q}_3\in\hat{T}^{n_2}_{G,n_3}$, $\mathbf{q}_3'\in\hat{T}^{n_2'}_{G,n_3'}$, $\mathbf{r}_3\in\hat{T}^{m_2}_{G,m_3}$, $\mathbf{q}_4\in T_G^{n_3}$, $\mathbf{q}_4'\in T_G^{n_3'}$, $\mathbf{r}_4\in T_G^{m_3}$, $a_0, a_0'\in A_0$, $\mathbf{a}_i\in A^{n_i}$, $\mathbf{a}_i'\in A^{n_i'}$, $\mathbf{b}_i\in A^{m_i}$ $(i=1,2,3)$. Furthermore, let $v'\in T_G$ and $v=r_1(\mathbf{r}_2(\mathbf{r}_3(\mathbf{r}_4)))$. Denote by A_i, A_i' and B_i $(i=1,2,3)$ the sets of components of $\mathbf{a}_i$, $\mathbf{a}_i'$ and $\mathbf{b}_i$, respectively. Assume that

(i) $p_1(p_2(p_3(p_4)))\in T$,

(ii) $a_0p_1\overset{*}{\Rightarrow} q_1(r_1(\mathbf{b}_1\mathbf{x}_1^{m_1}), \mathbf{a}_1\mathbf{x}_1^{n_1})$, $a_0'p_1\overset{*}{\Rightarrow} q_1'(v', \mathbf{a}_1'\mathbf{x}_1^{n_1'})$,

(iii) $\mathbf{a}_1\mathbf{p}_2^{n_1}\overset{*}{\Rightarrow}\mathbf{q}_2(\mathbf{a}_2\mathbf{x}_1^{n_2})$; $\mathbf{a}_1'\mathbf{p}_2^{n_1'}\overset{*}{\Rightarrow}\mathbf{q}_2'(\mathbf{a}_2'\mathbf{x}_1^{n_2'})$, $\mathbf{b}_1\mathbf{p}_2^{m_1}\overset{*}{\Rightarrow}\mathbf{r}_2(\mathbf{b}_2\mathbf{x}_1^{m_2})$,

(iv) $\mathbf{a}_2\mathbf{p}_3^{n_2}\overset{*}{\Rightarrow}\mathbf{q}_3(\mathbf{a}_3\mathbf{x}_1^{n_3})$, $\mathbf{a}_2'\mathbf{p}_3^{n_2'}\overset{*}{\Rightarrow}\mathbf{q}_3'(\mathbf{a}_3'\mathbf{x}_1^{n_3'})$, $\mathbf{b}_2\mathbf{p}_3^{m_2}\overset{*}{\Rightarrow}\mathbf{r}_3(\mathbf{b}_3\mathbf{x}_1^{m_3})$,

(v) $\mathbf{a}_3\mathbf{p}_4^{n_3}\overset{*}{\Rightarrow}\mathbf{q}_4$, $\mathbf{a}_3'\mathbf{p}_4^{n_3'}\overset{*}{\Rightarrow}\mathbf{q}_4'$, $\mathbf{b}_3\mathbf{p}_4^{m_3}\overset{*}{\Rightarrow}\mathbf{r}_4$,

(vi) $(p_4)_{\mathbf{B}}=(p_3(p_4))_{\mathbf{B}}=(p_2(p_3(p_4)))_{\mathbf{B}}$,

$A_1\subseteq A_2\subseteq A_3$, $A_1'\subseteq A_2'\subseteq A_3'$, $B_1=B_2\subseteq B_3$,

(vii) $v\neq v'$, $\mathrm{path}_1(q_1)=\mathrm{path}_1(q_1')$.

Then at least one of the trees $p_1(p_2(p_4))$, $p_1(p_3(p_4))$, $p_1(p_4)$ is in P.

Our last lemma is stated as follows:

Lemma 7. Let $p_1, p_2\in\hat{T}_{F,1}$, $p_3\in T_F$, $k, l, m, k', l', m'\geqq 0$, $q_1\in\hat{T}_{G,k+1}$, $q_1'\in\hat{T}_{G,k'+1}$, $q_2\in\hat{T}_{G,l+1}$, $q_2'\in\hat{T}_{G,l'+1}$, $\mathbf{r}\in\hat{T}^k_{G,m}$, $\mathbf{r}'\in\hat{T}^{k'}_{G,m'}$, $q_3\in\hat{T}_{G,1}$, q_3', $v\in T_G$, $\mathbf{s}\in T_G^l$, $\mathbf{s}'\in T_G^{l'}$, $\mathbf{t}\in T_G^m$, $\mathbf{t}'\in T_G^{m'}$, $a_0, a_0'\in A_0$, $a, a'\in A$, $\mathbf{a}\in A^k$, $\mathbf{a}'\in A^{k'}$, $\mathbf{b}\in A^l$, $\mathbf{b}'\in A^{l'}$, $\mathbf{c}\in A^m$, $\mathbf{c}'\in A^{m'}$. Let A_1, B_1 and C_1 denote the sets of all components of $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, respectively. Similarly, denote by A_1', B_1' and C_1' the sets of components of $\mathbf{a}'$, $\mathbf{b}'$ and $\mathbf{c}'$. Suppose that the following conditions are satisfied:

(i) $p_1(p_2(p_3))\in T$,

(ii) $a_0p_1\overset{*}{\Rightarrow} q_1(ax_1, \mathbf{a}\mathbf{x}_1^k)$, $a_0'p_1\overset{*}{\Rightarrow} q_1'(a'x_1, \mathbf{a}'\mathbf{x}_1^{k'})$,

(iii) $ap_2\overset{*}{\Rightarrow} q_2(ax_1, \mathbf{b}\mathbf{x}_1^l)$, $a'p_2\overset{*}{\Rightarrow} q_2'(a'x_1, \mathbf{b}'\mathbf{x}_1^{l'})$,

$\mathbf{a}\mathbf{p}_2^k\overset{*}{\Rightarrow}\mathbf{r}(\mathbf{c}\mathbf{x}_1^m)$, $\mathbf{a}'\mathbf{p}_2^{k'}\overset{*}{\Rightarrow}\mathbf{r}'(\mathbf{c}'\mathbf{x}_1^{m'})$,

(iv) $ap_3\overset{*}{\Rightarrow} q_3(v)$, $a'p_3\overset{*}{\Rightarrow} q_3'$, $\mathbf{b}\mathbf{p}_3^l\overset{*}{\Rightarrow}\mathbf{s}$, $\mathbf{b}'\mathbf{p}_3^{l'}\overset{*}{\Rightarrow}\mathbf{s}'$, $\mathbf{c}\mathbf{p}_3^m\overset{*}{\Rightarrow}\mathbf{t}$; $\mathbf{c}'\mathbf{p}_3^{m'}\overset{*}{\Rightarrow} t'$,

(v) $A_1\subseteq B_1\cup C_1$, $A_1'\subseteq B_1'\cup C_1'$, $(p_3)_{\mathbf{B}}=(p_2(p_3))_{\mathbf{B}}$,

(vi) $\mathrm{path}_1(q_1') = \mathrm{path}_1(q_1)\,\mathrm{path}(q_3)$, $\mathrm{path}_1(q_2)\,\mathrm{path}(q_3) = \mathrm{path}(q_3)\,\mathrm{path}_1(q_2')$, $v \neq q_3'$.

Then $p_1(p_3) \in P$.

Proof. Let us introduce the following notations: $\mathbf{d}=(\mathbf{b}, \mathbf{c})$, $\mathbf{d}'=(\mathbf{b}', \mathbf{c}')$, $\mathbf{u}=(\mathbf{s}, \mathbf{t})$, $\mathbf{u}'=(\mathbf{s}', \mathbf{t}')$. Choose the mappings $\varphi\colon [k]\to[l+m]$ and $\varphi'\colon [k']\to[l'+m']$ in such a way that we have $a_i=d_{\varphi(i)}$ and $a_j'=d_{\varphi'(j)}$ for every $i\in[k]$ and $j\in[k']$. Obviously, $a_0 p_1(p_3) \overset{*}{\Rightarrow} q_1(q_3(v), u_{\varphi(1)}, \ldots, u_{\varphi(k)})$ and $a_0' p_1(p_3) \overset{*}{\Rightarrow} q_1'(q_3', u_{\varphi'(1)}', \ldots, u_{\varphi'(k')}')$, furthermore, $p_1(p_3) \in T$. On the other hand $\mathrm{path}_1(q_1(q_3, u_{\varphi(1)}, \ldots, u_{\varphi(k)})) = \mathrm{path}_1(q_1'(x_1, u_{\varphi(1)}', \ldots, u_{\varphi'(k')}'))$ and $q_3' \neq v$. Therefore, $q_1(q_3(v), u_{\varphi(1)}, \ldots, u_{\varphi(k)}) \neq q_1'(q_3', u_{\varphi'(1)}', \ldots, u_{\varphi'(k')}')$, showing that $p_1(p_3) \in P$.

We are now able to prove our main result:

Theorem 8. The functionality of top-down as well as bottom-up tree transducers is decidable.

Proof. By Lemma 2 it suffices to prove our statement for regularly restricted top-down transducers. Hence take an arbitrary regularly restricted top-down transducer $\mathbf{A}=(F, T, A, G, A_0, \Sigma)$ with $T=T(\mathbf{B})$, where $\mathbf{B}$ is the tree automaton $\mathbf{B}=(F, B, B_0)$. Define the set P and integers $|A|$, $\|A\|$, $|B|$ and K as previously (cf. Lemma 4) and let L denote the number of nonempty strings over $[\nu(G)]$ with length not exceeding $\|A\|^2|B|K$. Furthermore, let $k=\|A\|^2|A|^2|B|(2L+1)$, $l=k+2\|A\|^3|A||B|(\|A\|^2|B|K+1)$ and finally, $m=l+2\|A\|^3|B|$.

We shall show that P is nonvoid if and only if it contains a tree of depth less than m. It is obvious if $K=0$. Therefore let $K\neq 0$ and assume that p is an element of P with minimal rank. Let q and q' be different images of p under $\tau_{\mathbf{A}}$.

Assume to the contrary $\mathrm{dp}(p) \geqq m$. Then there exist $a_0, a_0' \in A_0$, $p_0, \ldots, p_m \in \hat{T}_{F,1}$, $p_{m+1} \in T_F$, $n_i, n_i' \geqq 0$ $(i=0, \ldots, m)$, $q_0 \in \hat{T}_{G,n_0}$, $q_0' \in \hat{T}_{G,n_0'}$, $\mathbf{q}_i \in \hat{T}_{G,n_i}^{n_i-1}$, $\mathbf{q}_i' \in \hat{T}_{G,n_i'}^{n_i'-1}$ $(i=1, \ldots, m)$, $\mathbf{q}_{m+1} \in T_G^{n_m}$, $\mathbf{q}_{m+1}' \in T_G^{n_m'}$, $\mathbf{a}_i \in A^{n_i}$, $\mathbf{a}_i' \in A^{n_i'}$ $(i=0, \ldots, m)$ such that the following three conditions are satisfied:

(1) $p = p_0(p_1(\ldots(p_{m+1})\ldots))$, $p_i \neq x_1$ $(i = 1, \ldots, m)$,

(2) $q = q_0(\mathbf{q}_1(\ldots(\mathbf{q}_{m+1})\ldots))$, $q' = q_0'(\mathbf{q}_1'(\ldots(\mathbf{q}_{m+1}')\ldots))$,

(3) $a_0 p_0 \overset{*}{\Rightarrow} q_0(\mathbf{a}_0 \mathbf{x}_1^{n_0})$, $a_0' p_0 \overset{*}{\Rightarrow} q_0'(\mathbf{a}_0' \mathbf{x}_1^{n_0'})$,

$\mathbf{a}_i \mathbf{p}_{i+1}^{n_i} \overset{*}{\Rightarrow} \mathbf{q}_{i+1}(\mathbf{a}_{i+1} \mathbf{x}_1^{n_{i+1}})$, $\mathbf{a}_i' \mathbf{p}_{i+1}^{n_i'} \overset{*}{\Rightarrow} \mathbf{q}_{i+1}'(\mathbf{a}_{i+1}' \mathbf{x}_1^{n_{i+1}'})$ $(i = 0, \ldots, m-1)$,

$\mathbf{a}_m \mathbf{p}_{m+1}^{n_m} \overset{*}{\Rightarrow} \mathbf{q}_{m+1}$, $\mathbf{a}_m' \mathbf{p}_{m+1}^{n_m'} \overset{*}{\Rightarrow} \mathbf{q}_{m+1}'$.

Further on we shall often use the following notations. Let $i \in \{0, \ldots, m+1\}$, $j \in \{0, \ldots, m\}$. Then $\check{p}_i = p_0(p_1(\ldots(p_i)\ldots))$, $\check{q}_i = q_0(\mathbf{q}_1(\ldots(\mathbf{q}_i)\ldots))$, $\check{q}_i' = q_0'(\mathbf{q}_1'(\ldots(\mathbf{q}_i')\ldots))$. Similarly, $\hat{p}_j = \mathbf{p}_{j+1}(\ldots(\mathbf{p}_{m+1})\ldots)$ $\hat{\mathbf{q}}_j = \mathbf{q}_{j+1}(\ldots(\mathbf{q}_{m+1})\ldots)$, $\hat{\mathbf{q}}_j' = \mathbf{q}_{j+1}'(\ldots(\mathbf{q}_{m+1}')\ldots)$. Furthermore, for each $i=0, \ldots, m$, A_i and A_i' denotes the set of all components of $\mathbf{a}_i$ and $\mathbf{a}_i'$, respectively.

If for any $\mathbf{v} \in T_G^{n_i}$ and $\mathbf{v}' \in T_G^{n_i'}$ we have $\check{q}_i(\mathbf{v}) \neq \check{q}_i'(\mathbf{v}')$ then, by Lemma 3 and the fact that the cardinality of the set $\{l, \ldots, m\}$ is at least $\|A\|^2|B|+1$, we get that for some i, j $(l \leqq i < j \leqq m)$ $\check{p}_i(\hat{p}_j) \in P$. It is a contradiction.

Therefore we may assume that $n_l>0$ and the existence of an index $i_l\in[n_l]$ such that there are trees $u'\in\hat{T}_{G,1}$, $v'\in T_G$ with $q'=u'(v')$, $\mathrm{path}(u')=\mathrm{path}_{i_l}(\check{q}_l)$ and $v'\neq\hat{q}_{l,i_l}$. Obviously, $n_i>0$ holds for each $i<l$. Now let i_j $(0\leqq i<l,\ j\in[n_i])$ be those uniquely determined idices for which $\mathrm{path}_{i_j}(\check{q}_i)$ is a prefix of $\mathrm{path}_{i_l}(\check{q}_l)$. Of course we may assume that $i_0=\ldots=i_l=1$.

Suppose now that there is no $\alpha'\in\mathrm{path}(\check{q}'_l)$ such that $\mathrm{path}_1(\check{q}_l)$ is a prefix of α' or conversely. In this case let

$B_i=\{a_{i,j}\mid \mathrm{path}_1(\check{q}_l)$ is a prefix of $\mathrm{path}_j(\check{q}_i)\}$,

$C_i=\{a_{i,j}\mid \mathrm{path}_1(\check{q}_l)$ is not a prefix of $\mathrm{path}_j(\check{q}_i)\}$

for each i $(l\leqq i\leqq m)$. Since the cardinality of the set $\{l,\ldots,m\}$ is exactly $2\|A\|^3|B|+1$ there exist indices i_1, i_2, i_3 $(l\leqq i_1<i_2<i_3\leqq m)$ satisfying the following conditions:

$$(\hat{p}_{i_1})_B=(\hat{p}_{i_2})_B=(\hat{p}_{i_3})_B,\quad B_{i_1}=B_{i_2}\subseteqq B_{i_3},\quad C_{i_1}\subseteqq C_{i_2}\subseteqq C_{i_3},\quad A'_{i_1}\subseteqq A'_{i_2}\subseteqq A'_{i_3}.$$

By Lemma 6 this yields that at least one of the trees $\check{p}_{i_1}(\hat{p}_{i_2})$, $\check{p}_{i_2}(\hat{p}_{i_3})$, $\check{p}_{i_1}(\hat{p}_{i_3})$ is in P, which is a contradiction.

We have shown that there exists an $\alpha'\in\mathrm{path}(\check{q}'_l)$ such that $\mathrm{path}_1(\check{q}_l)$ is a prefix of α' or conversely. Consequently $n'_i>0$ holds for each i $(0\leqq i\leqq l)$ and there exist integers $i_0,\ldots,i_l$ with the property that $\mathrm{path}_{i_j}(\check{q}'_j)$ is a prefix of $\mathrm{path}_1(\check{q}_l)$ or conversely $(j=0,\ldots,l)$. We may also assume that if $j_1<j_2$ then $\mathrm{path}_{i_{j_1}}(\check{q}'_{j_1})$ is a prefix of $\mathrm{path}_{i_{j_2}}(\check{q}'_{j_2})$, moreover, we may assume that $i_0=\ldots=i_l=1$. In this way either $\mathrm{path}_1(\check{q}_j)$ is a prefix of $\mathrm{path}_1(\check{q}'_j)$ $(j=0,\ldots,l)$ or conversely.

Now there are two cases. First suppose that $\mathrm{path}_1(\check{q}'_k)$ is a prefix of $\mathrm{path}_1(\check{q}_l)$. If, within this case, there exists an integer i $(0\leqq i\leqq k)$ such that $\|\mathrm{path}_1(\check{q}_i)|-|\mathrm{path}_1(\check{q}'_i)\|>\|A\|^2|B|K$ then, by Lemma 4, there is a tree $r\in T_F$ satisfying both $\check{p}_i(r)\in P$ and $\mathrm{rn}(r)<\mathrm{rn}(\hat{p}_i)$. This is a contradiction because $\mathrm{rn}(r)<\mathrm{rn}(\hat{p}_i)$ implies $\mathrm{rn}(\check{p}_i(r))<\mathrm{rn}(p)$. Thus we have $\|\mathrm{path}_1(\check{q}_i)|-|\mathrm{path}_1(\check{q}'_i)\|\leqq\|A\|^2|B|K$ for every i $(0\leqq i\leqq k)$. But this yields another contradiction. Indeed, the cardinality of the set $\{0,\ldots,k\}$ is equal to $\|A\|^2|A|^2|B|(2L+1)+1$, thus, there are at least two indices i, j $(0\leqq i<j\leqq k)$ such that — say — $\mathrm{path}_1(\check{q}_i)$ is a prefix of $\mathrm{path}_1(\check{q}'_i)$, $\mathrm{path}_1(\check{q}_j)$ is a prefix of $\mathrm{path}_1(\check{q}'_j)$, $\mathrm{path}_1(\check{q}'_i)/\mathrm{path}_1(\check{q}_i)=\mathrm{path}_1(\check{q}'_j)/\mathrm{path}_1(\check{q}_j)$, moreover, $(\hat{p}_i)_B=(\hat{p}_j)_B$, $a_{i,1}=a_{j,1}$, $a'_{i,1}=a'_{j,1}$, $B_i\subseteqq B_j$, $B'_i\subseteqq B'_j$ where $B_s=\{a_{s,t}\mid 2\leqq t\leqq n_s\}$, $B'_s=\{a'_{s,t}\mid 2\leqq t\leqq n'_s\}$ $(s=i, j)$. By an application of Lemma 7 this results that $\check{p}_i(\hat{p}_j)\in P$ — contrary to the minimality of p.

We have shown that $\mathrm{path}_1(\check{q}'_k)$ can not be a prefix of $\mathrm{path}_1(\check{q}_l)$. Therefore $\mathrm{path}_1(\check{q}_l)$ is a prefix of $\mathrm{path}_1(\check{q}'_k)$. If we prove that $|\mathrm{path}_1(\check{q}_l)|-|\mathrm{path}_1(\check{q}_k)|>\|A\|^2|B|K$ then also $|\mathrm{path}_1(\check{q}'_k)|-|\mathrm{path}_1(\check{q}_k)|>\|A\|^2|B|K$. Again by Lemma 4, this yields a contradiction. Therefore it is enough to show that $|\mathrm{path}_1(\check{q}_l)|-|\mathrm{path}_1(\check{q}_k)|>\|A\|^2|B|K$.

Assume that this condition does not hold. The cardinality of the set $\{k+1,\ldots,l\}$ is exactly $2\|A\|^3|A||B|(\|A\|^2|B|K+1)$, therefore, there exist indices i_1, i_2 $(k\leqq i_1<i_2\leqq l)$ such that $i_2-i_1=2\|A\|^3|A||B|$ and $\mathrm{path}_1(\check{q}_{i_1})=\ldots=\mathrm{path}_1(\check{q}_{i_2})$, i.e. $q_{i_1+1,1}=\ldots=q_{i_2,1}=x_1$. Now let

$B_j=\{a'_{j,t}\mid 1\leqq t\leqq n'_j,\ \mathrm{path}_1(\check{q}_{i_1})$ is a prefix of $\mathrm{path}_t(\check{q}'_j)\}$,

$C_j=\{a'_{j,t}\mid 1\leqq t\leqq n'_j,\ \mathrm{path}_1(\check{q}_{i_1})$ is not a prefix of $\mathrm{path}_t(\check{q}'_j)\}$

for each $j(i_1\leqq j\leqq i_2)$. Since the cardinality of $\{i_1,\ldots,i_2\}$ is equal to $2\|A\|^3|A||B|+1$ there exist indices j_1, j_2, j_3 $(i_1\leqq j_1<j_2<j_3\leqq i_2)$ such that each of the following

conditions is satisfied: $(\hat{p}_{j_1})_{\mathbf{B}}=(\hat{p}_{j_2})_{\mathbf{B}}=(\hat{p}_{j_3})_{\mathbf{B}}$, $\bar{A}_{j_1}\subseteq\bar{A}_{j_2}\subseteq\bar{A}_{j_3}$, $B_{j_1}=B_{j_2}\subseteq B_{j_3}$, $C_{j_1}\subseteq C_{j_2}\subseteq C_{j_3}$, $a_{j_1,1}=a_{j_2,1}=a_{j_3,1}$, where $\bar{A}_{j_t}=\{a_{j_t,s}\mid 2\leqq s\leqq n_t\}$. Thus, applying Lemma 5, we get that one of the trees $\check{p}_{j_1}(\hat{p}_{j_2})$, $\check{p}_{j_2}(\hat{p}_{j_3})$, $\check{p}_{j_1}(\hat{p}_{j_3})$ is in P, contradicting to the minimality of p. This ends the proof of Theorem 8.

Observe that, by the decomposition result for top-down tree transducers with regular look-ahead in [6], the above theorem holds for this type of transducers as well. But Theorem 8 has some other important consequences, too.

Take two arbitrary top-down or bottom-up tree transducers $\mathbf{A}=(F, A, G, A_0, \Sigma)$ and $\mathbf{B}=(F, B, G, B_0, \Sigma')$. Assume that $\mathbf{A}$ is functional and A and B are disjoint. Then construct the sum of $\mathbf{A}$ and $\mathbf{B}$, i.e. take $\mathbf{C}=(F, A\cup B, G, A_0\cup B_0, \Sigma\cup\Sigma')$. For $\mathbf{C}$ we have the following equivalence: $\tau_{\mathbf{A}}=\tau_{\mathbf{B}}$ if and only if $\operatorname{dom}\tau_{\mathbf{A}}=\operatorname{dom}\tau_{\mathbf{B}}$ and $\mathbf{C}$ is functional. From this and by the fact that the equality of regular forests is decidable we get:

Theorem 9. There exists an algorithm to decide for an arbitrary tree transducer $\mathbf{A}$ and a functional transducer $\mathbf{B}$ whether they are equivalent, i.e. such that $\tau_{\mathbf{A}}=\tau_{\mathbf{B}}$.

COROLLARY. A similar argument shows that Theorem 9 holds even if $\tau_{\mathbf{A}}=\tau_{\mathbf{B}}$ is replaced by $\tau_{\mathbf{A}}\subseteq\tau_{\mathbf{B}}$. On the other hand every deterministic transducer is functional. Thus, the equivalence problem for deterministic transducers is decidable.

Another consequence of Theorem 8 concerns with minimization of transducers. For any given tree transducer $\mathbf{A}$ one can compute a bound k with the following property: $\mathbf{A}$ has a corresponding tree transducer $\mathbf{B}$ which is minimal and satisfies that each tree in the right hand side of a rule of $\mathbf{B}$ has depth not exceeding k. This k can be obtained as $2K\|A\|$ in the top-down case and as $2K|A|$ in the bottom-up case. (Here $|A|$, $\|A\|$ and K are determined as in the proof of Theorem 8.) Therefore, if we assume that $\mathbf{A}$ is functional and we want to minimize $\mathbf{A}$, it is enough to check only for a finite number of transducers whether they are equivalent to $\mathbf{A}$ or not. This proves

Theorem 10. The minimization of functional tree transducers is effectively solvable.

COROLLARY. As every deterministic tree transducer is functional the same statement holds for deterministic transducers.

This corollary as well as the positive decidability result concerning the equivalence problem for deterministic bottom-up transducers and a restricted class of deterministic top-down transducers was independently achieved by Z. ZACHAR in [12] too.

3. Minimization of deterministic transducers

Let $\mathscr{K}$ be a class of tree transducers. A transducer $\mathbf{A}\in\mathscr{K}$ is said to be *minimal* in $\mathscr{K}$ if there is no transducer $\mathbf{B}\in\mathscr{K}$ which is equivalent to $\mathbf{A}$ and has fewer states than $\mathbf{A}$. In the preceding section we have shown that if $\mathscr{K}$ is the class of all functional top-down or all bottom-up transducers, or if $\mathscr{K}$ is the class of all deterministic top-down or all bottom-up transducers, then, for every given $\mathbf{A}\in\mathscr{K}$, one can effectively find a minimal equivalent transducer $\mathbf{B}\in\mathscr{K}$. However, these minimal realiza-

tions are not uniquely determined. In this section we investigate conditions assuring the uniqueness (up to isomorphism) of minimal realizations. Similar results are already known for Mealy-type automata (cf. [9]) and tree automata [1, 3, 10]. We point out that the minimizing process of Mealy-type automata can be generalized in a natural way for certain classes of deterministic tree transducers. For the sake of simplicity we shall consider *completely defined* deterministic tree transducers only. Therefore, from now on, by a tree transducer we shall always mean a completely defined deterministic transducer. Furthermore, all transducers will be taken with a fixed input type F and output type G. Since the case $F=F_0$ is trivial we assume that $F\neq F_0$.

First we treat top-down transducers. Let $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma)$ be a top-down transducer. It is *completely defined,* i.e. for any $a\in A$ and $f\in F$ there is a rule in Σ with left side af. Let $\mathbf{B}=(F, B, G, \{b_0\}, \Sigma')$ be another top-down transducer and take a mapping $\varphi: A\to B$. If the following two conditions are satisfied for arbitrary $n, m\geqq 0$, $f\in F_n, p\in T_{G,m}$, $a, a_1, \ldots, a_m\in A$ and $i_1, \ldots, i_m\in[n]$ then φ is called a *homomorphism* of $\mathbf{A}$ into $\mathbf{B}$:

(i) if $af\to p(a_1x_{i_1}, \ldots, a_mx_{i_m})\in\Sigma$ then $bf\to p(b_1x_{i_1}, \ldots, b_mx_{i_m})\in\Sigma'$ where $b=\varphi(a), b_j=\varphi(a_j)$ $(j\in[m])$,

(ii) $\varphi(a_0)=b_0$.

If, moreover, φ is surjective then $\mathbf{B}$ is a *homomorphic image* of $\mathbf{A}$. If φ is bijective then we speak about *isomorphism,* written $\mathbf{A}\cong\mathbf{B}$. If $B\subseteq A$ and φ is the natural embedding of B into A then $\mathbf{B}$ is a *subtransducer* of $\mathbf{A}$. If $\mathbf{A}$ has not proper subtransducers then it is called *connected.*

The next statement is obvious:

Statement 11. If there is a homomorphism from $\mathbf{A}$ into $\mathbf{B}$ then $\tau_{\mathbf{A}}=\tau_{\mathbf{B}}$.

As in case of universal algebras there is a bijective correspondence between homomorphic images and congruence relations. Let $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma)$ be an arbitrary top-down transducer and take an equivalence relation θ on $\mathbf{A}$. It is called a *congruence relation* if for any two rules $af\to p(a_1x_{i_1}, \ldots, a_mx_{i_m})$, $bf\to q(b_1x_{j_1}, \ldots, b_lx_{j_l})\in\Sigma$ $(n, m, l\geqq 0$, $f\in F_n, p\in\tilde{T}_{G,m}$, $q\in\tilde{T}_{G,l}$, $i_1, \ldots, i_m$, $j_1, \ldots, j_l\in[n]$, $a_1, \ldots, a_m, b_1, \ldots, b_l, a, b\in A)$ $a\theta b$ implies $m=l, p=q, i_t=j_t$ and $a_t\theta b_t$ $(t=1, \ldots, m)$. Here for any nonnegative integer n the notation $\tilde{T}_{G,n}$ is used to denote the set $\tilde{T}_{G,n}=\{p\in T_{G,n} \mid \mathrm{fr}\,(p)=x_1\ldots x_n\}$.

Assume that θ is a congruence relation of $\mathbf{A}$. Then we can define the *quotient of* $\mathbf{A}$ induced by θ. This is the top-down transducer $\mathbf{A}/\theta=(F, A/\theta, G, \{\theta(a_0)\}, \Sigma')$ where for every $n, m\geqq 0$, $f\in F_n, p\in T_{G,m}$, $a, a_1, \ldots, a_m\in A$

$$\theta(a)f\to p(\theta(a_1)x_{i_1}, \ldots, \theta(a_m)x_{i_m})\in\Sigma'$$

if and only if

$$af\to p(a_1x_{i_1}, \ldots, a_mx_{i_m})\in\Sigma.$$

Statement 12. $\mathbf{A}/\theta$ is a homomorphic image of A. If $\mathbf{B}$ is a homomorphic image of $\mathbf{A}$ then there is a congruence relation θ of $\mathbf{A}$ such that $\mathbf{A}/\theta\cong\mathbf{B}$.

Take again the top-down tree transducer $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma)$. Let us define an equivalence relation $\theta_{\mathbf{A}}$ on A: $a\theta_{\mathbf{A}}b$ if and only if $\tau_{\mathbf{A}(a)}=\tau_{\mathbf{A}(b)}$. Unfortunately, this will not always be a congruence relation. We need certain additional requirements on $\mathbf{A}$.

Let ϱ be any mapping of the set of nonnegative integers into itself, i.e. $\varrho: \omega \to \omega$. Then let $\mathscr{K}(\varrho)$ denote the class of all top-down tree transducers $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma)$ which satisfy the condition $|\mathrm{path}_j(p)|=\varrho(i_j)$ for every $n, m \geqq 0$, $f \in F_n$, $p \in \tilde{T}_{G,m}$, $a, a_1, \ldots, a_m \in A$, $x_{i_1}, \ldots, x_{i_m} \in X_n$, $j \in [m]$ and $af \to p(a_1 x_{i_1}, \ldots, a_m x_{i_m}) \in \Sigma$, as well as the condition $|\tau_{\mathbf{A}(a)}(T_F)|>1$ for arbitrary state a appearing in the right side of a rule in Σ.

Statement 13. If $\mathbf{A} \in \mathscr{K}(\varrho)$ then $\theta_{\mathbf{A}}$ is a congruence relation.

Proof. Let $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma)$ and assume that $af \to p(a_1 x_{i_1}, \ldots, a_m x_{i_m})$ and $bf \to q(b_1 x_{j_1}, \ldots, b_l x_{j_l})$ are rules in Σ where $a, b \in A$, $a \theta_{\mathbf{A}} b$, $n, m, l \geqq 0$, $f \in F_n$, $p \in \tilde{T}_{G,m}$, $q \in \tilde{T}_{G,l}$, $a_1, \ldots, a_m, b_1, \ldots, b_l \in A$, $i_1, \ldots, i_m$, $j_1, \ldots, j_l \in [n]$. Assume that there is an integer $t \in [m]$ such that none of the strings in $\bigcup (\mathrm{path}_s(q) \mid i_t = j_s, s \in [l])$ is a prefix of $\mathrm{path}_t(p)$ or conversely. Then, by $|\tau_{\mathbf{A}(a_t)}(T_F)|>1$, it is easy to show the existence of a tree $r \in T_F$ with $\tau_{\mathbf{A}(a)}(r) \neq \tau_{\mathbf{A}(b)}(r)$. On the other hand if $i_t = j_s$ holds for some $t \in [m]$ and $s \in [l]$ then the equality $|\mathrm{path}_t(p)| = |\mathrm{path}_s(q)|$ is also valid. This proves that $m=l$, $i_t = j_t$, $\mathrm{path}_t(p) = \mathrm{path}_t(q)$ $(t=1, \ldots, m)$. But $\tau_{\mathbf{A}(a)} = \tau_{\mathbf{B}(b)}$, hence from this we get $p=q$, $a_t \theta_{\mathbf{A}} b_t$ $(t=1, \ldots, m)$.

Another class of top-down transducers in which $\theta_{\mathbf{A}}$ is always a congruence relation is the class $\mathscr{K}_d$, where d denotes an arbitrary nonnegative integer. A top-down transducer $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma)$ is in $\mathscr{K}_d$ if and only if for every $a \in A$, $f \in F_0$ and $p \in T_G$ if $af \to p \in \Sigma$ then $\mathrm{dp}(p)=d$, moreover, as in case of $\mathscr{K}(\varrho)$, $|\tau_{\mathbf{A}(a)}(T_F)|>1$ is satisfied for each $a \in A$ appearing in the right side of a rule in Σ.

Statement 14. If $\mathbf{A} \in \mathscr{K}_d$ then $\theta_{\mathbf{A}}$ is a congruence relation.

Proof. The proof of this statement is similar to that of Statement 13. Only use the conditions defining $\mathscr{K}_d$ to establish the bijective correspondence between the sets $\bigcup (\mathrm{path}_t(p) \mid t \in [m])$ and $\bigcup (\mathrm{path}_s(q) \mid s \in [l])$ for the rules $af \to p(a_1 x_{i_1}, \ldots, a_m x_{i_m})$ and $bf \to q(b_1 x_{j_1}, \ldots, b_l x_{j_l})$.

Note that for $\mathbf{A} \in \mathscr{K}(\varrho)$ or $\mathbf{A} \in \mathscr{K}_d$ the definition of $\theta_{\mathbf{A}}$ can be reformulated as follows. Let $a, b \in A$. Then $a \theta_{\mathbf{A}} b$ if and only if for every $n, m \geqq 0$, $p \in T_{F,n}$, $q \in \tilde{T}_{G,m}$ and $i_1, \ldots, i_m \in [n]$ the following equivalence holds:

$$\exists a_1, \ldots, a_m \in A \quad ap \overset{*}{\Rightarrow} q(a_1 x_{i_1}, \ldots, a_m x_{i_m})$$

if and only if

$$\exists b_1, \ldots, b_m \in A \quad bp \overset{*}{\Rightarrow} q(b_1 x_{i_1}, \ldots, b_m x_{i_m}).$$

This is an easy consequence of statements 13, 14. Observe that this new definition of $\theta_{\mathbf{A}}$ makes $\theta_{\mathbf{A}}$ a congruence relation without requiring $\mathbf{A} \in \mathscr{K}(\varrho)$ or $\mathbf{A} \in \mathscr{K}_d$.

A transducer $\mathbf{A} \in \mathscr{K}(\varrho)$ or $\mathbf{A} \in \mathscr{K}_d$ is called *reduced* if $\theta_{\mathbf{A}}$ is the equality relation. As both $\mathscr{K}(\varrho)$ and $\mathscr{K}_d$ are closed under homomorphic images the transducer $\mathbf{A}/\theta_{\mathbf{A}}$ is reduced for any $\mathbf{A} \in \mathscr{K}(\varrho)$ or $\mathbf{A} \in \mathscr{K}_d$. The following statement is the basic step to show that minimal transducers in $\mathscr{K}(\varrho)$ and $\mathscr{K}_d$ are exactly the connected and reduced transducers.

Theorem 15. Let $\mathbf{A}, \mathbf{B} \in \mathscr{K}(\varrho)$ be connected top-down transducers. Then $\mathbf{A}$ and $\mathbf{B}$ are equivalent if and only if $\mathbf{A}/\theta_{\mathbf{A}} \cong \mathbf{B}/\theta_{\mathbf{B}}$. The same holds for $\mathscr{K}_d$.

Proof. Sufficiency follows by statements 11—14. In order to prove necessity first observe that if $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma)$ and $\mathbf{B}=(F, B, G, \{b_0\}, \Sigma')$, moreover,

$a_0 p \overset{*}{\Rightarrow}_{\mathbf{A}} q(a_1 x_{i_1}, \ldots, a_m x_{i_m})$ — where $p \in T_{F,n}$, $n \geqq 0$, $q \in \hat{T}_{G,m}$, $m \geqq 0$, $a_1, \ldots, a_m \in A$, $i_1, \ldots, i_m \in [n]$ — then there exist states $b_1, \ldots, b_m \in B$ with $b_0 p \overset{*}{\Rightarrow}_{\mathbf{B}} q(b_1 x_{i_1}, \ldots, b_m x_{i_m})$. Furthermore, for these states b_i $(i=1, \ldots, m)$ we have $\tau_{\mathbf{A}(a_i)} = \tau_{\mathbf{B}(b_i)}$. This is a consequence of the assumption $\tau_{\mathbf{A}} = \tau_{\mathbf{B}}$ and the definitions of $\mathscr{K}(\varrho)$ and $\mathscr{K}_d$. Using the above mentioned facts it is easy to prove that the correspondence φ: $A/\theta_{\mathbf{A}} \to B/\theta_{\mathbf{B}}$ defined by $\varphi(\theta_{\mathbf{A}}(a)) = \theta_{\mathbf{B}}(b)$ if and only if there exist $p \in \hat{T}_{F,1}$, $q \in \hat{T}_{G,m+1}$ $(m \geqq 0)$, $a_1, \ldots, a_m \in A$, $b_1, \ldots, b_m \in B$ such that $a_0 p \overset{*}{\Rightarrow}_{\mathbf{A}} q(a x_1, a_1 x_1, \ldots, a_m x_1)$ and $b_0 p \overset{*}{\Rightarrow}_{\mathbf{B}} q(b x_1, b_1 x_1, \ldots, b_m x_1)$ forms an isomorphism of $\mathbf{A}/\theta_{\mathbf{A}}$ into $\mathbf{B}/\theta_{\mathbf{B}}$.

The next theorem is an immediate consequence of Theorem 15 and the fact that $\mathscr{K}(\varrho)$ and $\mathscr{K}_d$ are closed under the formation of subtransducers and homomorphic images.

Theorem 16. A transducer is minimal in $\mathscr{K}(\varrho)$ if and only if it is connected and reduced. If both $\mathbf{A}$ and $\mathbf{B}$ are minimal in $\mathscr{K}(\varrho)$ and they are equivalent then $\mathbf{A} \cong \mathbf{B}$, i.e. the minimal realization of a transducer in $\mathscr{K}(\varrho)$ is unique up to isomorphism. The same holds for the class $\mathscr{K}_d$.

Of course Theorem 16 holds for every class $\mathscr{K} \subseteqq \mathscr{K}(\varrho)$ or $\mathscr{K} \subseteqq \mathscr{K}_d$ provided $\mathscr{K}$ is closed under the formation of subtransducers and homomorphic images. The most important example for a class of this type is the class of all *top-down relabelings* (cf. [5]).

It is natural to raise the question whether the minimal transducers in $\mathscr{K}(\varrho)$ or $\mathscr{K}_d$ are minimal in the class of all top-down transducers. The following examples prove that the answer is negative in general. In these examples the adjectives "linear", "nondeleting" are used in the sense of [5]. Furthermore, a top-down tree transducer $\mathbf{A} = (F, A, G, \{a_0\}, \Sigma)$ will be called uniform if each rule $af \to p$ $(a \in A, f \in F_n\ (n \geqq 0), p \in T_{G, A \times X_n})$ can be written as $af \to q(a_1 x_1, \ldots, a_n x_n)$ for a tree $q \in T_{G,n}$ and states $a_1, \ldots, a_n \in A$.

Example 17. This example shows that there is a linear nondeleting top-down tree transducer $\mathbf{A} \in \mathscr{K}_1 \cap \mathscr{K}(\varrho)$ which is connected and reduced — i.e. minimal in both $\mathscr{K}_1$ and $\mathscr{K}(\varrho)$ — but which is not minimal in the class of all linear nondeleting top-down tree transducers. Here ϱ: $\omega \to \omega$ is the mapping defined by $\varrho(n) = 1$ $(n \geqq 0)$. Indeed, let $\mathbf{A} = (F, [5], F, [1], \Sigma)$ where F is the type determined by the conditions $F_0 = \{\#\}$, $F_1 = \{f, g\}$, $F_n = \emptyset$ if $n > 1$ and Σ consists of the rules (1)—(5) listed below:

(1) $1\# \to f(\#)$, $1f(x_1) \to f(2x_1)$, $1g(x_1) \to g(3x_1)$,

(2) $2\# \to f(\#)$, $2f(x_1) \to f(4x_1)$, $2g(x_1) \to f(4x_1)$,

(3) $3\# \to g(\#)$, $3f(x_1) \to g(4x_1)$, $3g(x_1) \to g(4x_1)$,

(4) $4\# \to f(\#)$, $4f(x_1) \to f(5x_1)$, $4g(x_1) \to g(5x_1)$,

(5) $5\# \to f(\#)$, $5f(x_1) \to f(1x_1)$, $5g(x_1) \to g(1x_1)$.

However, $\mathbf{A}$ is equivalent to $\mathbf{A}' = (F, [4], F, [1], \Sigma')$ where Σ' contains the following rules (1)—(4):

(1) $1\# \to f(\#)$, $1f(x_1) \to f(f(2x_1))$, $1g(x_1) \to g(g(2x_1))$,

(2) $2\# \to \#$, $2f(x_1) \to 3x_1$, $2g(x_1) \to 3x_1$,

(3) $3\# \to f(\#)$, $3f(x_1) \to f(4x_1)$, $3g(x_1) \to g(4x_1)$,

(4) $4\# \to f(\#)$, $4f(x_1) \to f(1x_1)$, $4g(x_1) \to g(1x_1)$.

Example 18. This example proves that there is a top-down tree transducer $\mathbf{A} \in \mathscr{K}_0$ which is minimal in $\mathscr{K}_0$ but not minimal in the class of all top-down transducers.

Let us define the types F and G by $F_0=\{\#\}$, $F_1=\{f\}$, $F_n=\emptyset$ if $n>1$ and $G_0=\{\#, \#_1, \#_2\}$, $G_1=\{f\}$, $G_2=\{g\}$, $G_n=\emptyset$ $(n>2)$, respectively. Then put $\mathbf{A}=(F, [4], G, [1], \Sigma)$ where Σ consists of the following rules:

(1) $1\# \to \#$, $1f(x_1) \to g(2x_1, 3x_1)$,

(2) $2\# \to \#_1$, $2f(x_1) \to f(4x_1)$,

(3) $3\# \to \#_2$, $3f(x_1) \to f(4x_1)$,

(4) $4\# \to \#$, $4f(x_1) \to f(4x_1)$.

It is easy to check that $\mathbf{A}$ is minimal in $\mathscr{K}_0$. On the other hand $\mathbf{A}$ is equivalent to $\mathbf{A}'=(F, [3], G, [1], \Sigma')$ with Σ' containing the following rules:

(1) $1\# \to \#$, $1f(x_1) \to 2x_1$,

(2) $2\# \to g(\#_1, \#_2)$, $2f(x_1) \to g(f(3x_1), f(3x_1))$,

(3) $3\# \to \#$, $3f(x_1) \to f(3x_1)$.

Observe that $\mathbf{A}$ was not uniform.

In spite of Example 18 we have

Theorem 19. If a uniform transducer is minimal in $\mathscr{K}_0$ then it is minimal in the class of all top-down tree transducers.

Proof. Let $\mathbf{A}=(F, A, G, \{a_0\}, \Sigma) \in \mathscr{K}_0$ be uniform and minimal in $\mathscr{K}_0$. Assume that the top-down tree transducer $\mathbf{B}=(F, B, G, \{b_0\}, \Sigma')$ is equivalent to $\mathbf{A}$ and has fewer states than $\mathbf{A}$, i.e. $|B|<|A|$.

Take an arbitrary state $a \in A$. We shall correspond to this state a state $\varphi(a) \in B$ as follows. First let us choose the trees $p \in \tilde{T}_{F,1}$ and $q \in \tilde{T}_{G,n}$ $(n>0)$ in such a way that we have $a_0 p \overset{*}{\Rightarrow}_{\mathbf{A}} q(\mathbf{a}^n \mathbf{x}_1^n)$. If $a=a_0$ choose $p=q=x_1$. This can be done since $\mathbf{A}$ is connected. Let $r \in \tilde{T}_{G,m}$ $(m \geqq 0)$ and $b_1, \ldots, b_m \in B$ be determined by $b_0 p \overset{*}{\Rightarrow}_{\mathbf{B}} r(b_1 x_1, \ldots, b_m x_1)$. As $|\tau_{\mathbf{A}(c)}(T_F)|>1$ is satisfied for each $c \in A$ occuring in the right side of a rule in Σ we must have $m>0$. Or even, there must be an index $j_i \in [m]$ for each $i \in [n]$ with the property that either $\mathrm{path}_{j_i}(r)$ is a prefix of $\mathrm{path}_i(q)$ or conversely. But, by the definition of $\mathscr{K}_0$, it is impossible that $\mathrm{path}_i(q)$ is a proper prefix of $\mathrm{path}_j(r)$. Therefore j_i is uniquely determined for each $i \in [n]$ and $\mathrm{path}_{j_i}(r)$ is a prefix of $\mathrm{path}_i(q)$. As $\mathbf{A}$ and $\mathbf{B}$ are equivalent this implies that there exist trees $r_1, \ldots, r_m \in T_{G,1}$ with $r(r_1, \ldots, r_m)=q$. Let $\varphi(a)=b_1$ and $r_a=r_{j_1}$. We must have $r_a(\tau_{\mathbf{A}(a)}(t))=\tau_{\mathbf{B}(\varphi(a))}(t)$ for each $t \in T_F$, i.e. $r_a(\tau_{\mathbf{A}(a)})=\tau_{\mathbf{B}(\varphi(a))}$.

As $|B|<|A|$ there exist states $a_1 \neq a_2 \in A$ with $\varphi(a_1)=\varphi(a_2)$. Consequently, $r_{a_1}(\tau_{\mathbf{A}(a_1)})=r_{a_2}(\tau_{\mathbf{A}(a_2)})$. But, again by the definition of $\mathscr{K}_0$, this is possible only if $r_{a_1}=r_{a_2}$ and $\tau_{\mathbf{A}(a_1)}=\tau_{\mathbf{A}(a_2)}$ yielding a contradiction.

We will now turn our attention to the bottom-up case. A deterministic bottom-pu tree transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)$ is called *completely defined* if there is a rule in Σ with left hand side $f(a_1x_1, \ldots, a_nx_n)$ for every $n\geqq 0$, $f\in F_n$ and $a_1, \ldots, a_n\in A$. First of all we have to define homomorphisms, congruence relations etc.

Let $\mathbf{A}=(F, A, G, A_0, \Sigma)$ and $\mathbf{B}=(F, B, G, B_0, \Sigma')$ be bottom-up transducers. By a *homomorphism* of $\mathbf{A}$ into $\mathbf{B}$ we mean a mapping $\varphi: A\to B$ which satisfies the following two conditions:

(i) $f(b_1x_1, \ldots, b_nx_n)\to bp\in\Sigma'$ if $f(a_1x_1, \ldots, a_nx_n)\to ap\in\Sigma$, $b_i=\varphi(a_i)$ $(i=1, \ldots, n)$, $b=\varphi(a)$ $(n\geqq 0,\ f\in F_n,\ a_1, \ldots, a_n, a\in A,\ p\in T_{G,n})$,

(ii) $\varphi(A_0)\subseteq B_0$, $\varphi^{-1}(B_0)\subseteq A_0$.

Again, if φ is surjective then $\mathbf{B}$ is a *homomorphic image* of $\mathbf{A}$ and bijective homomorphisms are called *isomorphisms*. If $B\subseteq A$ and φ is the natural embedding of B into A then $\mathbf{B}$ is a *substransducer* of $\mathbf{A}$.

We now define *congruence relations*. A congruence relation of $\mathbf{A}$ is an equivalence relation θ on A with the following property: for any $n\geqq 0$, $f\in F_n$, $a_i, b_i\in A$ $(i=1, \ldots, n)$, $a, b\in A$ and $p, q\in T_{G,n}$ if both $f(a_1x_1, \ldots, a_nx_n)\to ap$ and $f(b_1x_1, \ldots, b_nx_n)\to bq$ are in Σ and $a_i\theta b_i$ $(i=1, \ldots, n)$ are satisfied then $p=q$ and $a\theta b$ hold too. Furthermore, A_0 is required to be equal to the union of certain blocks of the partition induced by θ: $A_0=\bigcup(\theta(a)\,|\,a\in A_0)$. The *quotient transducer* determined by θ is the transducer $\mathbf{A}/\theta=(F, A/\theta, G, A_0/\theta, \Sigma')$ where

$$\Sigma'=\{f(\theta(a_1)x_1, \ldots, \theta(a_n)x_n)\to\theta(a)p\,|\,f(a_1x_1, \ldots, a_nx_n)\to ap\in\Sigma\}.$$

With the above definitions in mind one can easily prove the analogues of statements 11 and 12.

For a bottom-up transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)$ the relation $\theta_{\mathbf{A}}$ is defined as follows. Let $a, b\in A$. Then $a\theta_{\mathbf{A}}b$ if and only if the equivalence $\exists a_0\in A_0\ p(a_1x_1, \ldots, a_{i-1}x_{i-1}, ax_i, a_{i+1}x_{i+1}, \ldots, a_nx_n)\overset{*}{\Rightarrow}a_0q\Leftrightarrow\exists b_0\in A_0\ p(a_1x_1, \ldots, a_{i-1}x_{i-1}, bx_i, a_{i+1}x_{i+1}, \ldots, a_nx_n)\overset{*}{\Rightarrow}b_0q$ holds for all $n>0$, $i\in[n]$, $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n\in A$, $p\in T_{F,n}$ (or equivalently $p\in\hat{T}_{F,n}$ or $p\in\tilde{T}_{F,n}$) and $q\in T_{G,n}$.

Likewise in the top-down case, $\theta_{\mathbf{A}}$ will not always be a congruence relation, but it will be a congruence relation if we require $\mathbf{A}$ to be in $\mathscr{K}(\varrho)$ for a mapping ϱ of the set of nonnegative integers into itself. A bottom-up transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)$ belongs to $\mathscr{K}(\varrho)$ provided it satisfies the following three conditions:

(i) if $f(a_1x_1, \ldots, a_nx_n)\to ap\in\Sigma$ $(n>0,\ f\in F_n, a, a_1, \ldots, a_n\in A,\ p\in T_{G,n})$ then $|w|=\varrho(i)$ holds for each $i\in[n]$ and $w\in\mathrm{path}_i(p)$,

(ii) $\mathbf{A}$ is nondeleting, i.e. for all $n>0$, $f\in F_n$, $a, a_1, \ldots, a_n\in A$ and $p\in T_{G,n}$ if $f(a_1x_1, \ldots, a_nx_n)\to ap\in\Sigma$ then each of the variables $x_1, \ldots, x_n$ occurs in $\mathrm{fr}(p)$,

(iii) for any $a\in A$ there exist $p\in\hat{T}_{F,n+1}$, $q\in T_{G,n+1}$ $(n\geqq 0)$, $a_0\in A_0$, $a_1, \ldots, a_n\in A$ such that $p(ax_1, a_1x_2, \ldots, a_nx_{n+1})\overset{*}{\Rightarrow}a_0q$.

Statement 20. If $\mathbf{A}\in\mathscr{K}(\varrho)$ then $\theta_{\mathbf{A}}$ is a congruence relation.

Proof. Let $\mathbf{A}=(F, A, G, A_0, \Sigma)$, $a, b\in A$. Assume that $a\theta_{\mathbf{A}} b$ and let

$$f(a_1x_1, \ldots, a_{i-1}x_{i-1}, ax_i, a_{i+1}x_{i+1}, \ldots, a_nx_n) \to cp,$$
$$f(a_1x_1, \ldots, a_{i-1}x_{i-1}, bx_i, a_{i+1}x_{i+1}, \ldots, a_nx_n) \to dq$$

be arbitrary rules in Σ. Here $n>0$, $i\in[n]$, $f\in F_n$, $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n, c, d\in A$, $p, q\in T_{G,n}$. We have to show that $p=q$ and $c\theta_{\mathbf{A}} d$.

As $\mathbf{A}\in\mathscr{K}(\varrho)$, there exist $m\geqq 0$, $c_1, \ldots, c_m\in A$, $a_0\in A_0$ $r\in\hat{T}_{F,m+1}$ and $s\in T_{G,m+1}$ such that

$$r(cx_1, c_1x_2, \ldots, c_mx_{m+1}) \overset{*}{\Rightarrow} a_0s.$$

Let $r_1=r(f(x_1, \ldots, x_n), x_{n+1}, \ldots, x_{n+m})$, $s_1=s(p, x_{n+1}, \ldots, x_{n+m})$. Of course we have

$$r_1(a_1x_1, \ldots, a_{i-1}x_{i-1}, ax_i, a_{i+1}x_{i+1}, \ldots, a_nx_n, c_1x_{n+1}, \ldots, c_mx_{n+m}) \overset{*}{\Rightarrow} a_0s_1.$$

Since $a\theta_{\mathbf{A}} b$, this implies

$$r_1(a_1x_1, \ldots, a_{i-1}x_{i-1}, bx_i, a_{i+1}x_{i+1}, \ldots, a_nx_n, c_1x_{n+1}, \ldots, c_mx_{n+m}) \overset{*}{\Rightarrow} b_0s_1$$

for a state $b_0\in A_0$. But this is possible only if s_1 is of form $s_1=t(q, x_{n+1}, \ldots, x_{n+m})$ where $t\in T_{G,m+1}$ and $r(dx_1, c_1x_2, \ldots, c_mx_{m+1})\overset{*}{\Rightarrow} b_0t$.

We know that $s(p, x_{n+1}, \ldots, x_{n+m})=t(q, x_{n+1}, \ldots, x_{n+m})$. By (i) and (ii) in the definition of $\mathscr{K}(\varrho)$ this results that $s=t$ and $p=q$. Essentially the same argument shows that $c\theta_{\mathbf{A}} d$.

Observe that for a bottom-up transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)\in\mathscr{K}(\varrho)$ the relation $\theta_{\mathbf{A}}$ can be redefined as follows. Let $a, b\in A$. Then $a\theta_{\mathbf{A}} b$ if and only if the following two equivalences are satisfied for arbitrary $p\in T_{F,n}$, $q\in T_{G,n}$ $(n\geqq 0)$, $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n\in A$ and $i\in[n]$:

(i) $\exists a_0\in A\ p(a_1x_1, \ldots, a_{i-1}x_{i-1}, ax_ia_{i+1}x_{i+1}, \ldots, a_nx_n) \overset{*}{\Rightarrow} a_0q$

if and only if

$$\exists b_0\in A\ p(a_1x_1, \ldots, a_{i-1}x_{i-1}, bx_i, a_{i+1}x_{i+1}, \ldots, a_nx_n) \overset{*}{\Rightarrow} b_0q,$$

(ii) for a_0 and b_0 of (i) it holds that $a_0\in A_0$ if and only if $b_0\in A_0$.

A transducer $\mathbf{A}\in\mathscr{K}(\varrho)$ is called *reduced* if $\theta_{\mathbf{A}}$ is the equality relation on A. $\mathbf{A}/\theta_{\mathbf{A}}$ is always reduced.

In contrast with the top-down case there are nonisomorphic but equivalent minimal transducers in $\mathscr{K}(\varrho)$. However, if a bottom-up transducer is minimal in $\mathscr{K}(\varrho)$ then it is both reduced and connected (i.e. it has not proper subtransducers). The converse is not true in general.

According to the above discussion we need some further restrictions to guarantee the uniqueness of minimal realizations. For this purpose we introduce the subclass $\mathscr{K}'(\varrho)$ of $\mathscr{K}(\varrho)$. A bottom-up transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)\in\mathscr{K}(\varrho)$ belongs to $\mathscr{K}'(\varrho)$ if and only if it satisfies the condition:

if $f(a_1x_1, \ldots, a_nx_n)\to ap\in\Sigma$ where $n>0$, $f\in F_n$, $a_1, \ldots, a_n$, $a\in A$ and $p\in T_{G,n}$ then $p\in\hat{T}_{G,n}$ and none of the operational symbols in G_0 occurs in p.

Now we are able to state an analogue of Theorem 15 for bottom-up transducers.

Theorem 21. Let $\mathbf{A}, \mathbf{B}\in\mathscr{K}'(\varrho)$ be connected. Then they are equivalent if and only if $\mathbf{A}/\theta_{\mathbf{A}}\cong\mathbf{B}/\theta_{\mathbf{B}}$.

Proof. The sufficiency follows in the same way as in Theorem 14. In order to prove the necessity of our statement, first observe that if $\mathbf{A}=(F, A, G, A_0, \Sigma)$ and $\mathbf{B}=(F, B, G, B_0, \Sigma')$, moreover, $\tau_{\mathbf{A}(a)}(p)=q$ where $p\in T_F$, $q\in T_G$ and $a\in A$, then there is a state $b\in B$ with $\tau_{\mathbf{B}(b)}(p)=q$. In fact, if $a_i\in A$, $b_i\in B$ $(i=1, \ldots, n,\ n>0)$ are such that $\operatorname{dom}\tau_{\mathbf{A}(a_i)}\cap\operatorname{dom}\tau_{\mathbf{B}(b_i)}\neq\emptyset$ $(i=1, \ldots, n)$ and $p(a_1x_1, \ldots, a_nx_n)\overset{*}{\Rightarrow}_{\mathbf{A}}aq$ where $p\in T_{F,n}$, $q\in T_{G,n}$ and $a\in A$ then there is a state $b\in B$ satisfying $p(b_1x_1, \ldots, b_nx_n)\overset{*}{\Rightarrow}_{\mathbf{B}}bq$. The same assertions holds if we change the role of $\mathbf{A}$ and $\mathbf{B}$. By these observations it is easy to verify that the correspondence φ defined by $\varphi(\theta_{\mathbf{A}}(a))=\theta_{\mathbf{B}}(b)$ if and only if $\operatorname{dom}\tau_{\mathbf{A}(a)}\cap\operatorname{dom}\tau_{\mathbf{B}(b)}\neq\emptyset$ is an isomorphism of $\mathbf{A}/\theta_{\mathbf{A}}$ into $\mathbf{B}/\theta_{\mathbf{B}}$.

Theorem 22. A bottom-up transducer is minimal in $\mathscr{K}'(\varrho)$ if and only if it is both reduced and connected. The minimal realization of a bottom-up transducer in $\mathscr{K}'(\varrho)$ is unique up to isomorphism.

Proof. Immediate by Theorem 21.

Observe that Theorem 22 holds for every class $\mathscr{K}\subseteq\mathscr{K}'(\varrho)$ provided it is closed under the formation of subtransducers and homomorphic images. An example of a class of this sort is the class of all bottom-up relabelings satisfying condition (iii) in the definition of $\mathscr{K}(\varrho)$. A tree transducer $\mathbf{A}=(F, A, G, A_0, \Sigma)$ is called a bottom-up relabeling if each rule in Σ is of form

$$f(a_1x_1, \ldots, a_nx_n)\to ag(x_1, \ldots, x_n)$$

where $n\geqq 0$, $f\in F_n$, $g\in G_n$, $a_1, \ldots, a_n, a\in A$.

The following example shows that there is a transducer which is minimal in $\mathscr{K}'(\varrho)$ but which is not minimal in the class of all bottom-up transducers. Let $F_0=\{\#\}$, $F_1=\{f, g\}$ and $F_i=\emptyset$ if $i>1$. Take the bottom-up transducer $\mathbf{A}=(F, [5], F, [1], \Sigma)$ where Σ consists of the following rules:

(1) $\#\to 1\#$,

(2) $f(1x_1)\to 2f(x_1)$, $g(1x_1)\to 3g(x_1)$,

(3) $f(2x_1)\to 4f(x_1)$, $g(2x_1)\to 4f(x_1)$,

(4) $f(3x_1)\to 4g(x_1)$, $g(3x_1)\to 4g(x_1)$,

(5) $f(4x_1)\to 5f(x_1)$, $g(4x_1)\to 4g(x_1)$,

(6) $f(5x_1)\to 1f(x_1)$, $g(5x_1)\to 1g(x_1)$.

It is easy to see that $\mathbf{A}$ is minimal in $\mathscr{K}'(\varrho)$ where ϱ is a constant mapping: $\varrho(n)=1$ for all $n\geqq 0$. On the other hand $\tau_{\mathbf{A}}$ can be induced by a four state transducer $\mathbf{B}=(F, [4], F, [1], \Sigma')$ where Σ' consists of the rules (1)—(5) listed below:

(1) $\#\to 1\#$,

(2) $f(1x_1)\to 2f(f(x_1))$, $g(1x_1)\to 2g(g(x_1))$,

(3) $f(2x_1)\to 3x_1$, $g(2x_1)\to 3x_1$,

(4) $f(3x_1)\to 4f(x_1)$, $g(3x_1)\to 4g(x_1)$,

(5) $f(4x_1)\to 1f(x_1)$, $g(4x_1)\to 1g(x_1)$.

In spite of the preceding example the following theorem is valid.

Theorem 23. Let $\mathbf{A}=(F, A, G, A_0, \Sigma)$ be minimal in $\mathscr{K}'(\varrho)$. Assume that $A=A_0$. Then $\mathbf{A}$ is minimal in the class of all bottom-up transducers.

Proof. Let us correspond to each $a\in A$ a tree $p_a\in\operatorname{dom}\tau_{\mathbf{A}(a)}$. This can be done because $\mathbf{A}$ is connected. Assume that $\mathbf{B}=(F, B, G, B_0, \Sigma')$ is equivalent to $\mathbf{A}$ and has fewer states than $\mathbf{A}$, i.e. $|B|<|A|$. Of course $B=B_0$. Define the mapping $\varphi: A\to B$ by $\varphi(a)=b$ if and only if $p_a\in\operatorname{dom}\tau_{\mathbf{B}(b)}$. Since $|B|<|A|$ there are distinct states $a_1, a_2\in A$ with $\varphi(a_1)=\varphi(a_2)$. Denote this state $\varphi(a_1)$ by b. As $\mathbf{A}$ is reduced, there exist $p\in\tilde{T}_{F,n}$, $q_1\neq q_2\in T_{G,n}$ $(n>0)$ and $i_0\in[n]$, as well as states $c_1, \ldots, c_{i_0-1}$, $c_{i_0+1}, \ldots, c_n, d_1, d_2\in A$ such that

$$p(c_1x_1, \ldots, c_{i_0-1}x_{i_0-1}, a_1x_{i_0}, c_{i_0+1}x_{i_0+1}, \ldots, c_nx_n) \overset{*}{\Rightarrow}_{\mathbf{A}} d_1q_1,$$

$$p(c_1x_1, \ldots, c_{i_0-1}x_{i_0-1}, a_2x_{i_0}, c_{i_0+1}x_{i_0+1}, \ldots, c_nx_n) \overset{*}{\Rightarrow}_{\mathbf{A}} d_2q_2.$$

Of course $q_1, q_2\in\tilde{T}_{G,n}$.

As $A\in\mathscr{K}'(\varrho)$ we may assume that $p=f(x_1, \ldots, x_n)$ for an operational symbol $f\in F_n$. It can be seen, by $q_1\neq q_2$ and $\mathbf{A}\in\mathscr{K}'(\varrho)$, that q_1 and q_2 are of form $q_1=q_0(r_1, \ldots, r_m)$ and $q_2=q_0(r'_1, \ldots, r'_m)$, respectively, where $q_0\in\tilde{T}_{G,m}$ $(m>0)$, $r_j, r'_j\in T_{G,n}$, furthermore, there is at least one index $j_0\in[m]$ such that $r_{j_0}\neq r'_{j_0}$, $r_{j_0}, r'_{j_0}\notin X_n$. More exactly, we may choose q_0 in such a way that $r_{j_0}=g_1(\mathbf{s}_1)$ and $r'_{j_0}=g_2(\mathbf{s}_2)$ hold for some vectors $\mathbf{s}_1$, $\mathbf{s}_2$ and different operational symbols $g_1, g_2\in G$. This implies that

$$\tau_{\mathbf{A}}(f(p_{c_1}, \ldots, p_{c_{i_0-1}}, p_{a_1}, p_{c_{i_0+1}}, \ldots, p_{c_n})) \neq \tau_{\mathbf{A}}(f(p_{c_1}, \ldots, p_{c_{i_0-1}}, p_{a_2}, p_{c_{i_0+1}}, \ldots, p_{c_n})).$$

Now let $b_i=\varphi(c_i)$ $(i=1, \ldots, n,\ i\neq i_0)$. There is a state $e\in B$ and a tree $q\in T_{G,n}$ with $f(b_1x_1, \ldots, b_{i_0-1}x_{i_0-1}, bx_{i_0}, b_{i_0+1}x_{i_0+1}, \ldots, b_nx_n)\to eq\in\Sigma'$. Since $\mathbf{A}$ and $\mathbf{B}$ are equivalent we have $\tau_{\mathbf{A}}(p_{c_i})=\tau_{\mathbf{B}}(p_{b_i})$ $(i=1, \ldots, n,\ i\neq i_0)$, $\tau_{\mathbf{A}}(p_{a_i})=\tau_{\mathbf{B}}(p_{a_i})$ $(i=1, 2)$, $q_i(\tau_{\mathbf{A}}(p_{c_1}), \ldots, \tau_{\mathbf{A}}(p_{c_{i_0-1}}), \tau_{\mathbf{A}}(p_{a_i}), \tau_{\mathbf{A}}(p_{c_{i_0+1}}), \ldots, \tau_{\mathbf{A}}(p_{c_n}))=q(\tau_{\mathbf{B}}(p_{c_1}), \ldots, \tau_{\mathbf{B}}(p_{c_{i_0-1}}), \tau_{\mathbf{B}}(p_{a_i}), \tau_{\mathbf{B}}(p_{c_{i_0+1}}), \ldots, \tau_{\mathbf{B}}(p_{c_n}))$ $(i=1, 2)$.

But $\tau_{\mathbf{A}}(f(p_{c_1}, \ldots, p_{c_{i_0-1}}, p_{a_1}, p_{c_{i_0+1}}, \ldots, p_{c_n}))\neq\tau_{\mathbf{A}}(f(p_{c_1}, \ldots, p_{c_{i_0-1}}, p_{a_2}, p_{c_{i_0+1}}, \ldots, p_{c_n}))$. Thus $\tau_{\mathbf{B}}(p_{a_1})\neq\tau_{\mathbf{B}}(p_{a_2})$ and $\operatorname{path}_{i_0}(q)\neq\emptyset$. Even more, by $r_{j_0}\neq r'_{j_0}$, there is a string $w\in\operatorname{path}_{i_0}(q)$ which is a prefix of $\operatorname{path}_{j_0}(q_0)$. Now there are two cases.

First suppose that $\operatorname{path}_{j_0}(q_0)$ is a prefix of $\operatorname{path}_{i_0}(q_1)$ and let $p_i=f(p_{c_1}, \ldots, p_{c_{i_0-1}}, p_{a_i}, p_{c_{i_0+1}}, \ldots, p_{c_n})$ $(i=1, 2)$. Then $\tau_{\mathbf{A}}(p_1)=u(\tau_{\mathbf{A}}(p_{a_1}))$ and $\tau_{\mathbf{B}}(p_1)=u'(\tau_{\mathbf{B}}(p_{a_1}))$ where $u, u'\in\hat{T}_{F,1}$ satisfy $\operatorname{path}(u)=\operatorname{path}_{i_0}(q_1)$ and $\operatorname{path}(u')=w$, respectively. As w is a proper prefix of $\operatorname{path}_{i_0}(q_1)$ and $\tau_{\mathbf{A}}(p_{a_1})=\tau_{\mathbf{B}}(p_{a_1})$ this results that $\tau_{\mathbf{A}}(p_1)\neq\tau_{\mathbf{B}}(p_1)$, contrary to our assumption $\tau_{\mathbf{A}}=\tau_{\mathbf{B}}$. A similar argument yields a contradiction if $\operatorname{path}_{j_0}(q_0)$ is assumed to be a prefix of $\operatorname{path}_{i_0}(q_2)$.

Thus none of the strings $\operatorname{path}_{i_0}(q_1)$ and $\operatorname{path}_{i_0}(q_2)$ is a postfix of $\operatorname{path}_{j_0}(q_0)$. This implies that $\tau_{\mathbf{A}}(p_1)=u(v)$, $\tau_{\mathbf{A}}(p_2)=u'(v)$, $\tau_{\mathbf{B}}(p_1)=u(v)$ and $\tau_{\mathbf{B}}(p_2)=u'(v')$ where $u, u'\in\hat{T}_{F,1}$, $v, v'\in T_G$ satisfy the conditions $\operatorname{path}(u)=\operatorname{path}(u')=w$ and $v\neq v'$. Indeed, $v=\tau_{\mathbf{B}}(p_{a_1})$, and $v'=\tau_{\mathbf{B}}(p_{a_2})$. It is again a contradiction.

DEPT. OF COMPUTER SCIENCE
A. JÓZSEF UNIVERSITY
ARADI VÉRTANUK TERE 1.
SZEGED, HUNGARY
H—6720

References

[1] ARBIB, M. A. and Y. GIVE'ON, Algebra automata I: Parallel programming as a prolegomena to the categorical approach, *Inform. and Control*, v. 12, 1968, pp. 331—345.
[2] BLATTNER, M. and T. HEAD, The decidability of equivalence for deterministic finite transducers, *J. Comput. System Sci.*, v. 19, 1979, pp. 45—49.
[3] BRAINERD, W. S., The minimization of tree-automata, *Inform. and Control*, v. 13, 1968, pp. 484—491.
[4] CULIC II, K. and A. SALOMAA, On the decidability of homomorphism equivalence for languages, *J. Comput. System Sci.*, v. 17, 1978, pp. 163—175.
[5] ENGELFRIET, J., Bottom-up and top-down tree transformations, A comparison, *Math. Systems Theory*, v. 9, 1975, pp. 198—231.
[6] ENGELFRIET, J., Top-down tree transducers with regular look-ahead, *Math. Systems Theory*, v. 10, 1977, pp. 289—303.
[7] ENGELFRIET, J., On tree transducers for partial functions, *Inform. Process. Lett.*, v. 7, 1978, pp. 170—172.
[8] ÉSIK, Z., On functional tree transducers, in *Proceedings, Conference on Fundamentals of Computation Theory*, ed. Budach, L., Akademie-Verlag, Berlin, 1979, pp. 121—127.
[9] GÉCSEG, F. and I. PEÁK, *Algebraic theory of automata*, Akadémia Kiadó, Budapest, 1972.
[10] GÉCSEG, F. and M. STEINBY, Minimal ascending tree automata, *Acta Cybernet.*, v. 4, 1978, pp. 37—44.
[11] GRIFFITHS, T. V., The unsolvability of the equivalence problem for λ-free nondeterministic generalized machines, *J. Assoc. Comput. Mach.*, v. 15, 1968, pp. 409—413.
[12] ZACHAR, Z., The solvability of the equivalence problem for deterministic frontier-to-root tree transducers, *Acta Cybernet.*, v. 4, 1978, pp. 167—177.

(Received May 8, 1980)

On isomorphic representations of commutative automata with respect to α_i-products

By B. IMREH

The purpose of this paper is to study the α_i-products (see [1]) from the point of view of isomorphic completeness for the class of all commutative automata. Namely, we give necessary and sufficient conditions for a system of automata to be isomorphically complete for the class of all commutative automata with respect to the α_i-products. It will turn out that if $i \geqq 1$ then such isomorphically complete systems coincide with each other with respect to different α_i-products. Furthermore they coincide with isomorphically complete systems of automata.

By an *automaton* we mean a finite automaton $\mathbf{A}=(X, A, \delta)$ without output. Moreover *isomorphism* and *subautomaton* will mean A-isomorphism and A-subautomaton.

Take an automaton $\mathbf{A}=(X, A, \delta)$ and let us denote by X^* the free monoid generated by X. The elements $p \in X^*$ are called *input words* of $\mathbf{A}$. The transition function δ can be extended to $A \times X^* \to A$ in a natural way: for any $p=p'x$ $(p' \in X^*, x \in X)$ and $a \in A$ $\delta(a, p)=\delta(\delta(a, p'), x)$. Further on we shall use the more convenient notation $ap_{\mathbf{A}}$ for $\delta(a, p)$ and $A'p_{\mathbf{A}}$ for the set $\{ap_{\mathbf{A}}: a \in A'\}$ where $A' \subset A$ and $p \in X^*$. If there is no danger of confusion, then we omit the index $\mathbf{A}$ in $ap_{\mathbf{A}}$ and $A'p_{\mathbf{A}}$. Define a binary relation σ on X^* in the following manner: for two input words $p, q \in X^*$, $p \equiv q$ (σ) if and only if $ap=aq$ for all $a \in A$. The quotient semigroup X^*/σ is called the *characteristic semigroup* of $\mathbf{A}$, and it will be denoted by $S(\mathbf{A})$. We use the notation $[p]$ for the element of $S(\mathbf{A})$ containing $p \in X^*$.

An automaton $\mathbf{A}=(X, A, \delta)$ is *commutative* if $ax_1x_2=ax_2x_1$ for any $a \in A$ and $x_1, x_2 \in X$. Denote by $\mathfrak{K}$ the class of all commutative automata.

Take an automaton $\mathbf{A}=(X, A, \delta)$ and let ω be an equivalence relation of the set A. It is said that ω is a congruence relation of $\mathbf{A}$ if $a \equiv b(\omega)$ implies $ax \equiv bx(\omega)$ for all $a, b \in A$ and $x \in X$. The partition induced by the congruence relation ω is called *compatible partition* of $\mathbf{A}$.

Let $\mathbf{A}=(X, A, \delta)$ be an automaton. Define the relation C of A in the following way: $a \equiv b(C)$ if and only if there exist $p, q \in X^*$ such that $ap=b$ and $bq=a$. It is clear that C is a congruence relation of $\mathbf{A}$ if the automaton $\mathbf{A}$ is commutative. In the following we use the notation $C(a)$ for the block of the partition induced by C which contains a. On the set $A/C=\{C(a): a \in A\}$ we define a partial ordering in the following way: for any $a, b \in A$, $C(a) \leqq C(b)$ if there exists $p \in X^*$ such that $ap=b$. If $C(a) \leqq C(b)$ and $C(a) \neq C(b)$ then we write $C(a) \prec C(b)$.

The automaton $\mathbf{A}=(X, A, \delta)$ is called a *permutation automaton* if for any $a, b\in A$ and $p\in X^*$, $ap=bp$ implies $a=b$. The automaton $\mathbf{A}$ is *connected* if for any $a, b\in A$ there exist $p, q\in X^*$ such that $ap=bq$.

Let $\mathbf{A}_t=(X_t, A_t, \delta_t)$ $(t=1, \ldots, n)$ be a system of automata. Moreover, let X be a finite nonvoid set and φ a mapping of $A_1\times\ldots\times A_n\times X$ into $X_1\times\ldots\times X_n$ such that $\varphi(a_1, \ldots, a_n, x)=(\varphi_1(a_1, \ldots, a_n, x), \ldots, \varphi_n(a_1, \ldots, a_n, x))$, and each φ_j $(1\leqq j\leqq n)$ is independent of states having indices greater than or equal to $j+i$, where i is a fixed nonnegative integer. We say that the automaton $\mathbf{A}=(X, A, \delta)$ with $A=A_1\times\ldots\times A_n$ and $\delta((a_1, \ldots, a_n), x)=(\delta_1(a_1, \varphi_1(a_1, \ldots, a_n, x)), \ldots, \delta_n(a_n, \varphi_n(a_1, \ldots, a_n, x)))$ is the α_i-*product* of $\mathbf{A}_t$ $(t=1, \ldots, n)$ with respect to X and φ. For this product we use the notation $\prod_{t=1}^{n}\mathbf{A}_t(X, \varphi)$ and $\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$ for $n=2$. Moreover, if in α_i-product $\mathbf{A}$, $\mathbf{A}_t=\mathbf{B}$ for all t $(t=1, \ldots, n)$, then $\mathbf{A}$ is called an α_i-*power* of $\mathbf{B}$ and we use the notation $\mathbf{A}=\mathbf{B}^n(X, \varphi)$.

Let $\mathfrak{B}$ be an arbitrary class of automata. Further on let Σ be a system of automata. Σ is called *isomorphically complete for* $\mathfrak{B}$ with respect to the α_i-product if any automaton from $\mathfrak{B}$ can be embedded isomorphically into an α_i-product of automata from Σ. If $\mathfrak{B}$ is the class of all automata and Σ is isomorphically complete for $\mathfrak{B}$, then it is said that Σ is *isomorphically complete*.

Let us denote by $\mathbf{E}_2=(\{x, y\}, \{0, 1\}, \delta_E)$ the automaton for which $\delta_E(0, y)=0$, $\delta_E(0, x)=1$, $\delta_E(1, x)=\delta_E(1, y)=1$.

An automaton $\mathbf{A}=(X, A, \delta)$ is called *monotone* if there exists a partial ordering $\leqq$ on A such that $a\leqq\delta(a, x)$ holds for any $a\in A$ and $x\in X$.

For monotone automata the following result holds:

Lemma 1. Every connected monotone automaton can be embedded isomorphically into an α_0-power of $\mathbf{E}_2$.

Proof. We proceed by induction on the number of states of the automaton. In the cases $n=1$ and $n=2$ our statement is trivial. Now let $n>2$ and suppose that the statement is valid for any natural number $m<n$. Denote by $\mathbf{A}=(X, A, \delta)$ an arbitrary connected monotone automaton with n states. Since $\mathbf{A}$ is connected thus among the blocks $C(a)$ $(a\in A)$ there exists exactly one maximal element under our partial ordering of blocks. On the other hand, since $\mathbf{A}$ is monotone thus the partition induced by C has one-element blocks only. Denote by a_n the element of the maximal block. Since $n>2$ thus there exists an $a\in A$ such that $C(a)<C(a_n)$. Denote by a_k an element of A for which $C(a_k)<C(a_n)$ and $C(a_k)<C(a)$ implies $a=a_n$ for any $a\in A$. Obviously there exists such an a_k. It is also obvious that $(X, H, \delta_{|H\times X})$ is a subautomaton of $\mathbf{A}$, where $H=\{a_k, a_n\}$ and the restriction to $H\times X$ of the function δ is denoted by $\delta_{|H\times X}$. Let us define the automata $\mathbf{A}_1=(X, (A\setminus H)\cup\{*\}, \delta_1)$ and $\mathbf{A}_2=(((A\setminus H)\cup\{*\})\times X, H\cup\{\square\}, \delta_2)$ in the following way:

$$\delta_1(a, x)=\begin{cases}\delta(a, x) & \text{if } \delta(a, x)\notin H,\\ * & \text{otherwise,}\end{cases}$$

$$\delta_1(*, x)=*,$$

$$\delta_2(\square, (a, x))=\begin{cases}\delta(a, x) & \text{if } \delta(a, x)\in H,\\ \square & \text{otherwise,}\end{cases}$$

$$\delta_2(a', (a, x))=a',\ \delta_2(a', (*, x))=\delta(a', x),\ \delta_2(\square, (*, x))=\square$$

for all $a\in A\setminus H$, $x\in X$ and $a'\in H$. Take the α_0-product $\mathbf{B}=\mathbf{A}_1\times\mathbf{A}_2(X,\varphi)$ where $\varphi_1(x)=x$, $\varphi_2(v,x)=(v,x)$ for all $x\in X$ and $v\in(A\setminus H)\cup\{*\}$. It is easy to prove that the correspondence

$$v(a)=\begin{cases}(a,\square) & \text{if } a\in A\setminus H,\\ (*,a) & \text{if } a\in H,\end{cases}$$

is an isomorphism of $\mathbf{A}$ into $\mathbf{B}$.

Now let us consider the automata $\mathbf{A}_1$ and $\mathbf{A}_2$. Since $\mathbf{A}_1$ is a connected monotone automaton with $n-1$ states thus, by our assumption, $\mathbf{A}_1$ can be embedded isomorphically into an α_0-power of $\mathbf{E}_2$. Denote by U the set of input signals of $\mathbf{A}_2$ and take the following partitions of U:

$$U_1=\{(a,x):\ a\in A\setminus H,\ x\in X,\ \delta(a,x)\notin H\}\cup\{(*,x):\ x\in X\},$$

$$U_2=\{(a,x):\ a\in A\setminus H,\ x\in X,\ \delta(a,x)=a_k\},$$

$$U_3=\{(a,x):\ a\in A\setminus H,\ x\in X,\ \delta(a,x)=a_n\},$$

$$V_1=\{(a,x):\ a\in A\setminus H,\ x\in X\}\cup\{(*,x):\ x\in X,\ \delta(a_k,x)=a_k\},$$

$$V_2=\{(*,x):\ x\in X,\ \delta(a_k,x)=a_n\}.$$

Consider the α_0-product $\mathbf{E}^2(U,\varphi)$ where $\varphi_1(u_1)=y$, $\varphi_1(u_2)=\varphi_1(u_3)=x$, $\varphi_2(0,u_1)=\varphi_2(0,u_2)=y$, $\varphi_2(0,u_3)=x$, $\varphi_2(1,v_1)=y$ and $\varphi_2(1,v_2)=x$ for all $u_i\in U_i$ $(i=1,2,3)$ and $v_j\in V_j$ $(j=1,2)$. It can easily be seen that the correspondence $\square\to(0,0)$, $a_k\to(1,0)$ and $a_n\to(1,1)$ is an isomorphism of $\mathbf{A}_2$ into $\mathbf{E}^2(U,\varphi)$. Since the formation of the α_0-product is associative thus we have proved that $\mathbf{A}$ can be embedded isomorphically into an α_0-power of $\mathbf{E}_2$.

For any natural number $n\geqq 1$ let $\mathbf{M}_n=(\{x_0,\ldots,x_{n-1}\},\{0,\ldots,n-1\},\delta)$ denote the automaton for which $\delta(j,x_l)=j+l\,(\mathrm{mod}\,n)$ for any $j\in\{0,\ldots,n-1\}$ and $x_l\in\{x_0,\ldots,x_{n-1}\}$, where $j+l\,(\mathrm{mod}\,n)$ denotes the least nonnegative residue of $j+l$ modulo n. Moreover let $\mathfrak{M}$ denote the set of all $\mathbf{M}_n$ such that n is a prime number.

It holds the following

Lemma 2. If the number of states of a strongly connected commutative automaton $\mathbf{A}$ is a prime number, then there exists an automaton $\mathbf{M}\in\mathfrak{M}$ such that $\mathbf{A}$ is isomorphic to an α_0-product of $\mathbf{M}$ with a single factor.

Proof. First we prove that every strongly connected commutative automaton is a permutation automaton. Indeed, denote by $\mathbf{A}=(X,A,\delta)$ a strongly connected commutative automaton and assume that there exist $a,b\in A$ and $p\in X^*$ with $ap=bp$. Since $\mathbf{A}$ is strongly connected thus there exist input words $q,w\in X^*$ such $apq=a$ and $aw=b$. Using the commutativity of $\mathbf{A}$, we have $bpq=awpq=apqw=aw=b$. Therefore, $a=apq=bpq=b$, showing that $\mathbf{A}$ is a permutation automaton.

Now let us assume that the number of states of $\mathbf{A}$ is prime and denote it by r. Let $a\in A$ and $p\in X^*$ be arbitrary and consider the states $a, ap, ap^2, \ldots$. Since $\mathbf{A}$ is a permutation automaton thus there exists a t $(1\leqq t\leqq r)$ such that $a=ap^t$. Denote by (a,p) the set $\{a,ap,\ldots,ap^{t-1}\}$. Assume that $(a,p)\subset A$. Let $a'\in A\setminus(a,p)$ and consider the set (a',p), which is defined as above. Since $\mathbf{A}$ is a strongly connected

automaton thus there exists a $q\in X^*$ such that $aq=a'$. Using the commutativity of **A** we have $ap^iq=aqp^i=a'p^i$ $(i=0, \ldots, t-1)$. From this it follows that (a, p) and (a', p) have the same cardinality since **A** is a permutation automaton. On the other hand it can easily be seen that (a, p) and (a', p) are disjoint subsets of A. Therefore, the set $\varrho_p=\{(a, p): a\in A\}$ is a partition of A and the blocks of ϱ_p have the same cardinality. Since r is prime thus we get that ϱ_p has one-element blocks only, or it has one block only. Now we choose an $x\in X$ such that ϱ_x has one block only. The automaton **A** is strongly connected therefore such an $x\in X$ exists. Let $a\in A$ be a fixed state of **A** and write $a_0=a$, $a_i=a_0x^i$ $(i=1, \ldots, r-1)$. Thus the mapping induced by x on A can be described in the form $a_ix=a_{i+1\,(\mathrm{mod}\,r)}$ $(i=0, \ldots, r-1)$. Now let y be an arbitrary input signal of **A** and assume that $a_0y=a_j$ for some $j\in\{0, 1, \ldots, r-1\}$. From the commutativity of **A** we have $a_iy=a_0x^iy=a_0yx^i=a_jx^i=a_{i+j\,(\mathrm{mod}\,r)}$ for all $i\in\{0, 1, \ldots, r-1\}$. Take the α_0-product $\mathbf{B}=\Pi\mathbf{M}_r(X, \varphi)$ with a single factor, where $\varphi(x)=x_k$ if $a_0x=a_k$ for all $x\in X$. It is easy to prove that **A** is isomorphic to **B**, which completes the proof of Lemma 2.

Lemma 3. Every strongly connected commutative automaton can be embedded isomorphically into an α_0-product of automata from $\mathfrak{M}$.

Proof. We prove by induction on the number of states of the automaton. In case $n<4$, by Lemma 2, the statement holds. Now let $n\geqq 4$ and assume that our statement is valid for any natural number $m<n$. Denote by $\mathbf{A}=(X, A, \delta)$ an arbitrary strongly connected commutative automaton with n states. If n is prime then, by Lemma 2, the statement holds. Assume that n is not prime. Let $p\in X^*$ be arbitrary. Consider the partition ϱ_p. Since **A** is commutative thus ϱ_p is a compatible partition of **A**. Denote by Ω the set of all partitions ϱ_p of **A** such that $[p]\in S(\mathbf{A})\setminus\{[e]\}$, where e denotes the empty word of X^*. Take the partition ϱ of **A** given by $\varrho=\bigcap_{\varrho_p\in\Omega}\varrho_p$. We distinguish two cases.

First assume that ϱ has one-element blocks only. In this case it can easily be seen that **A** can be embedded isomorphically into the direct product of the quotient automata $\mathbf{A}/\varrho_p$ $(\varrho_p\in\Omega)$. On the other hand, for any $\varrho_p\in\Omega$ the quotient automaton $\mathbf{A}/\varrho_p$ is a strongly connected commutative automaton with number of states less than n. Therefore, by our induction hypothesis the statement is valid.

Now assume, that there exist $a, b\in A$ such that $a\neq b$ and $a\equiv b(\varrho)$. Take an input signal x of **A** such that the mapping induced by it on A is not the identity. Then $\varrho_x\in\Omega$ and thus $\varrho_x\geqq\varrho$. Therefore, $a\equiv b(\varrho_x)$. This means that there exists a natural number $l>0$ such that $ax^l=b$. Since ϱ is compatible thus $ax^l\equiv bx^l(\varrho)$. From this, by the above equality, we get that the states $a, ax^l, ax^{2l}, \ldots$ are in $\varrho(a)$. Therefore, $(a, x^l)\subseteqq\varrho(a)$. On the other hand $\varrho_{x^l}\geqq\varrho$ thus $(a, x^l)=\varrho(a)$, showing that $\varrho_{x^l}=\varrho$. Denote by p the word x^l and assume that $\varrho(a)=\{a, ap, \ldots, ap^{k-1}\}$. We show that k is prime. Indeed, if $1<v<k$ and $v|k$ then $(a, p^v)\subset(a, p)$ which contradicts the relation $\varrho_{p^v}\geqq\varrho$. Denote by $\varrho(a_0), \varrho(a_1), \ldots, \varrho(a_{s-1})$ the blocks of ϱ. From the equality $\varrho=\varrho_p$ it follows that $\varrho(a_i)=\{a_i, a_ip, \ldots, a_ip^{k-1}\}$ $(i=0, 1, \ldots, s-1)$. Thus $n=k\cdot s$. From this we get that $s\neq 1$ because k is prime. On the other hand, since **A** is strongly connected thus there exist words p_i, q_i $(i=0, \ldots, s-1)$ such that $a_0p_i=a_i$ and $a_iq_i=a_0$ for all $i\in\{0, 1, \ldots, s-1\}$. Using the commutativity of **A** we have $a_0p^jp_i=a_ip^j$ and $a_ip^jq_i=a_0p^j$ for any $j\in\{0, 1, \ldots, k-1\}$ and $i\in\{0, 1, \ldots, s-1\}$. Now define two automata $\mathbf{A}_1=(X, \varrho, \delta_1)$ and $\mathbf{A}_2=(\varrho\times X, \varrho(a_0), \delta_2)$ in the following way: $\delta_1(\varrho(a_i), x)=\varrho(\delta(a_i, x))$ for all $\varrho(a_i)\in\varrho$

and $x\in X$, $\delta_2(a_0p^j, (\varrho(a_i), x))=a_0p^jp_ixq_u$ if $\varrho(\delta(a_i, x))=\varrho(a_u)$ for all $a_0p^j\in\varrho(a_0)$ and $(\varrho(a_i), x)\in\varrho\times X$. Take the α_0-product $\mathbf{B}=\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$, where $\varphi_1(x)=x$ and $\varphi_2(\varrho(a_i), x)=(\varrho(a_i), x)$ for any $x\in X$ and $\varrho(a_i)\in\varrho$. It is not difficult to prove that the correspondence v: $a_ip^j\to(\varrho(a_i), a_0p^j)$ $(i=0, 1, \ldots, s-1;\ j=0, 1, \ldots, k-1)$ is an isomorphism of $\mathbf{A}$ into $\mathbf{B}$. Now consider the automata $\mathbf{A}_1$ and $\mathbf{A}_2$. They are strongly connected commutative automata with number of states less then n. Therefore, by our assumption, the statement holds.

For any prime number r, let $\overline{\mathbf{M}}_r=(\{x_0, x_1, \ldots, x_r\}, \{0, \ldots, r\}, \delta)$ denote the automaton for which $\delta(l, x_j)=l+j\,(\mathrm{mod}\,r)$, $\delta(r, x_j)=r$, $\delta(l, x_r)=r$ and $\delta(r, x_r)=r$ for any $l\in\{0, \ldots, r-1\}$ and $x_j\in\{x_0, \ldots, x_{r-1}\}$.

The next Theorem gives necessary and sufficient conditions for a system of automata to be isomorphically complete for $\mathfrak{K}$ with respect to the α_0-product.

Theorem 1. A system Σ of automata is isomorphically complete for $\mathfrak{K}$ with respect to the α_0-product if and only if the following conditions are satisfied:

(1) There exists $\mathbf{A}_0\in\Sigma$ such that the automaton $\mathbf{E}_2$ can be embedded isomorphically into an α_0-product of $\mathbf{A}_0$ with a single factor;

(2) For any prime number r there exists $\mathbf{A}\in\Sigma$ such that the automaton $\overline{\mathbf{M}}_r$ can be embedded isomorphically into an α_0-product of the automata $\mathbf{A}_0$ and $\mathbf{A}$.

Proof. In order to prove the necessity assume that Σ is isomorphically complete for $\mathfrak{K}$ with respect to the α_0-product. Then $\mathbf{E}_2$ can be embedded isomorphically into an α_0-product $\prod_{i=1}^{k}\mathbf{A}_i(\{x, y\}, \varphi)$ of automata from Σ. Assume that $k>1$ and let μ denote a suitable isomorphism. For any $j\in\{0, 1\}$ denote by $(a_{j1}, \ldots, a_{jk})$ the image of j under μ. Among the sets $\{a_{0t}, a_{1t}\}$ $(t=1, \ldots, k)$ there should be at least one which has more than one element. Let l be the least index for which $a_{0l}\neq a_{1l}$. It is obvious that the automaton $\mathbf{A}_l\in\Sigma$ satisfies condition (1).

Now take an arbitrary prime number r and consider the automaton $\overline{\mathbf{M}}_r$. By our assumption $\overline{\mathbf{M}}_r$ can be embedded isomorphically into an α_0-product $\prod_{i=1}^{k}\mathbf{A}_i(\{x_0, \ldots, x_r\}, \varphi)$ of automata from Σ. Assume that $k>1$ and let μ denote a suitable isomorphism. For any $t\in\{0, \ldots, r\}$ denote by $(a_{t1}, \ldots, a_{tk})$ the image of t under μ. Define compatible partitions π_j $(j=1, \ldots, k)$ of $\overline{\mathbf{M}}_r$ in the following way: for any $u, v\in\{0, \ldots, r\}$, $u\equiv v(\pi_j)$ if and only if $a_{u1}=a_{v1}, \ldots, a_{uj}=a_{vj}$. It is obvious that $\pi_1\geqq\pi_2\geqq\ldots\geqq\pi_k$ and π_k has one-element blocks only. On the other hand $\overline{\mathbf{M}}_r$ has only one nontrivial compatible partition: $\sigma=\{\{0, \ldots, r-1\}, \{r\}\}$. Denote by s the least index for which $\sigma>\pi_s$. It is not difficult to prove that the automaton $\mathbf{A}_s\in\Sigma$ satisfies condition (2).

To prove the sufficiency of the conditions of Theorem 1 we shall show that arbitrary commutative automaton can be embedded isomorphically into an α_0-product of automata from $\mathfrak{N}$ where $\mathfrak{N}=\{\mathbf{E}_2\}\cup\{\overline{\mathbf{M}}_r: r \text{ is a prime number}\}$.

We prove by induction on the number of states of the automaton. In the case $n\leqq 2$ our statement is trivial. Now let $n>2$ and assume that for any $m<n$ the statement is valid. Denote by $\mathbf{A}=(X, A, \delta)$ an arbitrary commutative automaton with n states.

If $\mathbf{A}$ is not connected then it can be given as a direct sum of its connected subautomata. Denote by $\mathbf{A}_t=(X, A_t, \delta_t)$ $(t=1, \ldots, k)$ these subautomata of $\mathbf{A}$. Take

an arbitrary symbol z such that $z \notin X$. Define the automata $\bar{\mathbf{A}}_i = (X \cup \{z\}, A_i, \bar{\delta}_i)$ $(i=1, \ldots, k)$ in the following way: $\bar{\delta}_i(a_i, x) = \delta_i(a_i, x)$ and $\bar{\delta}_i(a_i, z) = a_i$, for all $a_i \in A_i$ and $x \in X$ $(i=1, \ldots, k)$. Take the α_0-products $\mathbf{B}_i = \mathbf{E}_2 \times \bar{\mathbf{A}}_i(X, \varphi^{(i)})$ $(i=1, \ldots, k)$ where $\varphi_1^{(i)}(x) = y$, $\varphi_2^{(i)}(0, x) = z$ and $\varphi_2^{(i)}(1, x) = x$ for all $x \in X$. It is clear that $\mathbf{A}$ can be embedded isomorphically into the direct product $\prod_{i=1}^{k} \mathbf{B}_i$. On the other hand, for any index i $(1 \leqq i \leqq k)$ the automaton $\bar{\mathbf{A}}_i$ is commutative with number of states less than n. Therefore, by our induction hypothesis the statement holds.

Now assume that $\mathbf{A}$ is connected. Consider the partition $\{C(a): a \in A\}$ and the partial ordering of blocks introduced on page 1. Since $\mathbf{A}$ is connected thus among the blocks there exists one maximal only. Let $C(\bar{a})$ denote this block. We distinguish two cases.

(I) Assume that the cardinality of $C(\bar{a})$ is greater than one. In this case $(X, C(\bar{a}), \delta_{|C(\bar{a}) \times X})$ is a strongly connected subautomaton of $\mathbf{A}$. If $C(\bar{a}) = A$ then, by Lemma 2 and Lemma 3, the statement holds. If $C(\bar{a}) \subset A$ then we distinguish three cases.

(a) Assume that the cardinality of $C(\bar{a})$ is prime and denote it by r. Let us define the automata $\mathbf{A}_1 = (X, (A \setminus C(\bar{a})) \cup \{*\}, \delta_1)$ and $A_2 = (((A \setminus C(\bar{a})) \cup \{*\}) \times X, C(\bar{a}) \cup \{\square\}, \delta_2)$ in the following way:

$$\delta_1(a, x) = \begin{cases} \delta(a, x) & \text{if} \quad \delta(a, x) \notin C(\bar{a}), \\ * & \text{otherwise,} \end{cases}$$

$$\delta_1(*, x) = *,$$

$$\delta_2(a', (a, x)) = a', \quad \delta_2(a', (*, x)) = \delta(a', x), \quad \delta_2(\square, (*, x)) = \square,$$

$$\delta_2(\square, (a, x)) = \begin{cases} \delta(a, x) & \text{if} \quad \delta(a, x) \in C(\bar{a}), \\ \square & \text{otherwise,} \end{cases}$$

for all $x \in X$, $a \in A \setminus C(\bar{a})$ and $a' \in C(\bar{a})$. Take the α_0-product $\mathbf{B} = \mathbf{A}_1 \times \mathbf{A}_2(X, \varphi)$ where $\varphi_1(x) = x$ and $\varphi_2(v, x) = (v, x)$ for any $x \in X$, $v \in (A \setminus C(\bar{a})) \cup \{*\}$. It can be proved easily that the correspondence

$$\nu(a) = \begin{cases} (a, \square) & \text{if} \quad a \in A \setminus C(\bar{a}), \\ (*, a) & \text{if} \quad a \in C(\bar{a}), \end{cases}$$

is an isomorphism of $\mathbf{A}$ into $\mathbf{B}$. Consider the automata $\mathbf{A}_1$ and $\mathbf{A}_2$. $\mathbf{A}_1$ is a commutative automaton with number of states less than n. Therefore, by our induction assuption, it can be decomposed in the form required. For investigating $\mathbf{A}_2$ we need the automaton $\mathbf{C} = (\{x_0, \ldots, x_r\}, \{0, \ldots, r\}, \delta_C)$ where $\delta_C(l, x_i) = l + i \pmod r$, $\delta_C(l, x_r) = l$, $\delta_C(r, x_i) = i$ and $\delta_C(r, x_r) = r$ for any $l \in \{0, \ldots, r-1\}$, $x_i \in \{x_0, \ldots, x_{r-1}\}$. Now denote by U the set of the input signals of $\mathbf{A}_2$ and consider the following partitions of U:

$$U_1 = \{(*, x): x \in X\} \cup \{(a, x): a \in A \setminus C(\bar{a}), \ x \in X, \ \delta(a, x) \notin C(\bar{a})\},$$

$$U_2 = \{(a, x): a \in A \setminus C(\bar{a}), \ x \in X, \ \delta(a, x) \in C(\bar{a})\},$$

$$V_1 = \{(a, x): a \in A \setminus C(\bar{a}), \ x \in X\},$$

$$V_2 = \{(*, x): x \in X\}.$$

By Lemma 2, we have that $(X, C(\bar{a}), \delta_{|C(\bar{a})\times X})$ is isomorphic to an α_0-product of $\mathbf{M}_r$ with a single factor. Denote by μ this isomorphism. We write $a=a_i$ if $\mu(i)=a$ $(i=0, 1, \ldots, r-1)$. Now take the α_0-product $\mathbf{E}_2\times\mathbf{C}(U, \varphi)$ where for any $u_1\in U_1$, $u_2\in U_2$ and $v_1\in V_1$, $v_2\in V_2$, $\varphi_1(u_1)=y$, $\varphi_1(u_2)=x$, $\varphi_2(0, u_1)=x_r$, $\varphi_2(0, u_2)=x_i$ if $\delta_2(\square, u_2)=a_i$, $\varphi_2(1, v_1)=x_r$ and $\varphi_2(1, v_2)=x_j$ if $\delta_2(a_0, v_2)=a_j$. It is clear that the correspondence ν given by $\nu(\square)=(0, r)$ and $\nu(a_i)=(1, i)$ $(i=0, \ldots, r-1)$ is an isomorphism of $\mathbf{A}_2$ into $\mathbf{E}_2\times\mathbf{C}(U, \varphi)$. On the other hand, it is not difficult to prove that $\mathbf{C}$ can be embedded isomorphically into an α_0-product of $\mathbf{E}_2$ and $\mathbf{M}_r$. Thus $\mathbf{A}_2$ can be embedded isomorphically into an α_0-product of $\mathbf{E}_2$ and $\mathbf{M}_r$. Taking into consideration the above decomposition of $\mathbf{A}_1$, this ends the discussion of (a) in case (I).

(b) Assume that the cardinality of $C(\bar{a})$ is not prime and the partition ϱ of $(X, C(\bar{a}), \delta_{|C(\bar{a})\times X})$ has one-element blocks only where ϱ is defined for $(X, C(\bar{a}), \delta_{|C(\bar{a})\times X})$ in the same way as in the proof of Lemma 3. Now for any $\varrho_p\in\Omega$, define the partition $\bar{\varrho}_p$ of $\mathbf{A}$ in the following way:

$$\bar{\varrho}_p(a)=\begin{cases}\{a\} & \text{if } a\in A\setminus C(\bar{a}),\\ \varrho_p(a) & \text{otherwise.}\end{cases}$$

Now let $\bar{\Omega}$ denote the set of all such $\bar{\varrho}_p$. It can easily be seen that $\mathbf{A}$ can be embedded isomorphically into the direct product $\prod_{\bar{\varrho}_p\in\bar{\Omega}} \mathbf{A}/\bar{\varrho}_p$. On the other hand for any $\bar{\varrho}_p\in\bar{\Omega}$ the quotient automaton $\mathbf{A}/\bar{\varrho}_p$ is commutative with number of states less than n. Thus, by our induction assumption, we have a required decomposition of $\mathbf{A}$.

(c) Assume that the cardinality of $C(\bar{a})$ is not prime and the partition ϱ of $(X, C(\bar{a}), \delta_{|C(\bar{a})\times X})$ has at least one block whose cardinality is greater than one. Then, by the proof of Lemma 3, $(X, C(\bar{a}), \delta_{|C(\bar{a})\times X})$ can be embedded isomorphically into an α_0-product of automata $\bar{\mathbf{A}}_1=(X, \varrho, \bar{\delta}_1)$ and $\bar{\mathbf{A}}_2=(\varrho\times X, \varrho(a_0), \bar{\delta}_2)$ where $\bar{\mathbf{A}}_2$ is isomorphic to an α_0-product of $\mathbf{M}_r$ with a single factor for some prime $r<n$. Define the automata $\mathbf{A}_1=(X, (A\setminus C(\bar{a}))\cup\varrho, \delta_1)$ and $\mathbf{A}_2=(((A\setminus C(\bar{a}))\cup\varrho)\times X, \varrho(a_0)\cup\{\square\}, \delta_2)$ in the following way: for any $a\in A\setminus C(\bar{a})$, $\varrho(a_i)\in\varrho$, $x\in X$ and $a_0p^j\in\varrho(a_0)$

$$\delta_1(\varrho(a_i), x)=\bar{\delta}_1(\varrho(a_i), x),$$

$$\delta_1(a, x)=\begin{cases}\delta(a, x) & \text{if } \delta(a, x)\in A\setminus C(\bar{a})\\ \varrho(a_i) & \text{if } \delta(a, x)\in C(\bar{a}) \text{ and } \delta(a, x)\in\varrho(a_i),\end{cases}$$

$$\delta_2(a_0p^j, (a, x))=a_0p^j,\ \delta_2(a_0p^j, (\varrho(a_i), x))=\bar{\delta}_2(a_0p^j, (\varrho(a_i), x)),$$

$$\delta_2(\square, (\varrho(a_i), x))=\square,$$

$$\delta_2(\square, (a, x))=\begin{cases}\delta(a, x)q_s & \text{if } \delta(a, x)\in\varrho(a_s),\\ \square & \text{if } \delta(a, x)\notin C(\bar{a}).\end{cases}$$

Notations used in the above definition coincide with those used in the proof of Lemma 3. Take the α_0-product $\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$ where $\varphi_1(x)=x$ and $\varphi_2(v, x)=(v, x)$ for any $x\in X$ and $v\in(A\setminus C(\bar{a}))\cup\varrho$. It can easily be seen that the correspondence

$$\nu(a)=\begin{cases}(a, \square) & \text{if } a\in A\setminus C(\bar{a}),\\ (\varrho(a_i), a_0p^j) & \text{if } a\in\varrho(a_i) \text{ and } a=a_ip^j,\end{cases}$$

is an isomorphism of $\mathbf{A}$ into $\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$. Consider the automata $\mathbf{A}_1$ and $\mathbf{A}_2$. The automaton $\mathbf{A}_1$ is commutative with number of states less than n. Therefore, by our induction hypothesis, it can be decomposed in the form required. The automaton $\mathbf{A}_2$ can be embedded isomorphically into an α_0-product of automata $\mathbf{E}_2$ and $\mathbf{M}_r$. This can be proved in a similar way as in the case (a). Thus we get a required decomposition of $\mathbf{A}$.

(II) Now assume that the cardinality of $C(\bar{a})$ is equal to one. Denote by R' the set of all $a\in A$ for which the cardinality of $C(a)$ is equal to one and $C(a)<C(b)$ implies $b=\bar{a}$ for all $b\in A$. Let R be the set $R'\cup\{\bar{a}\}$. We distinguish two cases:

(a) First assume that R' is nonvoid. Then $(X, R, \delta_{|R\times X})$ is a connected monotone subautomaton of $\mathbf{A}$. Define the automata $\mathbf{A}_1=(X, (A\setminus R)\cup\{*\}, \delta_1)$ and $A_2=$ $=(((A\setminus R)\cup\{*\})\times X, R\cup\{\square\}, \delta_2)$ in the following way: for any $a\in A\setminus R$, $a'\in R$ and $x\in X$

$$\delta_1(a, x)=\begin{cases}\delta(a, x) & \text{if} \quad \delta(a, x)\notin R,\\ * & \text{otherwise,}\end{cases}$$

$$\delta_1(*, x)=*,$$

$$\delta_2(a', (a, x))=a',\ \delta_2(a', (*, x))=\delta(a', x),\ \delta_2(\square, (*, x))=\square,$$

$$\delta_2(\square, (a, x))=\begin{cases}\delta(a, x) & \text{if} \quad \delta(a, x)\in R,\\ \square & \text{otherwise.}\end{cases}$$

Take the α_0-product $\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$ where $\varphi_1(x)=x$, $\varphi_2(v, x)=(v, x)$ for any $x\in X$ and $v\in(A\setminus R)\cup\{*\}$. It is obvious that the correspondence

$$\nu(a)=\begin{cases}(a, \square) & \text{if} \quad a\in A\setminus R,\\ (*, a) & \text{if} \quad a\in R,\end{cases}$$

is an isomorphism of $\mathbf{A}$ into $\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$. Consider $\mathbf{A}_1$ and $\mathbf{A}_2$. $\mathbf{A}_1$ is commutative with number of states less than n. Thus by our induction assumption, it can be decomposed in the form required. On the other hand $\mathbf{A}_2$ is a connected monotone automaton thus, by Lemma 1, it can be embedded isomorphically into an α_0-power of $\mathbf{E}_2$. Therefore, we get a required decomposition of $\mathbf{A}$.

(b) Now assume that R' is empty. Denote by Q the set of all blocks $C(a)$ for which the cardinality of $C(a)$ is greater than one, and $C(a)<C(b)$ implies $b=\bar{a}$ for all $b\in A$. Since $\mathbf{A}$ is connected and R' is empty thus the set Q contains at least one block. We distinguish two cases.

(1) First assume that Q contains the bloks $C(a_1), \ldots, C(a_k)$ where $k>1$. Define compatible partitions ϱ_i $(i=1, \ldots, k)$ of $\mathbf{A}$ in the following way:

$$\varrho_i(a)=\begin{cases}\{a\} & \text{if} \quad a\notin C(a_i)\cup\{\bar{a}\},\\ C(a_i)\cup\{\bar{a}\} & \text{otherwise.}\end{cases}$$

It is not difficult to prove that $\bigcap\limits_{1\leq i\leq k}\varrho_i=\{\{a\}: a\in A\}$. From this we get that $\mathbf{A}$ can be embedded isomorphically into the direct product $\prod\limits_{i=1}^{k}\mathbf{A}/\varrho_i$. On the other hand, for any $i\in\{1, \ldots, k\}$ the quotient automaton $\mathbf{A}/\varrho_i$ is commutative with number of

states less than n. Therefore, by our induction assumption, we have a required decomposition of $\mathbf{A}$.

(2) Now assume that Q contains one block only and denote it by $C(b)$. Since C is a compatible partition of $\mathbf{A}$ thus $\{X_1, X_2\}$ is a partition of X where $X_1 = \{x: x\in X, C(b)x\subseteq C(b)\}$ and $X_2=\{x: x\in X, C(b)x=\bar{a}\}$. It is clear that X_1 and X_2 are nonvoid sets and $\mathbf{B}=(X_1, C(b), \delta_{|C(b)\times X_1})$ is a strongly connected commutative automaton. Now we distinguish three cases according to Lemma 3.

(i) Assume that the number of states of $\mathbf{B}$ is prime and denote it by r. Define the automata $\mathbf{A}_1=(X, (A\setminus(C(b)\cup\{\bar{a}\}))\cup\{*\}, \delta_1)$ and $\mathbf{A}_2=(((A\setminus(C(b)\cup\{\bar{a}\}))\cup\cup\{*\})\times X, C(b)\cup\{\bar{a}, \square\}, \delta_2)$ in the following way: for any $x\in X$, $a\in A\setminus(C(b)\cup\cup\{\bar{a}\})$ and $a'\in C(b)\cup\{\bar{a}\}$

$$\delta_1(a, x) = \begin{cases}\delta(a, x) & \text{if} \quad \delta(a, x)\notin C(b)\cup\{\bar{a}\},\\ * & \text{otherwise,}\end{cases}$$

$$\delta_1(*, x) = *,$$

$$\delta_2(\square, (a, x)) = \begin{cases}\delta(a, x) & \text{if} \quad \delta(a, x)\in C(b)\cup\{\bar{a}\},\\ \square & \text{otherwise,}\end{cases}$$

$$\delta_2(a', (a, x)) = a', \quad \delta_2(a', (*, x)) = \delta(a', x), \quad \delta_2(\square, (*, x)) = \square.$$

Take the α_0-product $\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$ where $\varphi_1(x)=x$ and $\varphi_2(v, x)=(v, x)$ for any $x\in X, v\in(A\setminus(C(b)\cup\{\bar{a}\}))\cup\{*\}$. It is clear that the correspondence

$$v(a) = \begin{cases}(a, \square) & \text{if} \quad a\notin C(b)\cup\{\bar{a}\},\\ (*, a) & \text{if} \quad a\in C(b)\cup\{\bar{a}\}\end{cases}$$

is an isomorphism of $\mathbf{A}$ into $\mathbf{A}_1\times A_2(X, \varphi)$. Consider the factors of the previous α_0-product. $\mathbf{A}_1$ is commutative with number of states less than n. Thus, by our induction assumption it can be decomposed in the required form. For investigating $\mathbf{A}_2$, we need the following automaton. Denote by $\mathbf{W}=(\{x_0, \ldots, x_r, \bar{x}\}, \{0, \ldots, r, \bar{r}\}, \delta_W)$ the automaton where $\delta_W(l, x_i)=l+i \pmod r$, $\delta_W(\bar{r}, x_i)=i$, $\delta_W(l, x_r)=r$, $\delta_W(l, \bar{x})=l$, $\delta_W(r, x_i)=r$ for any $l\in\{0, \ldots, r-1\}$ and $x_i\in\{x_0, \ldots, x_{r-1}\}$, and $\delta_W(r, x_r)=\delta_W(r, \bar{x})=\delta_W(\bar{r}, x_r)=r$, $\delta_W(\bar{r}, \bar{x})=\bar{r}$. Now denote by U the set of the input signals of $\mathbf{A}_2$ and take the following partitions of U.

$$U_1 = \{(*, x): x\in X\}\cup\{(a, x): a\in A\setminus(C(b)\cup\{\bar{a}\}), x\in X, \delta(a, x)\notin C(b)\cup\{\bar{a}\}\},$$

$$U_2 = \{(a, x): a\in A\setminus(C(b)\cup\{\bar{a}\}), x\in X, \delta(a, x)\in C(b)\},$$

$$U_3 = \{(a, x): a\in A\setminus(C(b)\cup\{\bar{a}\}), x\in X, \delta(a, x) = \bar{a}\},$$

$$V_1 = \{(a, x): a\in A\setminus(C(b)\cup\{\bar{a}\}), x\in X\},$$

$$V_2 = \{(*, x): x\in X_1\} \quad \text{and} \quad V_3 = \{(*, x): x\in X_2\}.$$

By definitions, we have that $(V_1\cup V_2, C(b), \delta_{2|C(b)\times(V_1\cup V_2)})$ is a strongly connected commutative automaton with r states. Thus, by Lemma 2, it is isomorphic to an α_0-product of $\mathbf{M}_r$ with a single factor. Denote by μ a suitable isomorphism, and for any $t\in\{0, 1, \ldots, r-1\}$ denote by b_t the image of t under μ. Now take the α_0-product $\mathbf{E}_2\times\mathbf{W}(U, \varphi)$ where $\varphi_1(u_1)=y$, $\varphi_1(u_2)=\varphi_1(u_3)=x$, $\varphi_2(0, u_1)=\bar{x}$,

$\varphi_2(0, u_2)=x_i$ if $\delta_2(\square, u_2)=b_i$, $\varphi_2(0, u_3)=x_r$, $\varphi_2(1, v_1)=\bar{x}$, $\varphi_2(1, v_2)=x_s$ if $\delta_2(b_0, v_2)=b_s$, $\varphi_2(1, v_3)=x_r$ for any $u_t\in U_t$ $(t=1, 2, 3)$, $v_j\in V_j$ $(j=1, 2, 3)$. It is obvious that the correspondence ν given by $\nu(\square)=(0, \bar{r})$, $\nu(\bar{a})=(1, r)$, $\nu(b_i)=(1, i)$ $(i=0, \ldots, r-1)$ is an isomorphism of $\mathbf{A}_2$ into $\mathbf{E}_2\times\mathbf{W}(U, \varphi)$. On the other hand, it is not difficult to prove that the automaton $\mathbf{W}$ can be embedded isomorphically into an α_0-product of $\mathbf{E}_2$ and $\overline{\mathbf{M}}_r$. Thus we get a required decomposition of $\mathbf{A}$.

(ii) Assume that the number of states of $\mathbf{B}$ is not prime and the partition ϱ of $\mathbf{B}$ has one-element blocks only where ϱ is defined for $\mathbf{B}$ in the same way as above. Now for any $\varrho_p\in\Omega$ define a partition $\bar{\varrho}_p$ of $\mathbf{A}$ in the following way:

$$\bar{\varrho}_p(a)=\begin{cases}\{a\} & \text{if} \quad a\in A\setminus C(\bar{a}),\\ \varrho_p(a) & \text{otherwise.}\end{cases}$$

Let $\bar{\Omega}$ denote the set of all such $\bar{\varrho}_p$. It is clear that $\mathbf{A}$ can be embedded isomorphically into the direct product $\prod_{\bar{\varrho}_p\in\bar{\Omega}} \mathbf{A}/\bar{\varrho}_p$. The quotient automaton $\mathbf{A}/\bar{\varrho}_p$ is commutative and its number of states is less than n for any $\bar{\varrho}_p\in\bar{\Omega}$. Thus, by our induction assumption we have a required decomposition of $\mathbf{A}$.

(iii) Assume that the number of states of $\mathbf{B}$ is not prime and the partition ϱ of $\mathbf{B}$ has at least one block whose cardinality is greater than one. Then, by Lemma 3, $\mathbf{B}$ can be embedded isomorphically into an α_0-product of the automata $\mathbf{B}_1=(X_1, \varrho, \bar{\delta}_1)$ and $\mathbf{B}_2=(\varrho\times X_1, \varrho(b_0), \bar{\delta}_2)$ where $\mathbf{B}_2$ is isomorphic to an α_0-product of $\mathbf{M}_r$ with a single factor for some prime r. Define the automata $\mathbf{A}_1=(X, (A\setminus C(b))\cup\varrho, \delta_1)$ and $\mathbf{A}_2=(((A\setminus C(b))\cup\varrho)\times X, \varrho(b_0)\cup\{*, \square\}, \delta_2)$ in the following way: for any $a\in A\setminus C(b)$, $\varrho(b_i)\in\varrho$, $x\in X$ and $b_0p^j\in\varrho(b_0)$

$$\delta_1(a, x)=\begin{cases}\delta(a, x) & \text{if} \quad \delta(a, x)\notin C(b),\\ \varrho(\delta(a, x)) & \text{otherwise,}\end{cases}$$

$$\delta_1(\varrho(b_i), x)=\begin{cases}\bar{\delta}_1(\varrho(b_i), x) & \text{if} \quad x\in X_1,\\ \bar{a} & \text{if} \quad x\in X_2,\end{cases}$$

$$\delta_2(b_0p^j, (\varrho(b_i), x))=\begin{cases}\bar{\delta}_2(b_0p^j, (\varrho(b_i), x)) & \text{if} \quad x\in X_1,\\ * & \text{if} \quad x\in X_2,\end{cases}$$

$$\delta_2(\square, (a, x))=\begin{cases}\square & \text{if} \quad \delta(a, x)\in A\setminus(C(b)\cup\{\bar{a}\}),\\ \delta(a, x)q_s & \text{if} \quad \delta(a, x)\in\varrho(b_s),\\ * & \text{if} \quad \delta(a, x)=\bar{a},\end{cases}$$

$$\delta_2(b_0p^j, (a, x))=b_0p^j, \quad \delta_2(*, (a, x))=\delta_2(*, (\varrho(b_i), x))=*,$$

$$\delta_2(\square, (\varrho(b_i), x))=\square.$$

(The notations coincide with those used in the proof of the Lemma 3.) Take the α_0-product $\mathbf{A}_1\times\mathbf{A}_2(X, \varphi)$ where $\varphi_1(x)=x$ and $\varphi_2(v, x)=(v, x)$ for any $x\in X$ and $v\in(A\setminus C(b))\cup\varrho$. It is not difficult to prove that the correspondence

$$\nu(a)=\begin{cases}(a, \square) & \text{if} \quad a\in A\setminus(C(b)\cup\{\bar{a}\}),\\ (\varrho(b_i), b_0p^j) & \text{if} \quad a\in C(b) \text{ and } a=b_ip^j,\\ (\bar{a}, *) & \text{if} \quad a=\bar{a},\end{cases}$$

is an isomorphism of $\mathbf{A}$ into $\mathbf{A}_1 \times \mathbf{A}_2(X, \varphi)$. Consider the automata $\mathbf{A}_1$ and $\mathbf{A}_2$. The automaton $\mathbf{A}_1$ is commutative with number of states less than n. Thus, by our induction assumption, it can be decomposed in the required form. The automaton $\mathbf{A}_2$ can be embedded isomorphically into an α_0-product of $\mathbf{E}_2$ and $\overline{\mathbf{M}}_r$. This can be proved in a similar way as in the case (i). Thus we get a required decomposition of $\mathbf{A}$.

The following statement is obvious for arbitrary natural number $i \geqq 0$.

Lemma 4. If $\mathbf{A}$ can be embedded isomorphically into an α_i-product of $\mathbf{B}$ with a single factor and $\mathbf{B}$ can be embedded isomorphically into an α_i-product of $\mathbf{C}$ with a single factor, then $\mathbf{A}$ can be embedded isomorphically into an α_i-product of $\mathbf{C}$ with a single factor.

The next Theorem holds for α_i-products with $i \geqq 1$.

Theorem 2. A system Σ of automata is isomorphically complete for $\mathfrak{K}$ with respect to the α_i-product ($i \geqq 1$) if and only if for any prime number r there exists an automaton $\mathbf{A} \in \Sigma$ such that $\mathbf{M}_r$ can be embedded isomorphically into an α_i-product of $\mathbf{A}$ with a single factor.

Proof. To prove the sufficiency, by Lemma 4, it is enough to show that arbitrary automaton with n states can be embedded isomorphically into an α_1-product of $\mathbf{M}_r$ with a single factor for some prime $r > n$. This is trivial.

To prove the necessity take a prime r. First we prove that $\mathbf{M}_r$ can be embedded isomorphically into an α_i-product of automata from Σ with at most i factors if $\mathbf{M}_r$ can be embedded isomorphically into an α_i-product of automata from Σ. Indeed, assume that $\mathbf{M}_r$ can be embedded isomorphically into the α_i-product $\mathbf{B} = \prod_{j=1}^{k} \mathbf{A}_j(\{x_0, \ldots, x_{r-1}\}, \varphi)$ of automata from Σ with $k > i$ and denote by μ such an isomorphism. For any $l \in \{0, \ldots, r-1\}$ denote by $(a_{l1}, \ldots, a_{lk})$ the image of l under μ. We may suppose that there exist natural numbers $s \neq t$ $(0 \leqq s, t \leqq r-1)$ such that $a_{s1} \neq a_{t1}$ since in the opposite case $\mathbf{M}_r$ can be embedded isomorphically into an α_i-product of automata from Σ with $k-1$ factors. Now assume that there exist natural numbers $u \neq v$ $(0 \leqq u, v \leqq r-1)$ such that $a_{ul} = a_{vl}$ $(l = 1, \ldots, i)$. Then $\varphi_1(a_{u1}, \ldots, a_{ui}, x_j) = \varphi_1(a_{v1}, \ldots, a_{vi}, x_j)$ for any $x_j \in \{x_0, \ldots, x_{r-1}\}$. Thus in the α_i-product $\mathbf{B}$ the automaton $\mathbf{A}_1$ obtains the same input signal in the states a_{u1} and a_{v1} for any $x_j \in \{x_0, \ldots, x_{r-1}\}$. Since μ is an isomorphism thus we have that $a_{u+j(\mathrm{mod}\, r)1} = a_{v+j(\mathrm{mod}\, r)1}$ for any $j \in \{0, \ldots, r-1\}$. On the other hand, r is prime thus from the above equations we get that $a_{u1} = a_{l1}$ for any $l \in \{0, \ldots, r-1\}$ which contradicts our assumption. Therefore, we have that the elements $(a_{l1}, \ldots, a_{li})$ $(0 \leqq l \leqq r-1)$ are pairwise different. Take the following α_i-product $\mathbf{C} = \prod_{t=1}^{i} \mathbf{A}_t(\{x_0, \ldots, x_{r-1}\}, \psi)$ where for any $j \in \{1, \ldots, i\}$, $(a_1, \ldots, a_i) \in A_1 \times \ldots \times A_i$ and $x_s \in \{x_0, \ldots, x_{r-1}\}$

$$\psi_j(a_1, \ldots, a_i, x_s) = \begin{cases} \varphi_j(a_{l1}, \ldots, a_{lj+i-1}, x_s) & \text{if } j+i-1 \leqq k \text{ and there exists} \\ \quad 0 \leqq l \leqq r-1 \text{ such that } a_u = a_{lu} \quad (u = 1, \ldots, i), \\ \varphi_j(a_{l1}, \ldots, a_{lk}, x_s) & \text{if } j+i-1 > k \text{ and there exists} \\ \quad 0 \leqq l \leqq r-1 \text{ such that } a_u = a_{lu} \quad (u = 1, \ldots, i), \\ \text{arbitrary input signal from } X_j \text{ otherwise.} \end{cases}$$

It is not difficult to prove that the correspondence $\nu(l)=(a_{l1}, \ldots, a_{li})$ $(l=0, \ldots, r-1)$ is an isomorphism of $\mathbf{M}_r$ into $\mathbf{C}$.

Now we prove that if $\mathbf{M}_r$ can be embedded isomorphically into an α_i-product $\prod_{j=1}^{k} \mathbf{A}_j(\{x_0, \ldots, x_{r-1}\}, \varphi)$ of automata from Σ with $k \leqq i$, then there exists an automaton $\mathbf{A} \in \Sigma$ such that $\mathbf{M}_{\text{prime}[\sqrt[i]{r}]}$ can be embedded isomorphically into an α_i-product of $\mathbf{A}$ with a single factor, where prime $[\sqrt[i]{r}]$ denotes the largest prime less than $\sqrt[i]{r}$. Denote by μ such an isomorphism. For any $l \in \{0, \ldots, r-1\}$ denote by $(a_{l1}, \ldots, a_{lk})$ the image of l under μ. Since μ is a $1-1$ mapping thus the elements $(a_{l1}, \ldots, a_{lk})$ $(l=0, \ldots, r-1)$ are pairwise different. Therefore, there exists an s $(1 \leqq s \leqq k)$ such that the number of pairwise different elements among $a_{0s}, a_{1s}, \ldots, a_{r-1s}$ is greater than or equal to prime $[\sqrt[i]{r}]$. Let $a_{j_0 s}, \ldots, a_{j_{u-1} s}$ denote pairwise different elements, where $u=\text{prime}\,[\sqrt[i]{r}]$, and denote by $\overline{X}$ the set $\{x_0, \ldots, x_{u-1}\}$. Take the α_0-product $\mathbf{C}=\Pi \mathbf{A}_s(\overline{X}, \psi)$ with a single factor, where for any $a_{j_t s} \in \{a_{j_0 s}, \ldots, a_{j_{u-1} s}\}$ and $x_v \in \overline{X}$, $\psi(a_{j_t s}, x_v)=\varphi_s(a_{j_t 1}, \ldots, a_{j_t k}, x_d)$ if $\delta_{\mathbf{M}_r}(\mu^{-1}(a_{j_t 1}, \ldots, a_{j_t k}), x_d)=$ $=\mu^{-1}(a_{j_{t+v(\mathrm{mod}\,u)} 1}, \ldots, a_{j_{t+v(\mathrm{mod}\,u)} k})$. It is not difficult to prove that $\mathbf{M}_u$ can be embedded isomorphically into $\mathbf{C}$ which ends the proof of Theorem 2.

From Theorem 2 we get the following.

COROLLARY. A system Σ of automata is isomorphically complete for $\mathfrak{R}$ with respect to the α_i-product if and only if it is isomorphically complete with respect to the α_i-product $(i \geqq 1)$.

DEPT. OF COMPUTER SCIENCE
A. JÓZSEF UNIVERSITY
ARADI VÉRTANUK TERE 1.
SZEGED, HUNGARY
H—6720

Reference

[1] GÉCSEG, F., Composition of automata, Proceedings of the 2nd Colloquium on Automata, Languages and Programming, Saarbrücken, Springer Lecture Notes in Computer Science, v, 14, 1974, pp. 351—363.

(Received Feb. 29, 1980)

Deterministic ascending tree automata I

By J. VIRÁGH

1. Introduction

In the early 60s Automata Theory was considerably influenced by the methods and results of Universal Algebra. In fact, if we regard the input signs as unary operational symbols over the state set, then the automaton can be identified with a special universal algebra (unoid). Allowing non-unary input signs Thatcher and Wright [7] and Doner [3] came to the notion of the generalized or tree automata which accept arbitrary trees instead of the linear words of ordinary automata.

Two types of tree automata are investigated in the literature. The first one, the descending tree automata (known also as frontier-to-root or sinking automata, cf. [3], [6], [7]) proceed the input trees from the leaves along all branches towards the root. All results of the 'classical theory' such as the equivalence of the deterministic and nondeterministic devices, the minimization algorithm, Nerode's theorem, regular (tree) grammars, regular expressions and the Kleene-theorem can be generalized for this type of automata (cf. [1]—[3] and [6], [7]).

A less investigated generalization led to the notion of the ascending (called also root-to-frontier or climbing) tree automata, cf. [4], [6]. This device reads the input trees starting at the root proceeding then towards the leaves along the branches. Our investigations were inspired by the results of Magidor and Moran [6] especially by Section 6 of their paper.

Our aim is to generalize the results of the classical theory for ascending tree automata. In this part I we investigate a generalization of the Kleene-theorem and the characterization of sets accepted by ascending tree automata as sets generated by special regular tree grammars. The algebraic notations developed by Gécseg and Steinby in [4] will be used throughout this paper. Nullary operations will be excluded. This restriction is necessary by some investigations concerning ascending tree automata, cf. [4].

2. Preliminaries

For arbitrary set A, $\mathcal{P}(A)$ denotes the power set of A. N stands for the set of all positive integers, i.e., $N=\{1, 2, 3, \ldots\}$.

Let F be a finite nonvoid set and r a mapping of F into N. We call the ordered pair $\langle F, r\rangle$ a *type*. The elements of F are the *operational symbols*. If $f\in F$ and

$r(f)=n$, then we say that the arity of f is n or f is an *n-ary operational symbol.* We often refer to the type $\langle F, r\rangle$ simply by F and we take the set F as the (disjoint) union $\bigcup (F_n | n\in N)$, where F_n is the set of all n-ary operational symbols from F.

Let $X=\{x_1, x_2, \ldots\}$ be a countable set of variables and $X_n=\{x_1, x_2, \ldots, x_n\}$ for every $n\in N$. We define the set $T_{F,n}$ of *n-ary F-trees* as the smallest set satisfying

(i) $X_n\subseteq T_{F,n}$ and

(ii) $f(p_1, \ldots, p_m)\in T_{F,n}$ whenever $p_1, \ldots, p_m\in T_{F,n}$ and $f\in F_m$ for some $m\in N$.

We note that the set $T_{F,n}$ is identical with the set of all n-ary polinomial symbols of type F in the sense of Grätzer [5]. The subsets of $T_{F,n}$ are called *n-ary F-forests* or simply forests when n and F are specified by the context. T_F stands for the set $\bigcup(T_{F,n}|n\in N)$.

Next we define devices capable of recognizing forests. To this we need some preparations.

Let F be an arbitrary type. The ordered pair $\mathfrak{A}=\langle A, F^{\mathfrak{A}}\rangle$ is called a *nondeterministic F-algebra* if A is a nonempty set and $F^{\mathfrak{A}}=(f^{\mathfrak{A}}|f\in F)$ is a set of nondeterministic operations on A, i.e., if $f\in F_k$, then $f^{\mathfrak{A}}$ is a mapping

$$f^{\mathfrak{A}}\colon A^k \to \mathscr{P}(A).$$

In that special case when $f(a_1, \ldots, a_k)$ is a singleton for every $f\in F$ and $(a_1, \ldots, a_k)\in A^k$, we speak about *F-algebra.* Identifying the singletons with their elements (and that will be our practice in the following discussions) we can define an F-algebra as a system $\mathfrak{A}=\langle A, F^{\mathfrak{A}}\rangle$, where every operation is a mapping

$$f^{\mathfrak{A}}: A^k \to A.$$

The triple $\mathbf{A}=\langle\mathfrak{A}, \mathbf{a}, A'\rangle$ is called an *n-ary nondeterministic descending F-automaton* if

(i) $\mathfrak{A}=\langle A, F^{\mathfrak{A}}\rangle$ is a nondeterministic F-algebra, whose carrier A is called the *state set* of the automaton,

(ii) $\mathbf{a}=(A^{(1)}, A^{(2)}, \ldots, A^{(n)})\in(\mathscr{P}(A))^n$ is the *initial vector,*

(iii) $A'\subseteq A$ is the *set of final states.*

If $\mathfrak{A}$ is an F-algebra and all components of the initial vector are singletons, then we say that $\mathbf{A}=\langle\mathfrak{A}, \mathbf{a}, A'\rangle$ is an *n-ary deterministic descending F-automaton.*

Every automaton $\mathbf{A}$ induces a mapping $\beta^{\mathbf{A}}\colon T_{F,n}\to\mathscr{P}(A)$ in the following manner:

(i) $\beta^{\mathbf{A}}(x_j) = A^{(j)}$ $(x_j\in X_n,\quad j=1, 2, \ldots, n)$,

(ii) $\beta^{\mathbf{A}}(p) = \bigcup(f^{\mathfrak{A}}(b_1, \ldots, b_k)|b_j\in\beta^{\mathbf{A}}(p_j),\quad j=1, 2, \ldots, k)$ if $p=f(p_1, \ldots, p_k)$.

Then $T(\mathbf{A})=\{p\,|\,p\in T_{F,n}\,\&\,\beta^{\mathbf{A}}(p)\cap A'\neq\emptyset\}$ is the *forest recognized by* $\mathbf{A}$. If $\mathbf{A}$ is deterministic, then $\beta^{\mathbf{A}}(p)$ is a singleton for every $p\in T_{F,n}$ and $T(\mathbf{A})$ can be written as

$$T(\mathbf{A}) = \{p|p\in T_{F,n}\,\&\,\beta^{\mathbf{A}}(p)\in A'\}.$$

Now we present the necessary definitions for our second type of devices, the so-called ascending tree automata.

The ordered pair $\mathfrak{B}=\langle B, F^{\mathfrak{B}}\rangle$ is called a *nondeterministic ascending F-algebra* if B is a nonempty set and $F^{\mathfrak{B}}=(f^{\mathfrak{B}}|f\in F)$ is a set of nondeterministic ascending

operations on B, i.e., if $f \in F_k$, then $f^{\mathfrak{B}}$ is a mapping

$$f^{\mathfrak{B}}: B \to \mathscr{P}(B^k).$$

Again, if for all $f \in F$ and $b \in B$, $f^{\mathfrak{B}}(b)$ is a singleton, then we say that $\mathfrak{B}$ is a *deterministic ascending F-algebra* and we write the operations as mappings

$$f^{\mathfrak{B}}: B \to B^k.$$

The triple $\mathbf{B} = \langle \mathfrak{B}, B', \mathfrak{b} \rangle$ is called an *n-ary* nondeterministic ascending F-automaton, if

(i) $\mathfrak{B} = \langle B, F^{\mathfrak{B}} \rangle$ is a nondeterministic ascending F-algebra, and B is called the *state set* of **B**,

(ii) $B' \subseteq B$ is the *set of initial states,*

(iii) $b = (B^{(1)}, B^{(2)}, \ldots, B^{(n)}) \in (\mathscr{P}(B))^n$ is the *vector of final states.*

If $\mathfrak{B}$ is a deterministic ascending F-algebra and B' is a singleton, then we say that $\mathbf{B} = \langle \mathfrak{B}, B', \mathfrak{b} \rangle$ is an *n-ary deterministic ascending F-automaton.*

With every ascending automaton **B** we associate a mapping $\alpha^{\mathbf{B}}: T_{F,n} \to \mathscr{P}(B)$ as follows:

(i) $\alpha^{\mathbf{B}}(x_j) = B^{(j)}$ $(x_j \in X_n, \quad j = 1, 2, \ldots, n)$,

(ii) $\alpha^{\mathbf{B}}(p) = \{b \mid f^{\mathfrak{B}}(b) \cap \alpha^{\mathbf{B}}(p_1) \times \alpha^{\mathbf{B}}(p_2) \times \ldots \times \alpha^{\mathbf{B}}(p_k) \neq \emptyset\}$

if $f \in F_k$ and $p = f(p_1, p_2, \ldots, p_k)$.

The forest recognized by **B** is defined by

$$T(\mathbf{B}) = \{p \mid p \in T_{F,n} \,\&\, \alpha^{\mathbf{B}}(p) \cap B' \neq \emptyset\}.$$

If **B** is deterministic and $B' = \{b'\}$, then we can write simply

$$T(\mathbf{B}) = \{p \mid p \in T_{F,n} \,\&\, b' \in \alpha^{\mathbf{B}}(p)\}.$$

Our definitions are rather general because we allow even infinite state sets of automata. But this general case is needed only by some later discussions. In this Part I by automata we always mean finite automata.

Two automata, **A** and **B** are called *equivalent* if $T(\mathbf{A}) = T(\mathbf{B})$. Two class of automata, C_1 and C_2 are equivalent if for every **A** from C_1 there is a **B** from C_2, equivalent to **A**, and conversely. The following statements are well known, cf. [6], [7].

Proposition 1. The classes of deterministic descending, nondeterministic descending and nondeterministic ascending automata are equivalent. Taking an arbitrary automaton from one of these classes, we can effectively construct two automata belonging to the two other classes equivalent to it.

Let $\mathscr{F}_{\mathrm{REC}}$ denote the class of forests, recognizable by deterministic descending automata and $\mathscr{F}_A$ the class, recognizable by deterministic ascending automata.

Proposition 2. $\mathscr{F}_A \subsetneq \mathscr{F}_{\mathrm{REC}}$.

Since forests are subsets of $T_{F,n}$, we may define the usual set-theoretic operations $\cup$ (union), $\cap$ (intersection) and $^-$ (complementation) on them. Let us now define two more operations the x_i-product and x_i-iteration.

The x_i-*product* of the forests T_1 and T_2, denoted by $T_1 \cdot_{x_i} T_2$, is the forest which can be obtained by replacing every occurence of x_i in some tree from T_1 by a tree from T_2. The x_i-*iteration* of the forest T, denoted by T^{*x_i}, is defined by $T^{*x_i} = \bigcup (T^{k,x_i} | k=0, 1, 2, \ldots)$ where

(i) $T^{0,x_i} = x_i$,

(ii) $T^{n,x_i} = T^{n-1,x_i} \cup T \cdot_{x_i} T^{n-1,x_i}$

or $n \in N$. We refer to the operations union, x_i-product and x_i-iteration as *regular operations*. It is well known that $\mathscr{F}_{\text{REC}}$ is closed under the regular operations, but $\mathscr{F}_A$ is not.

Let us take some language $\mathscr{L}$ suitable for describing the sets accepted by automata. In Automata Theory the following two problems play an important role:

(1) Given an automaton A. Describe the set accepted by A in terms of $\mathscr{L}$.

(2) Given a description of the set T in the language $\mathscr{L}$. Construct an automaton accepting T.

The solution is given by the famous Kleene-theorem if $\mathscr{L}$ is the language of regular expressions. Next we review briefly the generalization of this theorem for deterministic descending automata. First of all, we have to define the language $\mathscr{L}$ of (generalized) regular expressions.

Let F be an arbitrary type. The set $\mathscr{R}_F$ of *regular F-expressions* is the smallest set for which

(i) $\emptyset \in \mathscr{R}_F$,

(ii) if $p \in T_F$, then $p \in \mathscr{R}_F$, and

(iii) if $\mathscr{K}_1, \mathscr{K}_2 \in \mathscr{R}_F$ and $i \in N$, then $(\mathscr{K}_1 + \mathscr{K}_2)$, $(\mathscr{K}_1 i \mathscr{K}_2)$, $(\mathscr{K}_1)^{*i} \in \mathscr{R}_F$ hold.

If F is known from the context, we speak about regular expressions simply. An occurrence of the variable x_i in $\mathscr{K}$ is called *bounded* if this occurrence is in $\mathscr{K}_1$ for a subexpression $\mathscr{K}_1 i \mathscr{K}_2$ of $\mathscr{K}$. All other occurrences of x_i are called *free*. x_i is a *free variable* of $\mathscr{K}$ if x_i has at least one free occurrence in $\mathscr{K}$. $\mathscr{K}$ is an *n-regular expression* if all its free variables are in X_n.

Each regular expression $\mathscr{K} \in \mathscr{R}_F$ denotes a forest $|\mathscr{K}|$ given by the following rules:

(i) if $\mathscr{K} = \emptyset$, then $|\mathscr{K}|$ is the empty forest,

(ii) if $\mathscr{K} = p (p \in T_F)$, then $|\mathscr{K}| = \{p\}$,

(iii) if $\mathscr{K} = (\mathscr{K}_1 + \mathscr{K}_2)$, then $|\mathscr{K}| = |\mathscr{K}_1| \cup |\mathscr{K}_2|$,

(iv) if $\mathscr{K} = (\mathscr{K}_1 i \mathscr{K}_2)$, then $|\mathscr{K}| = |\mathscr{K}_1| \cdot_{x_i} |\mathscr{K}_2|$,

(v) if $\mathscr{K} = (\mathscr{K}_1)^{*i}$, then $|\mathscr{K}| = |\mathscr{K}_1|^{*x_i}$.

T is a n *n-regular forest* if $T = |\mathscr{K}|$ for some n-regular expression $\mathscr{K}$. Moreover, T is a *regular forest* if T is n-regular for some $n \in N$. Finally, $\mathscr{F}_{\text{REG}}$ stands for the class of all regular forests.

Proposition 3. (Kleene-theorem, cf. [7]) $\mathscr{F}_{\mathrm{REG}}=\mathscr{F}_{\mathrm{REC}}$. More precisely, for an arbitrary deterministic descending automaton one can effectively construct a regular expression denoting the forest accepted by this automaton, and conversely.

Regular tree grammars are direct generalizations of the well known regular (string) grammars. The following definition is equivalent to the usual one, cf. [2].

Let F be a type, a *regular F-grammar* is a system $\Gamma=\langle Q, F, P, S\rangle$, where

(i) Q is a finite nonempty *set of nonterminal symbols,*

(ii) $S\subseteq Q$ is the *set of initial symbols,*

(iii) P is a finite *set of rewriting rules* of the form

$$q\to x_i\ (q\in Q,\ x_i\in X)\quad\text{or}\quad q\to f(q_1,\ldots,q_k)\ (q, q_1,\ldots,q_k\in Q,\ f\in F_k).$$

If all variables occuring in the rules of P are from X_n, then we say that Γ is an *n-ary regular F-grammar.* When n and F are not specified, we speak about *regular (tree) grammars.* The n-ary regular F-grammar Γ induces a binary relation $\Rightarrow_\Gamma$ on the set $T_{F,Q\cup X_n}$, $t\Rightarrow_\Gamma r$ iff r can be obtained from t by replacing a nonterminal q in t with the right side of some rule $q\to f(q_1,\ldots,q_k)$ or $q\to x_i$ from P. Let $\Rightarrow_\Gamma^*$ denote the reflexive, transitive closure of the relation $\Rightarrow_\Gamma$.

The set $T(\Gamma)=\{p\,|\,p\in T_{F,n}\,\&\,(\exists s)(s\in S\,\&\,s\Rightarrow_\Gamma^* p)\}$ is the *forest generated by* Γ. T is called *generable,* if $T=T(\Gamma)$ for some regular grammar Γ. $\mathscr{F}_{\mathrm{GEN}}$ denotes the class of all generable forests.

Proposition 4. (Brainerd [2]) $\mathscr{F}_{\mathrm{REC}}=\mathscr{F}_{\mathrm{GEN}}$.

3. Closed forests

In this section we show that every forest $T\in\mathscr{F}_A$ contains all trees composed from the paths of some trees belonging to T. We shall use this characteristic feature for deriving a connection between $\mathscr{F}_{\mathrm{REC}}$ and $\mathscr{F}_A$.

With every type F we associate a new type $\bar{\delta}(F)$ of unary operational symbols in the following way:

(i) if $f\in F_k$, then $\bar{\delta}(f)=\{f_1, f_2,\ldots, f_k\}$,

(ii) if $f\neq g$, then $\bar{\delta}(f)\cap\bar{\delta}(g)=\emptyset$,

(iii) $\bar{\delta}(F)=\bigcup(\bar{\delta}(f)|f\in F)$.

Now let us define the functions $\delta_i\colon \mathscr{P}(T_F)\to\mathscr{P}(T_{\bar{\delta}(F)})$ for all $i\in N$ as follows

(1) $\delta_i(x_k)=\begin{cases} x_k & \text{if } i=k,\\ \emptyset & \text{otherwise,}\end{cases}$

(2) $\delta_i(f(p_1,\ldots,p_k))=f_1(\delta_i(p_1))\cup f_2(\delta_i(p_2))\cup\ldots\cup f_k(\delta_i(p_k))$,

(3) $\delta_i(T)=\bigcup(\delta_i(t)|t\in T)$.

Let $\delta\colon \mathscr{P}(T_{F,n})\to\mathscr{P}(T_{\bar{\delta}(F),n})$ be the function for which

$$\delta(T)=\bigcup(\delta_i(T)|i=1, 2,\ldots, n).$$

Speaking informally, $\delta(t)$ consist of all words which can be read along the paths of t. The inserted indices show the position of the node to be visited next. The elements of $\delta(T)$ can be regarded as words of the free semigroup generated by $\delta(F)\cup X_n$ as well.

Lemma 1. The function δ is monotone and commutes with the regular operations, i.e.

(i) if $T_1 \subseteq T_2$, then $\delta(T_1) \subseteq \delta(T_2)$,

(ii) $\delta(T_1 \cup T_2) = \delta(T_1) \cup \delta(T_2)$,

(iii) $\delta(T_1 \cdot_{x_i} T_2) = \delta(T_1) \cdot_{x_i} \delta(T_2)$,

(iv) $\delta(T^{*x_i}) = (\delta(T))^{*x_i}$.

Proof. (i) and (ii) are obvious. For verifying (iii), first we show the inclusion $\delta(T_1) \cdot_{x_i} \delta(T_2) \subseteq \delta(T_1 \cdot_{x_i} T_2)$. If $g \in \delta(T_1) \cdot_{x_i} \delta(T_2)$, then we must distinguish the following two cases:

1) $g \in \delta_j(T_1)$ and $j \neq i$. This directly implies $g \in \delta_j(T_1 \cdot_{x_i} T_2)$. Therefore, $g \in \delta(T_1 \cdot_{x_i} T_2)$.

2) $g = g_1 \cdot_{x_i} g_2$ where $g_1 \in \delta_i(t_1)$ for some $t_1 \in T_1$ and $g_2 \in \delta_j(t_2)$ for some $t_2 \in T_2$. Then $g \in \delta_j(t_3)$ holds for the tree $t_3 = t_1 \cdot_{x_i} t_2 \in T_1 \cdot_{x_i} T_2$. Thus $g \in \delta(T_1 \cdot_{x_i} T_2)$. The inclusion $\delta(T_1 \cdot_{x_i} T_2) \subseteq \delta(T_1) \cdot_{x_i} \delta(T_2)$ can be verified in a similar way.

Using (ii), (iii) and the identity $T^{n,x_i} = T^{n-1,x_i} \cup T \cdot_{x_i} T^{n-1,x_i}$ it is easy to prove by induction on n that (iv) holds, too. □

REMARK. From Lemma 1 it follows that the regularity of T implies the regularity of $\delta(T)$. The converse is not true. For this it is enough to consider the forest T of all balanced trees in $T_{F,n}$.

Let $\delta^{-1}: \mathscr{P}(T_{\delta(F),n}) \to \mathscr{P}(T_{F,n})$ denote the inverse of δ, i.e., $\delta^{-1}(U) = \{t \mid \delta(t) \subseteq U\}$ for every $U \subseteq T_{\delta(F),n}$. We define the operator $\Delta: \mathscr{P}(T_{F,n}) \to \mathscr{P}(T_{F,n})$ as the composition $\delta^{-1} \cdot \delta$

$$\Delta(T) = \{t \mid \delta(t) \subseteq \delta(T)\} \quad (T \subseteq T_{F,n}).$$

Lemma 2. Δ is an algebraic closure operator on $\mathscr{P}(T_{F,n})$, that is

(i) $T_1 \subseteq T_2 \Rightarrow \Delta(T_1) \subseteq \Delta(T_2)$,

(ii) $T \subseteq \Delta(T)$,

(iii) $\Delta(T) = \Delta(\Delta(T))$,

(iv) if $t \in \Delta(T)$, then $t \in \Delta(T_1)$ for some finite $T_1 \subseteq T$.

Proof. Obvious. □

We say that $T \subseteq T_{F,n}$ is *Δ-closed* if $\Delta(T) = T$. Let $\mathscr{F}_c$ denote the class of all Δ-closed forests.

Lemma 3. If F has at least one non-unary operational symbol then $\mathscr{F}_c$ and $\mathscr{F}_{\mathrm{REC}}$ are incomparable.

Proof. For the sake of simplicity assume that $F_2 \neq \emptyset$ and $f \in F_2$. Let

$$T_1 = \{f(x_1, f(x_1, x_1)), f(f(x_1, x_1), x_1)\}$$

and

$$T_2 = T_3 \cap T_4, \text{ where}$$

$T_3 = T(\Gamma)$, $\Gamma = \langle \{S, R\}, \{f\}, \{S \to f(R, S), S \to x_1, R \to x_1\}, S\rangle$

and

$T_4 = \{p | p$ is composed from an arbitrary number of f's and a prime number of x_1's$\}$.

$T_1 \in \mathscr{F}_{\text{REC}}$ is obvious, but $T_1 \notin \mathscr{F}_c$ because of $f(x_1, x_1) \in \Delta(T_1)$ and $f(x_1, x_1) \notin T_1$. On the other hand $T_2 \in \mathscr{F}_c$, but $T_2 \in \mathscr{F}_{\text{REC}}$ would lead to a contradiction. □

Now we generalize the construction of the well-known powerset automaton for ascending tree automata. Let $\mathbf{A} = \langle \mathfrak{A}, A', \mathbf{a}\rangle$ be an n-ary nondeterministic ascending F-automaton. The *powerset automaton,* belonging to $\mathbf{A}$ is the n-ary deterministic ascending F-automaton $\mathbf{PA} = \langle \mathfrak{P}\mathfrak{A}, A', \mathbf{b}\rangle$, where $\mathfrak{P}\mathfrak{A} = \langle \mathscr{P}(A), F\rangle$ is the deterministic ascending F-algebra with operations $f^{\mathfrak{P}\mathfrak{A}}$ defined by

$$f^{\mathfrak{P}\mathfrak{A}}(C) = \prod_{i=1}^{k} \left(\bigcup (\pi_i(f^{\mathfrak{A}}(c)) \mid c \in C)\right)$$

for every $C \subseteq A$ and $f \in F_k$, $\mathbf{b} = (B^{(1)}, B^{(2)}, \ldots, B^{(n)})$ with $B^{(i)} = (D \mid D \subseteq A \,\&\, D \cap A^{(i)} \neq \emptyset)$ and π_i denotes the ith projection.

Now we recall some concepts from [4]. For any state a of the n-ary nondeterministic ascending F-automaton $\mathbf{A}$ we define

$$T(\mathbf{A}, a) = \{p | p \in T_{F,n} \,\&\, a \in \alpha^{\mathbf{A}}(p)\}.$$

A state a is called a 0-*state* if $T(\mathbf{A}, a) = \emptyset$. We say that $\mathbf{A}$ is *normalized* if, for all $a \in A$, $n \in N$ and $f \in F_n$ either all of the components of $f^{\mathfrak{A}}(a)$ are 0-states or none of them is a 0-state.

Lemma 4. For any nondeterministic ascending automaton $\mathbf{A}$ an equivalent normalized nondeterministic ascending automaton $\mathbf{A}^*$ can be constructed.

Proof. This lemma is a generalization of Theorem 3 in [4] for nondeterministic automata. The proof can be performed similarly. □

Lemma 5. For every normalized nondeterministic ascending automaton $\mathbf{A}$, $T(\mathbf{PA}) = \Delta(T(\mathbf{A}))$.

Proof. We shall verify the inclusion $T(\mathbf{PA}) \subseteq \Delta(T(\mathbf{A}))$ first. Let $t \in T(\mathbf{PA})$ and $g \in \delta_j(t)$ for some $1 \leq j \leq n$. It follows from the definition of $T(\mathbf{PA})$, that in this case we can correspond to the branches of g a sequence $a_0, a_1, \ldots, a_v$ of states from A such that

(i) $a_k \in A$ $(0 \leq k \leq v)$, $a_0 \in A'$ and $a_v \in A^{(j)}$,

(ii) $a_{i+1} \in \pi_k(f^{\mathfrak{A}}(a_i))$ if a_i $(0 \leq i \leq v-1)$ corresponds to the branch labelled by f_k.

We can complete g to a tree $\bar{t}$ accepted by $\mathbf{A}$ because $\mathbf{A}$ is normalized. Thus $g \in \delta_j(\bar{t}) \subseteq \delta_j(T(\mathbf{A}))$ which implies $t \in \Delta(T(\mathbf{A}))$ since g and t were arbitrary.

The reverse inclusion can be verified in a similar manner. □

REMARK. We show by a simple counterexample that the preceding Lemma does not hold for unnormalized automata. Let $\mathbf{A}=\langle \mathfrak{A}, A', \mathbf{a}\rangle$ be the automaton where

$$A = \{a_0, d\}, \quad F = F_2 = \{f\}, \quad n = 1,$$
$$f^{\mathfrak{A}}(a_0) = \{(d, a_0), (a_0, d)\}, \quad f^{\mathfrak{A}}(d) = \{(d, d)\}$$
$$A' = A^{(1)} = \{a_0\}.$$

Then $f(x_1, x_1)$ is in $T(\mathbf{PA})$, but not in $\Delta(T(\mathbf{A}))$. □

If we apply the construction of **PA** for a *deterministic* automaton **A**, we get an automaton equivalent to **A**. From this assertion, using Lemma 5, it directly follows.

Corollary 6. The regular forest T can be recognized by a deterministic ascending automaton iff T is closed.

Corollary 7. It is effectively decidable for every regular forest T whether T is recognizable by a deterministic ascending automaton.

Proof. Let $T=T(\mathbf{A})$, where **A** is a nondeterministic normalized ascending automaton. In this case, by Corollary 6, $T\in\mathscr{F}_A$ iff $T(\mathbf{A})=T(\mathbf{PA})$. By Proposition 2.1 we can construct two deterministic descending automata equivalent to **A** and **PA**, respectively. For this type of automata the equivalence problem is decidable. □

4. D-regular operations and the generalized Kleene-theorem

It can easily be seen that $\mathscr{F}_A$ is not closed under the regular operations. In fact, it is not closed under the polinomials of these operations, either. More precisely, this is true for almost all polinomials but those unary ones constructed by unions only. Since $\mathscr{F}_A$ can be obtained as the Δ-closure of $\mathscr{F}_{\mathrm{REG}}$ it seems reasonable to define 'D-regular operations' by combining Δ and the regular operations. This enables us to derive a 'Kleene theorem' for $\mathscr{F}_A$.

Now let us define the *D-regular operations* for any T_1 and T_2 from T_F as follows:

(1) Δ-union $\widehat{\cup}$: $T_1\widehat{\cup}T_2=\Delta(T_1\cup T_2)$,
(2) (Δ, x_i)-product $\odot_{x_i}$: $T_1\odot_{x_i}T_2=\Delta(T_1\cdot_{x_i}T_2)$,
(3) (Δ, x_i)-iteration $\widehat{*}_{x_i}$: $(T)\widehat{*}x_i=\Delta(T^*x_i)$,

where on the right sides $\cup$, $\cdot_{x_i}$ and $*_{x_i}$ stand for the ordinary regular operations.

Lemma 1. The D-regular operations preserve regularity and recognizability by deterministic ascending automata.

Proof. Follows from Proposition 2.3 and Corollary 3.6. □

Lemma 2. For D-regular operations the following identities hold

(i) $T_1\widehat{\cup}T_2 = \Delta(T_1)\widehat{\cup}\Delta(T_2)$,
(ii) $T_1\odot_{x_i}T_2 = \Delta(T_1)\odot_{x_i}\Delta(T_2)$,
(iii) $(T)\widehat{*}x_i = (\Delta(T))\widehat{*}x_i$.

Proof. These identities can be derived applying the function δ^{-1} for both sides of the equations

$$(*) \qquad \delta(T_1 \cup T_2) = \delta(\Delta(T_1) \cup \Delta(T_2)),$$

$$(**) \qquad \delta(T_1 \cdot_{x_i} T_2) = \delta(\Delta(T_1) \cdot_{x_i} \Delta(T_2)),$$

$$(***) \qquad \delta((T)^{*x_i}) = \delta(\Delta(T)^{*x_i}).$$

We know from Lemma 3.1 that δ commutes with the regular operations. It is also easily seen that $\delta(\Delta(T))=\delta(T)$ holds for every forest T. These assertions imply the identities $(*)$—$(***)$. □

Next we introduce a new '*D-regular interpretation*' $\|\mathscr{K}\|$ of the regular expression $\mathscr{K}$

(i) if $\mathscr{K} = \emptyset$, then $\|\mathscr{K}\|$ is the empty forest,

(ii) if $\mathscr{K} = p(p \in T_F)$, then $\|\mathscr{K}\| = \{p\}$,

(iii) if $\mathscr{K} = (\mathscr{K}_1 + \mathscr{K}_2)$, then $\|\mathscr{K}\| = \|\mathscr{K}_1\| \widehat{\cup} \|\mathscr{K}_2\|$,

(iv) if $\mathscr{K} = (\mathscr{K}_1 i \mathscr{K}_2)$, then $\|\mathscr{K}\| = \|\mathscr{K}_1\| \odot_{x_i} \|\mathscr{K}_2\|$,

(v) if $\mathscr{K} = (\mathscr{K}_1)^{*i}$, then $\|\mathscr{K}\| = \|\mathscr{K}_1\|^{\widehat{*}}{}_{x_i}$.

T is a *D-regular forest* if $T=\|\mathscr{K}\|$ for some regular expression $\mathscr{K}$.

Lemma 3. For every regular expression $\mathscr{K}$, $\|\mathscr{K}\| = \Delta(|\mathscr{K}|)$.

Proof. By induction on the number of the symbols of regular operations in $\mathscr{K}$ using Lemma 2. □

Theorem 4. The forest T is recognizable by deterministic ascending automata iff T is *D*-regular, and this connection is effective.

Proof. (1) Let T be given by the deterministic ascending automaton **A**. According to Proposition 2.1 and 2.3 we can effectively construct a deterministic descending automaton **B** and a regular expression $\mathscr{K}$ such that

$$T = T(\mathbf{A}) = T(\mathbf{B}) = |\mathscr{K}|.$$

T is closed (see Corollary 3.6) thus $T=\Delta(T)=\Delta(|\mathscr{K}|)$. But this yields, by Lemma 3, $T=\|\mathscr{K}\|$.

(2) Now let us assume that $T=\|\mathscr{K}\|$ for the regular expression $\mathscr{K}$. By Proposition 2.1 and 2.3 we can effectively construct a nondeterministic ascending automaton **C** accepting $|\mathscr{K}|$. **C** can be assumed to be normalized (see Lemma 3.4). Proceeding as in Lemma 3.5 we get the deterministic ascending powerset automaton **PC** for which $T(\mathbf{PC})=\Delta(T(\mathbf{C}))=\Delta(|\mathscr{K}|)=\|\mathscr{K}\|$ holds. □

REMARK. In the preceding proof we used Proposition 2.3 in both directions. Theorem 4 could be proved without it, but in that case the proof would be more lengthy and difficult.

5. D-regular tree grammars

In Proposition 2.4 we have established a close connection between forests generable by regular tree grammars and forests recognizable by deterministic descending automata. In this section we shall give a similar characterization for forests accepted by deterministic ascending automata. To this we define a special kind of regular tree grammars.

The regular tree grammar $\Gamma=\langle Q, F, P, S\rangle$ is called *deterministic regular*, or briefly *D-regular* if

(i) S is a singleton and

(ii) for every nonterminal q and operational symbol f there is exactly one derivation rule in P, whose left side is q and whose right side begins with f.

Theorem 1. The forest T is recognizable by deterministic ascending automata iff T is generable by D-regular tree grammars.

Proof. The well-known constructions of converting a regular tree grammar into an equivalent (nondeterministic) ascending automaton and vice versa can be used. The only thing to be noted is that the assumptions (i) and (ii) in the definition guarantee the preservation of the determinism in both directions. □

Corollary 2. For every regular tree grammar Γ one can decide effectively whether $T(\Gamma)$ can be generated by D-regular tree grammars.

Proof. Since a nondeterministic ascending automaton accepting $T(\Gamma)$ can effectively be constructed (cf. Proposition 2.1 and [2]) Corollary 3.6 and Theorem 1 immediately imply our Corollary. □

Acknowledgement. The author wishes to thank Professor M. Steinby for his valuable suggestions especially for those concerning Theorem 4.4 and for providing the example for the Remark in Section 3.

DEPT. OF COMPUTER SCIENCE
A. JÓZSEF UNIVERSITY
ARADI VÉRTANUK TERE 1.
SZEGED, HUNGARY
H—6720

References

[1] BRAINERD, W. S., The minimization of tree-automata, *Inform. and Control*, v. 13, 1968, pp. 484—491.

[2] BRAINERD, W. S., Tree generating regular systems, *Inform. and Control*, v. 14, 1969, pp. 217—231.

[3] DONER, J., Tree acceptors and some of their applications, *J. Comput. System. Sci.*, v. 4, 1970, pp. 406—451.

[4] GÉCSEG, F. and M. STEINBY, Minimal ascending tree automata, *Acta Cybernet.*, v. 4, 1978, pp. 37—44.

[5] GRÄTZER, G., *Universal algebra*, Van Nostrand, Princeton, N. J., 1968.

[6] MAGIDOR, M. and G. MORAN, Finite automata over finite trees, *Tech. Rep. Hebrew Univ.*, Jerusalem, No. 30, 1969.

[7] THATCHER, J. W. and J. B. WRIGHT, Generalized finite automata theory with an application to a decision problem of second order logic, *Math. Systems Theory*, v. 2, 1968, pp. 57—81.

(Received Oct. 24, 1979)

Iterated grammars

By L. CSIRMAZ

1. Notations and definitions

1.1. Let Σ be any, finite or infinite, set. Σ^* denotes the set of finite sequences of elements of Σ including the empty sequence which is denoted by ε. Σ^+ stands for $\Sigma^*-\{\varepsilon\}$. If $\alpha\in\Sigma^*$ then $|\alpha|$ is the length of the sequence, in particular $|\varepsilon|=0$. The elements of Σ^* are called words. The mirror image of a word α is denoted by α^{-1}.

If Σ is finite we refer it as an alphabet. The subsets of Σ^* are the languages over Σ.

If Σ does not contain the comma symbol, we define Σ^s as the set of sequences of elements of Σ separated by commas. For example, if $\Sigma=\{ab, a, b\}$ then "a, b, ab" and "ab" are elements of Σ^s. Clearly $\Sigma^*\cap\Sigma^s=\Sigma\cup\{\varepsilon\}$. If $\alpha\in\Sigma^s$ then $\|\alpha\|$ denotes the length of the sequence, i.e. the number of commas in α plus one. For example $\|a, b, ab\|=3$, $\|ab\|=1$, $\|\varepsilon\|=0$ but $|a, b, ab|=6$.

1.2. A grammar or metagrammar is a 4-tuple $\mathcal{G}=\langle N, T, P, S\rangle$ where N and T are disjoint finite sets of nonterminal and terminal symbols, respectively, P is a finite set of production rules of the form $\alpha\to\beta$ where $\alpha\in N^+$, $\beta\in(N\cup T)^*$, and $S\in N$ is the starting symbol. $\mathcal{L}(\mathcal{G})$ denotes the language generated by $\mathcal{G}$.

Grammars are classified by the structure of their production rules as it can be seen in Table 1 below. The language $L\subset T^*$ is of type τ $(=0, 1, 2, 3)$ if there is a grammar of type τ generating L. The family of languages of type τ is denoted by $\chi(\tau)$.

$\mathcal{G}$ is of type	if $\alpha\to\beta\in P$ implies
0 (phrase structure)	anyway
1 (context sensivite)	S does not occur in β and either $\|\alpha\|\leqq\|\beta\|$ or $\alpha=S$ and $\beta=\varepsilon$
2 (context free)	$\|\alpha\|=1$
3 (regular)	$\|\alpha\|=1$ and either $\|\beta\|\leqq 1$ or β is of the form tn where $t\in T$ and $n\in N$

Table 1.

1.3. An iterated grammar is a 5-tuple $\mathscr{I}=\langle \Sigma, N, T, P, S\rangle$ where Σ and T are disjoint finite sets none of them containing the symbols $\Rightarrow$ and , (double arrow and comma). $N\subseteq \Sigma^+$ is the set of nonterminal symbols, T is the set of terminal symbols. P is the set of production rules of the form $\alpha\Rightarrow\beta$ where $\alpha\in N^s-\{\varepsilon\}$, $\beta\in(N\cup T)^s$ and $S\in N$ is the starting symbol. The sets N and P may be infinite. The language $\mathscr{L}(\mathscr{I})$ generated by the iterated grammar $\mathscr{I}$ is a subset of T^* the elements of which can be derived from S in the usual way using finitely many production rules only. During the derivation the commas serve as separators between the symbols but they are abandoned at the end.

Iterated grammars are classified also as Table 2 shows.

$\mathscr{I}$ is of type	if $\alpha\Rightarrow\beta\in P$ implies
0	anyway
1	S does not occur in β and either $\|\|\alpha\|\|\leqq\|\|\beta\|\|$ or $\alpha=S$ and $\beta=\varepsilon$
2	$\|\|\alpha\|\|=1$
3	$\|\|\alpha\|\|=1$ and either $\|\|\beta\|\|\leqq 1$ or β is of the form t, w where $t\in T$ and $w\in N$

Table 2.

The iterated grammar $\mathscr{I}=\langle \Sigma, N_2, T_2, P_2, S_2\rangle$ is said to be generated by the metagrammar $\mathscr{G}=\langle N_1, T_1, P_1, S_1\rangle$ if

$$\Sigma = T_1-T_2\neq\emptyset,\quad N_2=\Sigma^+,\quad T_2\subseteq T_1,\quad P_2=\mathscr{L}(\mathscr{G}).$$

An iterated grammar is of type (σ, τ) if it is of type τ and there is a metagrammar of type σ which generates it. A language L is of type (σ, τ) if there is an iterated grammar of type (σ, τ) generating L. The family of languages of type (σ, τ) is denoted by $\chi(\sigma, \tau)$.

2. The theorems

Because every finite language is regular, and $\chi(\tau)\subseteq\chi(\tau')$ if $\tau\geqq\tau'$ we have the following

PROPOSITION. If $\sigma\geqq\sigma'$ and $\tau\geqq\tau'$ then

$$\chi(\tau)\subseteq\chi(3,\tau)\subseteq\chi(\sigma,\tau)\subseteq\chi(\sigma',\tau')\subseteq\chi(0).$$

Theorem 1. $\chi(3,\tau)=\chi(\tau)$ for $\tau=0, 1, 2, 3$.

Proof. For $\tau=0$ the Proposition implies the statement. For the other cases first we need a

LEMMA. Let $L\subseteq(T\cup\{a\})^*$ be a regular language, $a\notin T$. Then there is a finite set R, a regular language $K\subseteq R^*$ and regular languages $K_b\subseteq T^*$ for every $b\in R$ such that

$$L=\{w_1 a w_2 a\dots a w_n:\ w_i\in K_{b_i}\text{ and }b_1 b_2\dots b_n\in K\}.$$

REMARK. The converse of the Lemma is evidently true, i.e. if K and the K_b's are regular languages then L is regular, too.

Proof of the lemma. It is well-known (see, e.g., [1]) that $L-\{\varepsilon\}$ can be generated by a regular grammar $\mathcal{G}=\langle N, T\cup\{a\}, P, S\rangle$ where P consists of rules of the form $A\to x$ and $A\to xB$ only ($A, B\in N$, $x\in T\cup\{a\}$). Now define $P_0, P_1, Q_0, Q_1\subseteq P$ as follows.

$$P_0 = \{\alpha\in P\colon \alpha = A\to aB \text{ for some } A, B\in N\},$$

$$P_1 = \{\alpha\in P\colon \alpha = A\to xB \text{ for some } A, B\in N,\ x\in T\},$$

$$Q_0 = \{\alpha\in P\colon \alpha = A\to a \text{ for some } A\in N\},$$

$$Q_1 = \{\alpha\in P\colon \alpha = A\to x \text{ for some } A\in N,\ x\in T\}.$$

Obviously, $P=P_0\cup P_1\cup Q_0\cup Q_1$. Let s and f be two new symbols (for start and finish) and define

$$R = \{\langle\alpha,\beta\rangle\colon \alpha,\beta\in P_0\cup Q_0\}\cup\{\langle s,\beta\rangle\colon \beta\in P_0\cup Q_0\}\cup\{\langle\alpha, f\rangle\colon \alpha\in P_0\cup Q_0\}\cup\{\langle s, f\rangle\}.$$

The languages $K_{\langle\alpha,\beta\rangle}$ for $\langle\alpha,\beta\rangle\in R$ will be the "cuts" starting after symbol a generated by the rule α and ending before the next symbol a generated by the rule β. We need two more definitions. For $A\in N$ let

$$P_0(A) = \begin{cases}\{A\to\varepsilon\} & \text{if } A\to aB\in P_0 \text{ for some } B\in N \text{ or } A\to a\in Q_0,\\ \emptyset & \text{otherwise,}\end{cases}$$

$$P_1(A) = P_1\cup\{B\to x\colon B\to xA\in P_1\}.$$

Now we are ready to define the languages $K_{\langle\alpha,\beta\rangle}$ for all $\langle\alpha,\beta\rangle\in R$. If $\alpha\in Q_0$, $\beta\in P_0\cup Q_0$ then let $K_{\langle\alpha,\beta\rangle}=\emptyset$, $K_{\langle\alpha, f\rangle}=\{\varepsilon\}$. If $\alpha=A\to aB\in P_0$ and either $\beta=C\to aD\in P_0$ or $\beta=C\to a\in Q_0$ then $K_{\langle\alpha,\beta\rangle}$ is the language generated by the grammar $\langle N, T, P_0(B)\cup P_1(C), B\rangle$, $K_{\langle s,\beta\rangle}$ is the language generated by $\langle N, T, P_0(S)\cup P_1(C), S\rangle$ and $K_{\langle\alpha, f\rangle}$ is generated by $\langle N, T, P_0(B)\cup P_1\cup Q_1, B\rangle$. Finally, $K_{\langle s, f\rangle}$ is the language generated by $\langle N, T, P_1\cup Q_1, S\rangle$ plus the empty word if it was also in L.

What remained is to define the language K. It is the one which is generated by the grammar

$$\langle P_0\cup Q_0\cup\{s, f\}, R, P^K, s\rangle$$

where

$$P^K = \{\alpha\to\langle\alpha,\beta\rangle\beta\colon \langle\alpha,\beta\rangle\in R,\ \beta\neq f \text{ and } K_{\langle\alpha,\beta\rangle}\neq\emptyset\}\cup$$

$$\cup\{\alpha\to\langle\alpha, f\rangle\colon \langle\alpha, f\rangle\in R \text{ and } K_{\langle\alpha, f\rangle}\neq\emptyset\}.$$

It is easy to check that R, K and the K_b's satisfy the requirements. □

Now we return to the proof of the Theorem 1. Let P be the regular set of production rules of the iterated grammar $\mathcal{I}=\langle\Sigma, N, T, P, S\rangle$. In this case $P\subseteq(\Sigma\cup T\cup\{\Rightarrow\}\cup\{\ ,\ \})^*$ and neither the double arrow nor the comma is an element of $\Sigma\cup T$. The double arrow must occur exactly once in every production rule, so, by the Lemma, there are regular languages P_j^L and P_j^R over $\Sigma\cup T\cup\{\ ,\ \}$ such that P is the finite union of languages

$$\{w_1\Rightarrow w_2\colon w_1\in P_j^L,\ w_2\in P_j^R\}.$$

Applying the Lemma to the languages P_j^L and P_j^R with the comma as the special terminal symbol, we get languages K_j^L and K_j^R over disjoint alphabets R_j^L and R_j^R for each j, and finitely many regular languages K_i over $\Sigma \cup T$ indexed by the elements of $I = \bigcup_j (R_j^L \cup R_j^R)$. To be more precise the K_i's are subsets of $\Sigma^+ \cup T$.

For $w_1, w_2 \in \Sigma^+$ define the relation $w_1 \equiv w_2$ as $w_1 \in K_i \leftrightarrow w_2 \in K_i$ for all $i \in I$. It is clear that this is an equivalence relation and there are finitely many equivalence classes (no more than 2^k where k is the cardinality of I). The definition of equivalence means that if $\alpha \in P$ is a production rule, $w_1 \in \Sigma^+$ is a nonterminal symbol in it and $w_1 \equiv w_2$ then putting w_2 in places of the nonterminal occurences of w_1 in α the resulting word is in P, too. Therefore every derivation can be rewritten so that it contains at most one element from each equivalence class, i.e. only finitely many different nonterminal symbols are used. It means that the languages K_i can be assumed to be finite, or, equivalently, to have one element. This element will be denoted by $\mu(i)$.

We now have finitely many regular languages K_j^L and K_j^R over the finite set I, and a function $\mu\colon I \to (N \cup T)$ such that $S \in \text{range}(\mu)$. The set of production rules was reduced to the finite union of sets

$$\{w_1 \Rightarrow w_2\colon\ w_1 \in P_j^L,\ w_2 \in P_j^R\}$$

where

$$P_j^L = \{\mu(i_1), \mu(i_2), \ldots, \mu(i_n)\colon\ i_1 i_2 \ldots i_n \in K_j^L\},$$

$$P_j^R = \{\mu(i_1), \mu(i_2), \ldots, \mu(i_n)\colon\ i_1 i_2 \ldots i_n \in K_j^R\}.$$

Our next aim is to show that the K_j^L's are finite languages. If not, there are arbitrary long elements in K_j^L, i.e. fixing some $w_2 \in P_j^R$ there is an $x \in K_j^L$ such that $|x| > \|w_2\| + 1$. Let $w_1 \in P_j^L$ be the word belonging to x. Then $\|w_1\| = |x| > \|w_2\| + 1$ which contradicts the assumption that $\mathscr{G} \in \chi(3, \tau)$ with $\tau > 0$.

If in the languages K_j^R we replace $i \in I$ by $\mu(i)$ if $\mu(i) \in T$ then the following set of production rules

$$Q = \bigcup_j \{w_1 \to w_2\colon\ w_1 \in K_j^L,\ w_2 \in K_j^R\}$$

generates the same language as P does. Moreover if all of the rules of P are of type τ, then the same is true for Q.

Now we are able to give a finite grammar which generates the same language as $\mathscr{G}$ does. It is enough to start from Q and we may assume that $T \subseteq I$ and $I - T$ is the set of nonterminal symbols of Q.

Case $\tau = 3$. The same argument as above shows that the languages K_j^R must be finite. Therefore Q is finite and obviously of type 3.

Case $\tau = 2$. Because K_j^R is a regular language it is generated by some type 3 grammar $\mathscr{G}_j = \langle N_j, I, P_j, S_j \rangle$ where N_j and I are disjoint sets, $S_j \in N_j$ and the N_j's are disjoint for different j's. The grammar Q is of type 2 so $w \in K_j^L$ implies $w \in I$. Now take the following set of rules:

$$Q_j = \{w \to S_j\colon\ w \in K_j^L\} \cup P_j.$$

Obviously, $\bigcup_j Q_j$ is finite and of type 2 and $\mathscr{L}(Q) = \mathscr{L}(\bigcup_j Q_j)$.

Case $\tau = 1$. K_j^L is finite, so we may assume that it contains only one word, w_1^j, and let $|w_1^j| = n_j$. The lengths of the words of K_j^R are at least n_j, except if w_1^j

is the starting symbol, then K_j^R may contain the empty word, too. If we fix the first n_j symbols of the right hand side of a rule then the remaining part forms a regular language, which may be empty. There are only finitely many words of length n_j, therefore we may drop them into different sets, i.e. we arrive at

$$Q = \bigcup_j \{w_1^j \to w_2^j w\colon\ w \in K_j^R\} \cup Q^*$$

where $|w_1^j| = |w_2^j|$, K_j^R is regular, and Q^* is either empty or contains the rule $S \to \varepsilon$ only. The method of Case 2 now gives immediately a finite language of type 3 generating $\mathscr{L}(Q)$, only a little care should be taken of the empty word in K_j^R. □

REMARK. A close examination of the proof shows that given some regular metagrammar $\mathscr{G}$ and an iterated grammar $\mathscr{I}$ generated by $\mathscr{G}$, there is an effective procedure which gives from $\mathscr{G}$ and $\mathscr{I}$ a grammar $\mathscr{H}$ for which $\mathscr{L}(\mathscr{I}) = \mathscr{L}(\mathscr{H})$.

Theorem 2. $\chi(2, 3) = \chi(0)$.

Proof. By the Proposition, it is enough to prove that $\chi(2, 3) \supseteq \chi(0)$. Let $\mathscr{G} = \langle N, T, P, S\rangle$ be a type 0 grammar and assume that the comma and the double arrow are not in $N \cup T$. We give the iterated grammar of type (2, 3) simply by listing its production rules, which form evidently a context free language, or, what is more, a deterministic one.

Choose a new symbol $\tilde{t}$ for each $t \in T$ and let $\tilde{T} = \{\tilde{t}\colon t \in T\}$. Change all terminals in the production rules to their counterpart, let $\tilde{P}$ be the resulting set. Let $\Sigma = N \cup \tilde{T}$ and R a new symbol not in Σ or T. The desired iterated grammar is

$$\mathscr{I} = \langle \Sigma \cup \{R\}, (\Sigma \cup \{R\})^+, T, Q, S\rangle$$

where Q consists of

$$\begin{aligned}
&[R\alpha^{-1} \Rightarrow \alpha] && \text{for each } \alpha \in \Sigma^* \\
&\gamma\alpha\delta \Rightarrow R\delta^{-1}\beta^{-1}\gamma^{-1} && \text{for each } \gamma, \delta \in \Sigma^* \quad \text{and} \quad \alpha \to \beta \in \tilde{P} \\
&[\tilde{t}\alpha \Rightarrow t, R\alpha^{-1} && \text{for each } t \in T \quad \text{and} \quad \alpha \in \Sigma^*.
\end{aligned}$$

The production rules of $\mathscr{I}$ are of type 3, the derivations of the grammar $\mathscr{G}$ are encoded in the nonterminals of $\mathscr{I}$ in a straighforward way. □

Abstract

The definition of the programming language Algol 68 [2] raised the following problem: If a grammar is not given by some finite description but itself is a language generated by some metagrammar, what strength may the iterated grammar have? We show that a regular metagrammar does not increase the strength of the iterated grammar, but a context free metagrammar (even a deterministic one) with a regular iterated grammar has the strength of the phrase structure grammars.

MATHEMATICAL INSTITUTE OF THE
HUNGARIAN ACADEMY OF SCIENCES
REALTANODA U. 13—15.
BUDAPEST, HUNGARY
H—1053

References

[1] HOPCROFT and ULLMAN, *Formal languages and their relation to automata*, Addison-Wesley, 1967.
[2] VAN WIJNGAARDEN A, et al., Revised report on the algorithmic language Algol 68, Springer, 1976.

(Received May 31, 1979)

d-dependency structures in the relational model of data

By G. CZÉDLI

I. Introduction

The use of the relational model of data structures proposed by E. F. CODD [2, 3] is a promising mathematical tool for handling data. In this model the user's data are represented by relationships. For definition, let Ω be a finite non-empty set, and for each $b \in \Omega$ let T_b be a nonempty set associated with b. The elements of Ω are called attribute names and T_b is said to be the domain of b. Now a relationship over Ω is defined to be any finite subset of $\prod_{b \in \Omega} T_b$. A relationship R over $\Omega = \{a_1, \ldots, a_n\}$ can be represented by a two-dimensional table in which the columns correspond to attribute names and rows correspond to the elements of R:

	a_1	a_2	...	a_n
g	$g(a_1)$	$g(a_2)$	...	$g(a_n)$
$\vdots$				

($g \in R$ and $g(a_i) \in T_{a_i}$).

This table is not unique, the order of columns and that of rows are arbitrary.

The concept of functional dependency is due to E. F. CODD [2, 3]. For the definition, let A and B be subsets of Ω and let R be a relationship over Ω. We say that B functionally depends on A in R (in notation $A \xrightarrow[R]{f} B$ or simply $A \xrightarrow{f} B$) if for all $g, h \in R$

$$(\forall a \in A)(g(a) = h(a)) \Rightarrow (\forall b \in B)(g(b) = h(b))$$

is satisfied. The link $A \xrightarrow[R]{f} B$ is said to be a functional dependency.

From the above definition we can obtain three other concepts of dependency by changing the quantifiers. Particularly, the concept of d-dependency is introduced as follows:

DEFINITION. Let A and B be subsets of Ω and let R be a relationship over Ω. B is said to be *d-dependent* on A in the relationship R (in notation $A \xrightarrow[R]{d} B$ or simply $A \xrightarrow{d} B$) if for any $g, h \in R$

$$(\exists a \in A)(g(a) = h(a)) \Rightarrow (\exists b \in B)(g(b) = h(b))$$

holds.

In any relationship of a time-varying data structure at a particular moment of time there are dependencies. Some of them may be fortuitous or unimportant, but it is reasonable to require that at least certain dependencies be present at any time. Organizing the data structure and some of the user's activities can be based on these constant dependencies. In case of *functional* dependencies this has been shown in Codd's papers [2, 3]. Now we want to show the applicability of *d*-dependencies in this aspect. For this reason we give an example. Let

$$\Omega = \{\text{author, title, room, bookcase}\}$$

and let a relationship R be given in the following table:

author	title	room	bookcase	author	title	room	bookcase
1	1	1	2	10	10	3	2
2	2	1	3	11	11	3	3
3	3	1	1	12	12	3	1
4	4	1	2	1	4	1	1
5	5	2	3	5	8	3	3
6	6	2	1	4	1	1	3
7	7	2	2	7	10	3	2
8	8	2	3	6	10	2	2
9	9	3	1	6	9	2	1

For the sake of visibility we can think R is a library in which eighteen books are stocked. The library consists of three rooms, each room has three bookcases, and only two books can go in each bookcase. The library is organized so that $\{\text{author, title}\} \xrightarrow[R]{d} \{\text{room, bookcase}\}$. Furthermore, the book with author=title=i ($i=1, 2, \ldots, 12$) is in the $\left[\frac{i+3}{4}\right]$-th room in the $\left(1+3\left\{\frac{i}{3}\right\}\right)$-th bookcase. (Here $[x]$ denotes the largest integer not greater than x and $\{x\}=x-[x]$.) A reader who knows that either the title or the author of a particular book is, say, i can find the book by scanning the $\left[\frac{i+3}{4}\right]$-th room and the $\left(1+3\left\{\frac{i}{3}\right\}\right)$-th bookcases only.

Now in connection with this example we try to express why the concept of *d*-dependency can have some practical importance. The task of obtaining information from a given data structure is closely connected with the dependencies that are present in the data stucture. So, when we list some possibly advantageous properties of using *d*-dependencies below, we restrict our interest to the case of obtaining information only. Suppose the user "knows" the values of attributes of a given set A of attribute names and wants to learn the values of attributes of another set B.

(1) If $A \xrightarrow{d} B$ (in a given relationship R) then the user is not assumed to know all the attribute values from A. If he knows the value of at least one attribute in A and the *d*-dependency $A \xrightarrow{d} B$ is also given (by a suitable family of functions δ_a ($a \in A$), δ_a: $T_a \to \coprod_{b \in B} T_b$, compare with the functions $\left[\frac{i+3}{4}\right]$ and $1+3\left\{\frac{i}{3}\right\}$ in

our example), then he can find the values of attributes in B by scanning a part of R only. (The values of attributes in B can be not unique if $A \xrightarrow{f} B$ does not hold.)

(2) Suppose both $A \xrightarrow{f} B$ and $A \xrightarrow{d} B$ hold. (This was the case in our example with $A = \{\text{author, title}\}$ and $B = \{\text{room, bookcase}\}$.) Sometimes, in spite of scanning a part of R, the method of (1) can be more immediate than making use of the explicit function φ: $\prod_{a \in A} T_a \to \prod_{b \in B} T_b$ which describes the functional dependency $A \xrightarrow{f} B$, since such a function φ can be given by another table in general.

(3) One can have $A \xrightarrow{d} B$ without having $A \xrightarrow{f} B$.

(4) The user can need only at least one value of attributes in B (without knowing which one is correct). E.g., this can occur when he is interested in C, B is an intermediate step, and $B \xrightarrow[Q]{d} C$ holds in an other relationship Q.

For a given relationship R over Ω let

$$\mathscr{F}_R = \left\{(A, B):\ A \subseteq \Omega,\ B \subseteq \Omega,\ A \xrightarrow[R]{f} B\right\}$$

and

$$\mathscr{D}_R = \left\{(A, B):\ A \subseteq \Omega,\ B \subseteq \Omega,\ A \xrightarrow[R]{d} B\right\}.$$

$\mathscr{F}_R$ and $\mathscr{D}_R$ are called the full family of functional dependencies of R and the full family of d-dependencies of R, respectively. In [1] W. W. ARMSTRONG has given an abstract characterization of full families of functional dependencies. Our main goal here is to give an abstract characterization for full families of d-dependencies. Due to duality between the concept of functional dependency and that of d-dependency, a considerable part of Armstrong's paper [1] is dualized and used in the present paper.

II. Abstract characterization of d-dependencies

Let Ω be a finite non-empty set and let $P(\Omega)$ denote the set of all subsets of Ω. We define a partial order $\leqq$ over $P(\Omega) \times P(\Omega)$ by $(A, B) \leqq (C, D)$ iff $A \subseteq C$ and $B \supseteq D$. We recall a definition from Armstrong's paper [1]:

A subset $\mathscr{F}$ of $P(\Omega) \times P(\Omega)$ is called an *abstract full family of functional dependencies* over Ω if the following four axioms hold for any elements A, B, C and D in $P(\Omega)$:

(F1) $(A, A) \in \mathscr{F}$.
(F2) $(A, B) \in \mathscr{F}$ and $(B, C) \in \mathscr{F}$ imply $(A, C) \in \mathscr{F}$.
(F3) If $(A, B) \in \mathscr{F}$ and $(A, B) \leqq (C, D)$ then $(C, D) \in \mathscr{F}$.
(F4) If $(A, B) \in \mathscr{F}$ and $(C, D) \in \mathscr{F}$ then $(A \cup C, B \cup D) \in \mathscr{F}$.

Now Armstrong's abstract characterization of functional dependencies is the following:

A subset $\mathscr{F}$ of $P(\Omega) \times P(\Omega)$ is of the form $\mathscr{F} = \mathscr{F}_R$ for some relationship R over Ω iff $\mathscr{F}$ is an abstract full family of functional dependencies.

To formulate our main result the following definition is needed.

DEFINITION. A subset $\mathscr{D}$ of $P(\Omega)\times P(\Omega)$ is called an *abstract full family of d-dependencies* if the following five axioms hold for any elements A, B, C and D in $P(\Omega)$. (The notation $X \xrightarrow{d} Y$ will be used instead of $(X, Y)\in\mathscr{D}$).

(D1) $A \xrightarrow{d} A$.

(D2) If $A \xrightarrow{d} B$ and $B \xrightarrow{d} C$ then $A \xrightarrow{d} C$.

(D3) If $A \xrightarrow{d} B$ and $(C, D)\leqq(A, B)$ then $C \xrightarrow{d} D$.

(D4) If $A \xrightarrow{d} B$ and $C \xrightarrow{d} D$ then $A\cup C \xrightarrow{d} B\cup D$.

(D5) If $A \xrightarrow{d} \emptyset$ then $A=\emptyset$.

In the main theorem below an abstract characterization of *d*-dependencies is given.

Theorem. *Let Ω be an arbitrary non-empty set of attribute names. Then, for any non-empty relationship R over Ω, $\mathscr{D}_R$ is an abstract full family of d-dependencies. Conversely, for any abstract full family $\mathscr{D}$ of d-dependencies over Ω there exists a nonempty relationship R over Ω such that $\mathscr{D}=\mathscr{D}_R$.*

REMARK. The case $R=\emptyset$ is excluded from the Theorem. However this fact does not mean the loss of generality, since $\mathscr{D}_\emptyset$ trivially can be characterized by $\mathscr{D}_\emptyset=P(\Omega)\times P(\Omega)$.

III. The proof of the Theorem

It is a straightforward consequence of definitions that $\mathscr{D}_R$ is an abstract full family of *d*-dependencies.

To prove the converse several lemmas will be needed. In what follows all concepts and statements concern a fixed set $\Omega=\{a_1, \ldots, a_n\}$ of attribute names. For an abstract full family $\mathscr{D}$ of *d*-dependencies let us denote by $\mathscr{M}_\mathscr{D}$ the set of maximal elements of $\mathscr{D}$.

Claim 1. Let us denote $(A, B)\in\mathscr{M}_\mathscr{D}$ by $A \nearrow B$. Then $\mathscr{M}_\mathscr{D}$ has the following four properties:

(M1) For any $A\in P(\Omega)$ there exist X and Y in $P(\Omega)$ such that $(A, A)\leqq(X, Y)$ and $X \nearrow Y$;

(M2) If $A \nearrow B$, $C \nearrow D$ and $(A, B)\leqq(C, D)$, then $(A, B)=(C, D)$;

(M3) If $A \nearrow B$, $B\subseteq C$ and $C \nearrow D$, then $A\subseteq C$;

(M4) If $A \nearrow \emptyset$ then $A=\emptyset$;

where A, B, C and D are universally quantified over $P(\Omega)$.

Proof. M1, M2 and M4 are trivially satisfied. Suppose we have $A \nearrow B$, $B\subseteq C$ and $C \nearrow D$. Then $B \xrightarrow{d} C$ (i.e., $(B, C)\in\mathscr{D}$) follows from D1 and D3, whence $A \xrightarrow{d} D$ follows by D2. Now D4 yields $A\cup C \xrightarrow{d} D$. The maximality of (C, D) in $\mathscr{D}$ implies $A\cup C\subseteq C$, whence we obtain the required inclusion $A\subseteq C$. □

Let a subset $\mathscr{M}$ of $P(\Omega)\times P(\Omega)$ be called an *m-family* if it satisfies the axioms M1, M2, M3 and M4.

Claim 2. For any m-family $\mathscr{M}$ the set

$$\mathscr{D}_{\mathscr{M}} = \{(A, B) \in P(\Omega) \times P(\Omega) \colon \text{there exists } (C, D) \in \mathscr{M} \text{ such that } (A, B) \leqq (C, D)\}$$

is an abstract full family of d-dependencies.

Proof. It is trivial that $\mathscr{D}_{\mathscr{M}}$ satisfies D1, D3 and D5. To check D2, let (A, B) and (B, C) belong to $\mathscr{D}_{\mathscr{M}}$. Then $(A, B) \leqq (A_1, B_1)$ and $(B, C) \leqq (B_2, C_2)$ hold for some (A_1, B_1) and $(B_2, C_2) \in \mathscr{M}$. From $B_1 \subseteq B \subseteq B_2$ and M3 we obtain $A_1 \subseteq B_2$. Now $(A, C) \in \mathscr{D}_{\mathscr{M}}$ follows from $(A, C) \leqq (B_2, C_2)$.

As for D4, suppose (A, B) and (C, D) are in $\mathscr{D}_{\mathscr{M}}$. Let (A_1, B_1) and (C_1, D_1) be taken from $\mathscr{M}$ such that $(A, B) \leqq (A_1, B_1)$ and $(C, D) \leqq (C_1, D_1)$. Now M1 yields the existence of an (U, V) in $\mathscr{M}$ with the property $(B_1 \cup D_1, B_1 \cup D_1) \leqq (U, V)$. Since $B_1 \subseteq U$ and $D_1 \subseteq U$, M3 applies. We obtain $A_1 \subseteq U$ and $C_1 \subseteq U$. Thus the required $(A \cup C, B \cup D) \in \mathscr{D}_{\mathscr{M}}$ follows from $(A \cup C, B \cup D) \leqq (A_1 \cup C_1, B_1 \cup D_1) \leqq (U, V)$. □

Lemma 1. For any abstract full family $\mathscr{D}$ of d-dependencies the family $\mathscr{M}_{\mathscr{D}}$ of maximal elements of $\mathscr{D}$ is an m-family. Conversely, any m-family $\mathscr{M}$ is the family of maximal elements of exactly one abstract full family

$$\mathscr{D}_{\mathscr{M}} = \{(A, B) \in P(\Omega) \times P(\Omega) \colon (A, B) \leqq (C, D) \text{ for some } (C, D) \in \mathscr{M}\}$$

of d-dependencies.

Proof. D3 yields that any abstract full family $\mathscr{D}$ of d-dependencies is uniquely determined by $\mathscr{M}_{\mathscr{D}}$. The rest has already been proved in Claims 1 and 2. □

Now we could deal with m-families instead of abstract full families by Lemma 1. However, the concept of m-families is still complicated to our purposes. Surprisingly, certain semilattices will be suitable to characterize both abstract full families and m-families. For the sake of brevity, 0—1 subsemilattices of $P(\Omega)$ will be called *d-semilattices*. I.e., $\mathscr{S}$ is a d-semilattice over Ω iff it is a subset of $P(\Omega)$ containing $\emptyset$, Ω and the intersection of any two of its elements. The following statements will show the significance of d-semilattices. First, for an m-family $\mathscr{M}$, $\mathscr{S}_{\mathscr{M}}$, the semilattice according to $\mathscr{M}$, is defined by

$$\mathscr{S}_{\mathscr{M}} = \{A \colon A \subseteq \Omega \quad \text{and} \quad (A, B) \in \mathscr{M} \text{ for some } B\}.$$

Similarly, for an abstract full family $\mathscr{D}$, $\mathscr{S}_{\mathscr{D}}$ is defined by $\mathscr{S}_{\mathscr{M}_{\mathscr{D}}}$.

Claim 3. $\mathscr{S}_{\mathscr{M}}$ and $\mathscr{S}_{\mathscr{D}}$ are d-semilattices for any abstract full family $\mathscr{D}$ of d-dependencies and m-family $\mathscr{M}$.

Proof. It is enough to check that $\mathscr{S}_{\mathscr{M}}$ is a d-semilattice. From M1 we conclude that $\Omega \in \mathscr{S}_{\mathscr{M}}$. M1 and M4 yields that $\emptyset \in \mathscr{S}_{\mathscr{M}}$. Suppose A and B are in $\mathscr{S}_{\mathscr{M}}$, and let C, D be chosen so that $(A, C) \in \mathscr{M}$ and $(B, D) \in \mathscr{M}$. By M1 a pair $(U, V) \in \mathscr{M}$ is obtained such that $(A \cap B, A \cap B) \leqq (U, V)$. Since $V \subseteq A$ and $V \subseteq B$, M3 applies and we obtain $U \subseteq A$ and $U \subseteq B$. Hence $A \cap B = U$ implies $A \cap B \in \mathscr{S}_{\mathscr{M}}$. □

Claim 4. For any d-semilattice $\mathscr{S}$ over Ω the family

$$\mathscr{D}_{\mathscr{S}} = \{(A, B) \in P(\Omega) \times P(\Omega) \colon \text{for any } X \in \mathscr{S} \; B \subseteq X \text{ implies } A \subseteq X\}$$

is an abstract full family of d-dependencies.

The *proof* is straightforward and so it will be omitted.

Lemma 2. For any abstract full family $\mathscr{D}$ of d-dependencies $\mathscr{S}_{\mathscr{D}}$ is a d-semilattice. Conversely, any d-semilattice $\mathscr{S}$ coincides with $\mathscr{S}_{\mathscr{D}}$ for exactly one abstract full family $\mathscr{D}$ of d-dependencies, namely for $\mathscr{D} = \mathscr{D}_{\mathscr{S}}$.

Proof. We have already proved that $\mathscr{D}_{\mathscr{S}}$ is an abstract full family of d-dependencies and $\mathscr{S}_{\mathscr{D}}$ is a d-semilattice. First we show that $\mathscr{S} = \mathscr{S}_{\mathscr{D}_{\mathscr{S}}}$. Suppose $A \in \mathscr{S}$ and choose $B \in P(\Omega)$, $B \subseteq A$, such that B is minimal with respect to the property $(A, B) \in \mathscr{D}_{\mathscr{S}}$. In order to show that (A, B) is a maximal element in $\mathscr{D}_{\mathscr{S}}$, we assume that $(A, B) \prec (C, D) \in \mathscr{D}_{\mathscr{S}}$. Then $A \subset C$ because of the coice of B, and we have $(A, B) \prec (C, B) \leqq (C, D)$. Hence we obtain $(C, B) \in \mathscr{D}_{\mathscr{S}}$. Now $B \subseteq A \in \mathscr{S}$ and the definition of $\mathscr{D}_{\mathscr{S}}$ yield $C \subseteq A$, which is a contradiction. Therefore (A, B) is maximal in $\mathscr{D}_{\mathscr{S}}$ and so $A \in \mathscr{S}_{\mathscr{D}_{\mathscr{S}}}$.

To show the converse inclusion, suppose $A \in \mathscr{S}_{\mathscr{D}_{\mathscr{S}}}$. Then (A, B) is maximal in $\mathscr{D}_{\mathscr{S}}$ for some B. Let $\mathscr{H}$ denote the set $\{X \colon X \in P(\Omega) \text{ and } A \subset X\}$. Since $(X, B) \notin \mathscr{D}_{\mathscr{S}}$ $(X \in \mathscr{H})$, we can assign an element $U_X \in \mathscr{S}$ such that $B \subseteq U_X$ and $X \not\subseteq U_X$. Since $\mathscr{S}$ is a finite semilattice, $H = \bigcap \{U_X \colon X \in \mathscr{H}\}$ belongs to $\mathscr{S}$. Now $B \subseteq H$ and $(A, B) \in \mathscr{S}$ implies $A \subseteq H$. If we had $H \in \mathscr{H}$ then $H \not\subseteq U_H$ would contradict $H = \bigcap \{U_X \colon X \in \mathscr{H}\} \subseteq U_H$. Consequently, $A \not\subset H$ and so $A = H \in \mathscr{S}$. The equality $\mathscr{S} = \mathscr{S}_{\mathscr{D}_{\mathscr{S}}}$ has been shown.

For the uniqueness of $\mathscr{D}$ we assume that $\mathscr{S} = \mathscr{S}_{\mathscr{D}_1} = \mathscr{S}_{\mathscr{D}_2}$. We denote $\mathscr{M}_{\mathscr{D}_i}$ by $\mathscr{M}_i$. Suppose (A, B) belongs to $\mathscr{D}_1$. We can choose elements (A_i, B_i) from $\mathscr{M}_i$ $(i = 1, 2)$ such that $(A, B) \leqq (A_1, B_1)$ and $(B, B) \leqq (A_2, B_2)$. Since $A_2 \in \mathscr{S}$, $(A_2, C) \in \mathscr{M}_1$ for a suitable C. Now M3 yields $A_1 \subseteq A_2$, which implies $(A, B) \leqq (A_2, B_2)$. By D3 we obtain $(A, B) \in \mathscr{D}_2$. We have shown the inclusion $\mathscr{D}_1 \subseteq \mathscr{D}_2$, while $\mathscr{D}_2 \subseteq \mathscr{D}_1$ follows similarly. □

A map $\varphi \colon P(\Omega) \to P(\Omega)$ is called a closure operator (on Ω) if for any $X, Y \in P(\Omega)$, $X \subseteq Y$,

$$X \subseteq X\varphi = (X\varphi)\varphi$$

and

$$X\varphi \subseteq Y\varphi$$

hold. For any d-semilattice $\mathscr{S}$ we define a closure operator $\varphi_{\mathscr{S}}$ by the following way:

$$X\varphi_{\mathscr{S}} = \bigcap \{Y \colon Y \in \mathscr{S} \text{ and } X \subseteq Y\}.$$

It is easy to see that, for any $X \in P(\Omega)$, $X\varphi_{\mathscr{S}} \in \mathscr{S}$. Moreover $X \in \mathscr{S}$ iff $X = X\varphi_{\mathscr{S}}$

Lemma 3. Let $\mathscr{S}$ be a d-semilattice and $X \in P(\Omega)$. Then $X\varphi_{\mathscr{S}} = \{a \colon (\{a\}, X) \in \mathscr{D}_{\mathscr{S}}\}$.

Proof. Let U denote the right-hand side of the above equality, and let $\mathscr{D} = \mathscr{D}_{\mathscr{S}}$. D1 and D3 yield $X \subseteq U$ and $(X, U) \in \mathscr{D}$. We have $(U, X) \in \mathscr{D}$ by D4. Let A be a minimal (with respect to $\subseteq$) subset of X for which $(U, A) \in \mathscr{D}$. We claim that

$(U, A)\in\mathscr{M}_{\mathscr{D}}$. To show this let the opposite case, $(U, A)\prec(V, B)\in\mathscr{D}$, be assumed. By the choice of A we have $U\subset V$ and $(U, A)\prec(V, A)\leqq(V, B)$. We obtain

$(V, A)\in\mathscr{D}$ by D3,

$(A, X)\in\mathscr{D}$ by D1 and D3,

and

$(V, X)\in\mathscr{D}$ by D2.

Now $(\{v\}, X)\in\mathscr{D}$ for any $v\in V$ by D3. This means $V\subseteq U$, which contradicts $U\subset V$. Thus we have shown $(U, A)\in\mathscr{M}_{\mathscr{D}}$. Therefore $U\in\mathscr{S}_{\mathscr{D}}$, whence $U\in\mathscr{S}$ by Lemma 2.

Now $X_{\varphi_{\mathscr{S}}}\subseteq U$ follows from $X\subseteq U$ and $U\in\mathscr{S}$. To make the proof complete we have to show that if $X\subseteq C\in\mathscr{S}$ then $U\subseteq C$. Suppose $X\subseteq C\in\mathscr{S}$ and choose an element $D\in P(\Omega)$ such that (C, D) is a maximal element in $\mathscr{D}$. Since (U, A) is also a maximal element in $\mathscr{D}$ and $A\subseteq X\subseteq C$, we obtain $U\subseteq C$ from M3. □

COROLLARY. Let $\mathscr{S}$ be a *d*-semilattice and let $X\in P(\Omega)$. Then $X\in\mathscr{S}$ iff $(\{a\}, X)\in\mathscr{D}_{\mathscr{S}}$ for $a\in X$ only.

The concept of *d*-semilattices is already simple and worth being connected with relationships. A natural connection is given in the following definition.

DEFINITION. Let R be a relationship. We define $\mathscr{S}_R$, the *d*-semilattice associated with R, to be $\mathscr{S}_{\mathscr{D}_R}$.

Now, by Lemma 2, we have only to prove that for any *d*-semilattice $\mathscr{S}$ there exists a relationship R such that $\mathscr{S}=\mathscr{S}_R$. The simplest case is settled in the following

Claim 5. Let $\mathscr{S}=\{\emptyset, \Omega\}$. Then $\mathscr{S}=\mathscr{S}_R$ for any one-element relationship R.

The *proof*, which is trivial by definitions, will be omitted.

For $A\in P(\Omega)$ we define an at most three-element *d*-semilattice $\mathscr{T}_A$ to be $\{\emptyset, A, \Omega\}$. A relatively simple case is handled in the following

Lemma 4. Let $A\in P(\Omega)$, $A\neq\emptyset$ and $A\neq\Omega$. Then $\mathscr{T}_A=\mathscr{S}_R$ where $R=\{g, h\}$ is a two-element relationship defined by

$g(a)=1$ for all $a\in\Omega$,

$h(a)=2$ for $a\in A$,

$h(a)=1$ for $a\in\Omega\setminus A$.

Proof. The relationship R can be visualized by the following table:

	A			Ω/A		
	a	b	...	c	d	...
g	1	1		1	1	
h	2	2		1	1	

Since $(\{x\}, A)\notin\mathscr{D}_R$ for $x\notin A$, we have $A\in\mathscr{S}_R$ by the Corollary. Hence $\mathscr{T}_A\subseteq\mathscr{S}_R$. Now suppose $X\in P(\Omega)$, $X\neq A$, $X\neq\emptyset$, $X\neq\Omega$. If $A\setminus X$ is non-empty, say $u\in A\setminus X$, then $(\{u\}, X)\in\mathscr{D}_R$. Hence $u\in X\varphi_{\mathscr{S}_R}\setminus X$. If $A\setminus X$ is empty, i.e. $A\subset X\subset\Omega$, then $(\{v\}, X)\in\mathscr{D}_R$ and $v\in X\varphi_{\mathscr{S}_R}\setminus X$ for any $v\in\Omega\setminus X$. In both cases $X\neq X\varphi_{\mathscr{S}_R}$, whence $X\notin\mathscr{S}_R$. Therefore $\mathscr{T}_A=\mathscr{S}_R$. □

For d-semilattices $\mathscr{S}_i$ $(i\in I)$ we define $\sum_{i\in I}\mathscr{S}_i$, the sum of $\mathscr{S}_i$, to be the smallest d-semilattice containing $\mathscr{S}_i$ for all $i\in I$. It is easy to check that:

Claim 6. Let $\mathscr{S}_i$ $(i\in I,\ I$ finite) be d-semilattices over Ω. Then the following equality holds:

$$\sum_{i\in I}\mathscr{S}_i=\Big\{\bigcap_{i\in I}A_i\colon\ A_i\in\mathscr{S}_i\Big\}.$$

Now we introduce an addition concept for relationships, which will be in a close connection with the addition of d-semilattices.

DEFINITION. Let R_i $(i\in I,\ I$ finite) be non-empty relationships over Ω, where $R_i\subseteq\prod_{b\in\Omega}T_{b,i}$. For $i\in I$ and $f\in R_i$ we define $f^i\in\prod_{b\in\Omega}(\{i\}\times T_{b,i})$ by $f^i(b)=(i,f(b))$ $(b\in\Omega)$. Set $R_i'=\{f^i\colon f\in R_i\}$. Then $\sum_{i\in I}R_i$, the sum of R_i, is defined to be $\bigcup_{i\in I}R_i'$.

Roughly saying, we obtain $\sum_{i\in I}R_i$ if we make R_i $(i\in I)$ pairwise disjoint by indices and take their disjoint union.

A crucial step of our proof is

Lemma 5. Let R_i $(i\in I,\ I$ finite) be arbitrary relationships over Ω. Let $\sum_{i\in I}R_i$ be denoted by R. Then $\mathscr{S}_R=\sum_{i\in I}\mathscr{S}_{R_i}$.

Proof. Let $\mathscr{D}_R$, $\mathscr{D}_{R_i}$, $\mathscr{S}_R$ and $\mathscr{S}_{R_i}$ $(i\in I)$ be denoted by $\mathscr{D}$, $\mathscr{D}_i$, $\mathscr{S}$ and $\mathscr{S}_i$, respectively. First we show that for any $(A,B)\in P(\Omega)\times P(\Omega)$

$$(A,B)\in\mathscr{D}\quad\text{iff}\quad(A,B)\in\mathscr{D}_i\text{ for all }i\in I. \tag{1}$$

Suppose $(A,B)\in\mathscr{D}$. Let $g,h\in R_i$ such that $g(a)=h(a)$ for some $a\in A$. Then $g^i(a)=h^i(a)$ as well, whence $g^i(b)=h^i(b)$ and so $g(b)=h(b)$ for some $b\in B$. I.e., $(A,B)\in\mathscr{D}_i$. Conversely, let $(A,B)\in\mathscr{D}_i$ for all $i\in I$. Suppose $g^i,h^j\in R$ and $g^i(a)=h^j(a)$. Then $(i,g(s))=(j,h(a))$ implies $i=j$ and $g(a)=h(a)$ in R_i. Therefore there exists $b\in B$ such that $g(b)=h(b)$, from which we obtain $g^i(b)=(i,g(b))=(j,h(b))=h^j(b)$. Thus (1) has been shown.

Now let us assume that $A\in\mathscr{S}$. We compute by Lemma 3, Corollary 1 and (1) as follows:

$$A=A\varphi_{\mathscr{S}}=\{a\colon(\{a\},A)\in\mathscr{D}\}=\{a\colon(\{a\},A)\in\mathscr{D}_i\text{ for }i\in I\}=$$

$$=\bigcap_{i\in I}\{a\colon(\{a\},A)\in\mathscr{D}_i\}=\bigcap_{i\in I}A\varphi_{\mathscr{S}_i}.$$

Therefore $A\in\sum_{i\in I}\mathscr{S}_i$ by Claim 6. We have obtained that $\mathscr{S}\subseteq\sum_{i\in I}\mathscr{S}_i$. To prove the converse inclusion let $A\in\mathscr{S}_i$ and suppose $A\notin\mathscr{S}$. Then there exists an $a\in\Omega$ such that $a\in A\varphi_{\mathscr{S}}\setminus A$. We have $(\{a\},A)\in\mathscr{D}$ by Lemma 3 and $(\{a\},A)\in\mathscr{D}_i$ by (1). But $(\{a\},A)\in\mathscr{D}_i$ implies $a\in A\varphi_{\mathscr{S}_i}=A$, which is a contradiction. Hence $A\in\mathscr{S}$ and therefore $\mathscr{S}_i\subseteq\mathscr{S}$. Finally, $\mathscr{S}_i\subseteq\mathscr{S}$ $(i\in I)$ implies $\sum_{i\in I}\mathscr{S}_i\subseteq\mathscr{S}$, which completes the proof.

Now we can prove

Lemma 6. For any *d*-semilattice $\mathscr{S}$ there exists a relationship R such that $\mathscr{S}=\mathscr{S}_R$.

Proof. If $\mathscr{S}$ has at least three elements then Lemmas 4 and 5 together with the equality

$$\mathscr{S} = \sum_{\substack{A \in \mathscr{S} \\ A \neq \emptyset, \Omega}} \mathscr{T}_A$$

imply our statement. The rest is included in Claim 5. □

Finally, Lemmas 2 and 6 complete the proof of Theorem.

Abstract

The concept of *d*-dependencies in relationships is introduced. An axiomatic description of *d*-dependencies in an arbitrary relationship is presented.

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

References

[1] ARMSTRONG, W. W., Dependency structures of data base relationships, *Information Processing* 74, North Holland, 1974, pp. 580—583.
[2] CODD, E. F., A relational model of data for large shared data banks, *Comm. ACM*, v. 13, 1970, pp. 377—387.
[3] CODD, E. F., Further normalization of the data base relational model, Courant Inst. Comp. Sci. Symp. 6, *Data Base Systems*, Prentice-Hall, Englewood Cliffs, N. J., 1971, pp. 33—64.

(Received Sept. 17, 1979)

Synthesis of abstract algorithms

By L. VARGA

1. Introduction

In the second half of the sixties the programming methodology evolved on the basis of the recognation that the correctness of a program, corresponding to a given specification can be proven, and during the last ten years, the programming methodology has become one of the most important branches of computer science.

The programming methodology has been changed radically and it now includes broad research areas that deal with both practical and theoretical questions of program development and management. The current research directions in programming methodology is summarized in [14] and a more detailed description of its principal subareas can be found in the books [13], [15].

Within the scope of programming methodology important research has been concentrated on studying the correctness of programs. In recent years there has been increasing activity in this field. There are two different approaches to achieving program correctness.

1. The *engineering approach* has been aiming at turning the art of programming into an engineering science. Its aim is to develop more efficient software tools and specify standard that can be used in the process of development of programs as a means to improve the relaibility and reduce the software cost. Complete systems called automated software evaluation systems, for different phase of software life cycle have been developed such a system includes automatic tools for requirement and design analysis, testing, maintenance etc. A comprehensive survey of the software tools and automated software evaluation systems can be found in [12].

Altough the engineering approach generally is not capable of demonstrating the correctness of a program, this approach seems to be an effective approach to the validation of programs in practice.

2. The *analytic approach* uses program verification methods to ensure that the desired program conformes to its correctness specifications. The role of software verification, the proof techniques and various verification systems are discussed in the survey papers [5], [7].

As far as the analytic approach is concerned, in the beginning the attention was focused on a *posteriori verification* of programs. That is, the problem of the program correctness proof was approached in the following way. Given a program

and a specification, it is to be proven, that the program realizes exactly the mapping stated in the specification.

Later on, however it turned out, that different programs, which realize the same mapping may be essentially different from the point of view of the correctness proof. The difficulty of a proof depends on the complexity of the program.

This led to the conclusion, that the correctness of a program has to be established during its construction. Programs have to be designed such a way, that the proof of their correctness should be simple. In fact, first a correctness proof has to be constructed and than a corresponding program to this proof should be given. This is the *constructive approach* to achieving program correctness, which represents one of the most significant advance in programming methodology. This approach has produced various program design and construction methods ([1], [3], [10], [17]). This methods require as input a specification of what is to be achieved and produce as output a program text which is a specification of how is to be achieved.

The methods initiated by the constructive approach have made a fundamental contribution to the *synthesis of programs* in extracting principles for deriving programs systematically from their specifications. These principles are formulated precisely enough to be carried out by an automatic synthesis system in [8], [9].

The main steps of an automatic program synthesis system are.

1. The system accepts specifications, which describe some function to be realized by means of primitives of a well defined system. Generally these primitives are the statements of a programming language.

The basic approach is to transform the specifications step by step according to certain transformation rules, which are guided by two kinds of strategic controls:

2. Some transformations attempt to transform the specifications into equivalent specifications or replace them by stronger assertions about the states of the desired programs. The aim is to produce an appropriate form for applying programming strategies.

3. Other transformations attempt to transform the specification into the desired program text, decomposing a given program description into subprogram descriptions.

These new descriptions are transformed into newer ones repeatedly until a program text of a source programming language is obtained.

In this paper we concentrate on the third problem, and programming strategies are formulated for developing the desired abstract programs step by step using the Hoare's deductive system [3]. The levels of abstraction are used with the Vienna Definition Language [16]. This language permits concentration on logical solutions to problems, rather than the form and contraints within that the solution must be stated. The language helps the programmers to think in terms of hierarchy of macro statements and express structured programming logic in stepwise refinement of a program and its data structure.

We are influenced by the strategies formulated by N. Dershowitz and Z. Manna [2]. The essence of our paper is the presentation of similar strategies applied to VDL-programs.

The programming strategies are based on the Hoare's deductive system. Extending the Hoare's methods to VDL-statements of similar structure to the statements of usual programming languages is relatively simple. However the indeterminism — which means, that the VDL-language allows programs to be written

in that the execution order of the statements is not predefined — presents a special problem. The programming strategies of such structures can be formulated by using the results of correctness proof of parallel programs [4]. A detailed treatment of correctness proofs of parallel programs can be found in Gries and Owicky's papers [11].

The next section presents the basic strategies for developing VDL-programs from appropriate forms of specifications.

In Section 3 the VDL-graph is defined as an abstraction of a class of data structures. The VDL-graph specify a connected graph which has one entry node at least, but may have several terminal nodes and there must be a path from one entry node at least to a terminal node through every node in the graph.

In Section 4 the deductive technique is illustrated by the example of an abstract graph walk algorithm. Sections 5 and 6 demonstrate the application of VDL-graph to specifying a linkage editor and an inverse assembler model, respectively.

2. Strategies for stepwise refinement

In this section the main strategies are formulated for developing a VDL-program from its specification.

The following notation will be used

$$\{R\}\ P\ \{Q\}$$

where Q and R are logical statements about states of the abstract machine (VDL-machine) and P is a VDL-program (program-tree). This may be interpreted as follows: If Q is true before execution of a VDL-program P, and if the execution terminates, then R will hold after executing P. This notation expresses the partial correctness of a VDL-program P with respect to its input specification Q and output specification R.

Our initial goal is to synthesize a program of the general form

$$\{R(\xi)\}\ stmt(x_1, x_2, \ldots, x_n)\ \{Q(\xi_0) \wedge x_1 = f_1(\xi_0) \wedge x_2 = f_2(\xi_0) \wedge \ldots \wedge x_n = f_n(\xi_0)\}$$

where ξ_0 is the initial state of the abstract machine and $f_1, f_2, \ldots, f_n$ are given functions. We require that the output state ξ of the desired program *stmt* satisfy the given specification $R(\xi)$, provided the initial state ξ_0 satisfies the given input specification $Q(\xi_0)$.

In order to synthesize the program this top-level goal may be transformed into equivalent goal or it may be replaced by a stronger goal, which can be achieved by an assignement (value returning) instruction or reduced to subgoals by using the following strategies.

2.1. The strategy of assignment. Given the goal of the form

$$\{R(e_0';\ \mu(\xi;\ \langle s-c_1:\ e_1'\rangle, \ldots, \langle s-c_n:\ e_n'\rangle))\}\ e_0':\ stmt\ (x_1, \ldots, x_k)\ \{Q(x_1, \ldots, x_k;\ \xi)\}$$

where the selectors $s-c_1, s-c_2, \ldots, s-c_n$ are independent, and

$$e_i' = e_i(x_1, \ldots, x_k;\ \xi), \quad i = 0, 1, \ldots, n$$

then this goal can be achieved by the following value returning instruction

$$\begin{aligned} &stmt(x_1, x_2, \ldots, x_k) = \\ &\quad \text{PASS}: e_0(x_1, \ldots, x_k; \zeta) \\ &\quad s-c_1: e_1(x_1, \ldots, x_k; \zeta) \\ &\quad \vdots \\ &\quad s-c_n: e_n(x_1, \ldots, x_k; \zeta). \end{aligned}$$

2.2. The conditional strategy. A goal of the form

$$\{q_1 \vee q_2 \vee \ldots \vee q_n\}\ stmt\ \{p\}$$

can be reduced by the conditional instruction

$$\begin{aligned} &stmt = \\ &\quad p_1 \rightarrow stmt_1 \\ &\quad p_2 \rightarrow stmt_2 \\ &\quad \vdots \\ &\quad T \rightarrow stmt_n \end{aligned}$$

to the subgoals

$$\{q_1\}\ stmt_1\ \{p \wedge p_1\},\ \{q_2\}\ stmt_2\ \{p \wedge \neg p_1 \wedge p_2\},\ \ldots,\ \{q_n\}\ stmt_n\ \{p \wedge \neg p_1 \wedge \ldots \wedge \neg p_{n-1}\}.$$

Any control tree can be constructed by using only the following two macro definitions:

$$\begin{aligned} &stmt = \\ &\quad stmt_1; \\ &\quad\quad stmt_2 \end{aligned}$$

and

$$\begin{aligned} &stmt = \\ &\quad null; \\ &\quad\quad stmt_1, \\ &\quad\quad\quad \vdots \\ &\quad\quad stmt_n \end{aligned}$$

The strategies for these basic forms will now be given.

2.3. The strategy of composition. A goal

$$\{r\}\ stmt\ \{p\}$$

can be decomposed by the instruction

$$\begin{aligned} &stmt = \\ &\quad stmt_1; \\ &\quad\quad stmt_2 \end{aligned}$$

to the subgoals

$$\{q\}\ stmt_2\ \{p\} \text{ and } \{r\}\ stmt_1\ \{q\}.$$

2.4. The strategy of indeterminism. Given a conjunctive goal of the form

$$\{q_1 \wedge q_2 \wedge \ldots \wedge q_n\}\ stmt\ \{p_1 \wedge p_2 \wedge \ldots \wedge p_n\}$$

then it can be reduced by the instruction

$$\begin{array}{l} stmt = \\ \quad null; \\ \qquad stmt_1, \\ \qquad stmt_2, \\ \qquad \vdots \\ \qquad stmt_n \end{array}$$

to the following subgoals

$$\{q_1\}\ stmt_1\ \{p_1\},\ \{q_2\}\ stmt_2\ \{p_2\}, \ldots, \{q_n\}\ stmt_n\ \{p_n\}$$

provided these theorems are *interference-free*. This property of the theorems is defined as follows:

Definition 2.1. Given a control tree t with the theorem

$$\{q\}\ t\ \{p\} \tag{i}$$

and the value returning instruction *stmt* with some precondition pre(*stmt*). If the execution of *stmt* after t does not alter the validity of q, that is

$$\{q\}\ stmt\ \{\text{pre}\,(stmt) \wedge q\}$$

and the execution of *stmt* before any *st* within t does not alter the validity of the precondition of *st*, that is

$$\{\text{pre}\,(st)\}\ stmt\ \{\text{pre}\,(stmt) \wedge \text{pre}\,(st)\}$$

then we say that *stmt* does not interfere with theorem (i).

Definition 2.2. Given the theorems

$$\{q_1\}\ stmt_1\ \{p_1\},\ \{q_2\}\ stmt_2\ \{p_2\}, \ldots, \{q_n\}\ stmt_n\ \{p_n\} \tag{ii}$$

and let st_i be a value returning instruction within $stmt_i$. If for all i, $i=1, 2, \ldots, n$ st_i does not interfere with

$$\{q_j\}\ stmt_j\ \{p_j\};\quad j = 1, 2, \ldots, n,\quad j \neq i$$

then the theorems (ii) are interference-free.

Accordingly, in applying the strategy of indeterminism, we must ensure the interference-free. If $q_1, q_2, \ldots, q_n$ are statements about different components of the state ξ and similarly $p_1, p_2, \ldots, p_n$ do not contain common variables and the macros $stmt_1, stmt_2, \ldots, stmt_n$ operate on independent components of ξ then the interference-freeness obviously satisfies.

The conditional strategy has an important special case:

2.5. The strategy of iteration. A goal of the form

$$\{q\}\ stmt\ \{p\}$$

can be decomposed by the iteration

$$\begin{array}{l} stmt = \\ \quad p_1 \rightarrow stmt_2 \\ \quad T \rightarrow stmt; \\ \qquad\qquad stmt_1 \end{array}$$

to the subgoals

$$\{p\}\ stmt_1\ \{p \wedge \neg p_1\} \quad \text{and} \quad \{q\}\ stmt_2\ \{p \wedge p_1\}$$

provided the iteration terminates.

Here the conjunctive goal $p \wedge p_1$ is achieved by forming an iteration so that the predicate p remains invariant during the iteration until the predicate p_1 is found false.

In the special case of $q = p \wedge p_1$, the instruction

$$\begin{array}{l} stmt = \\ \quad p_1 \rightarrow null \\ \quad T \rightarrow stmt; \\ \qquad\qquad stmt_1 \end{array}$$

can be used for reducing our goal to the subgoal

$$\{p\}\ stmt_1\ \{p \wedge \neg p_1\}.$$

At last a rule will be given here, which can be used for proving the termination of an iteration.

2.6. The rule of termination. Let the iteration

$$\begin{array}{l} stmt = \\ \quad p_1 \rightarrow null \\ \quad T \rightarrow stmt; \\ \qquad\qquad stmt_1 \end{array}$$

with the precondition

$$\text{pre}\,(stmt) \equiv p$$

be given.

Let u be an integer function of the appropriate variables. If

a) $p \supset u \geqq 0$

b) $p \wedge \neg p_1 \supset u > 0$

c) $\{u' < u\}\ stmt_1\ \{p \wedge \neg p_1\}$ (u' is the value of u after $stmt_1$)

d) and any assignment statement, that can be executed parallel with the statement *stmt* does not interfere with the theorem c;

then the iteration terminates.

For example, the termination of the iteration

$$\begin{array}{l} process\,(t) = \\ \quad \text{length (list)} = 0 \rightarrow null \\ \quad T \rightarrow process\,(\text{tail}\,(t)); \\ \qquad proc\,(\text{head}\,(t)) \end{array}$$

is guaranted, because for the function

$$u(t) = \text{length}\,(t)$$

with the precondition

$$\text{is-pred-list}\,(t)$$

all the criterions a)—d) hold.

3. The VDL-graph

The graph, that can be walked from its entry nodes, plays an important role in programming. In this section the definition of the VDL-graph is given, which can be viewed as an abstraction of graph data structures.

Let

$$\text{is-node-set} = (\{\langle s\text{: is-node}\rangle | \text{is-select}\,(s)\})$$
$$\text{is-node} = (\langle s\text{-value: is-pred}\rangle, \langle s\text{-desc: is-select-list}\rangle)$$

where "is-select" and "is-pred" represent arbitrary predicates. Such an object is shown in Figure 3.1, where

$$a \in \{x | \text{is-pred}\,(x)\}$$

and

$$s, s_i \in \{s' | \text{is-select}\,(s')\}$$

Let

$$\text{is-node-set}\,(f) = T$$

The notation $t \in f$ is used if

$$(\exists s, \text{is-select}\,(s) = T)(s(f) = t).$$

Definition 3.1. Let $t \in f$ and $n \in f$. The node n *refers* to t if and only if

$$(\exists i,\ 1 \leqq i \leqq \text{length}\,(s\text{-desc}\,(n)))(\text{elem}\,(i)(s\text{-desc}\,(n))(f) = t).$$

Notationally, we shall use the form

$$n \Rightarrow t.$$

Definition 3.2. The node t_k is *reachable* from node t_1, or there exists a *reference path* from t_1 to t_k if and only if

$$t_1 \Rightarrow t_2 \Rightarrow \ldots \Rightarrow t_k, \quad (t_i \in f, \quad i = 1, 2, \ldots, k).$$

We shall use the following notation for the reference path

$$t_1 \Rightarrow * t_k.$$

Definition 3.3. The set of VDL-*graph* is

$$\{g | \text{is-pred-graph}\,(g)\}$$

where

$$\text{is-pred-graph} = \text{is-node-set}$$

and there exists a non-empty subset $M(g)$ of the nodes of g distinquished with the property that any node $n \in g$ and $n \notin M(g)$ can be reached from at least one element of $M(g)$.

The elements of $M(g)$ are called *directly reachable nodes.*

Consequently each node of a VDL-graph can be reached from at least one directly reachable node.

Definition 3.4. Let root (i) be the function, for which

$$\text{root}\,(i) = s_i, \quad i = 1, 2, \ldots, n$$

if

$$M(g) = \{s_1(g), s_2(g), \ldots, s_n(g)\}$$

and let

$$\text{value}\,(n) = s\text{-value}\,(n),$$

$$\text{next}\,(i)(n) = (\text{elem}\,(i)(s\text{-desc}\,(n))(g), \quad 1 \leqq i \leqq \text{length}\,(s\text{-desc}\,(n)).$$

These functions can be used as selectors, for example

$$\text{value.next}\,(2).\text{next}\,(1).\text{root}\,(3)(g) = \text{value}\,(\text{next}\,(2)(\text{next}\,(1)(\text{root}\,(3)(g)))).$$

Using the functions defined above, the structure of a VDL-graph can be visualized by a graph. For example, the VDL-graph, denoted by the following relations

$$\begin{aligned}
g &= \mu_0(\langle \text{root}\,(1)\colon\ n_1\rangle, \langle \text{root}\,(2)\colon\ n_2\rangle, \langle s_3\colon\ n_3\rangle, \langle s_4\colon\ n_4\rangle, \langle s_5\colon\ n_5\rangle),\\
n_1 &= \mu_0(\langle s\text{-value}\colon\ a\rangle, \langle s\text{-desc}\colon\ \langle s_3, s_4\rangle\rangle),\\
n_2 &= \mu_0(\langle s\text{-value}\colon\ b\rangle, \langle s\text{-desc}\colon\ \langle s_4\rangle\rangle),\\
n_3 &= \mu_0(\langle s\text{-value}\colon\ c\rangle, \langle s\text{-desc}\colon\ \langle\,\rangle\rangle),\\
n_4 &= \mu_0(\langle s\text{-value}\colon\ d\rangle, \langle s\text{-desc}\colon\ \langle s_5, s_2\rangle\rangle),\\
n_5 &= \mu_0(\langle s\text{-value}\colon\ e\rangle, \langle s\text{-desc}\colon\ \langle\,\rangle\rangle),
\end{aligned}$$

can be represented by the graph shown in Figure 3.2.

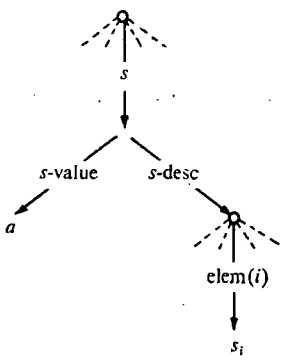


Fig. 3.1
A node set

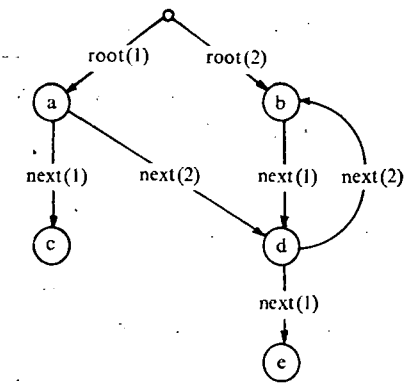


Fig. 3.2
A VDL-graph

The nodes of the graph in the Fig. 3.2 are circles and the values yielded by the nodes are put in the circles. The relationships between the nodes are represented by arrows and the arrows are named by the function next (i).

The figure of a VDL-graph reflects its structure in this way, but the formula of a VDL-graph does not do it directly. However it is not difficult to construct a formula that also satisfies this requirement. This problem is not dealt here.

In Definition 3.4 selection operations are defined on the VDL-graph. Construction operations can also defined on it, but we intend to deal with statical VDL-graph only, hence construction operations are not defined.

4. The synthesis of the graph walk algorithm

The graph walk is a fundamental operation. Most of the selection and construction operations of a graph can be established on it.

A graph walk can be carried out according to different strategies. In a graph walk algorithm each node of the graph is processed one after the other. The walk strategy determines the order of the nodes to be processed. In the following, we present a systematic development of a general graph walk algorithm, where the walk strategy and the operations over the nodes are not specified. In this way an abstraction of the graph walk algorithms is given from which concrete graph walks can be deduce by the specification of the walk strategy and the operation over the nodes.

Our top level goal is

Goal 1.

$$\{\text{is-target-graph}\ (g')\}\ g' \colon \mathit{walk}\ (g)\ \{\text{is-source-graph}\ (g)\}$$

where the map

$$\text{trans} \colon \{x | \text{is-source}\ (x)\} \to \{y | \text{is-target}\ (y)\}$$

is not specified. Therefore the function trans will be used as a parameter of the desired algorithm:

$$\mathit{walk}\ (g;\ \text{trans}).$$

Let the set of states of the abstract machine be

$$\{\xi | \text{is-state}\ (\xi)\}$$

where

$$\text{is-state} = (\langle s\text{-graph} \colon \text{is-pred-graph}\rangle, \langle s\text{-table} \colon \text{is-table}\rangle, \langle s\text{-control} : \text{is-control}\rangle).$$

The component s-graph (ξ) is the graph g to be walked. The component s-table (ξ) is used to mark which nodes of the graph g have been processed. Therefore

$$\text{is-table} = (\{\langle s \colon \text{is-value}\rangle | \text{is-select}\ (s)\})$$

where

$$\widehat{\text{is-value}} = \{Y, N\}$$

so that

$$s(s\text{-table}\ (\xi)) = Y$$

if and only if the s-value $(s(g))$ has been maped to a target value.

We do not intend to specify the walk strategy. Therefore we introduce the function

$$\text{next-selector}$$

as follows:

Definition 4.1. The function next-selector is a function over the set

$$\{t|\text{is-table}\ (t)\}$$

and the range of the function is

$$\widehat{\text{is-select}} \cup \{\Omega\}$$

so that if

$$(\exists s,\ \text{is-select}\ (s) = T)(s(t) = F)$$

then next-selector provides a selector s with the property

$$s(t) = N$$

else

$$\text{next-selector}\ (t) = \Omega.$$

Informally, the function next-selector (t) provides one of the selectors of the table t as $s(t)=N$, if such an s exists and the object Ω otherwise.

It is supposed that if the function next-selector is applied to the same table t several times the result is the same.

The function next-selector will also be used as a formal parameter of the desired algorithm and the formal parameter g will be omitted, because it is a component of the state ξ:

$$\textit{walk}\ (;\ \text{next-selector, trans})$$

and it is not a value returning macro.

We can now define the initial state ξ_0 of the abstract machine as follows

$$\xi_0 = \mu_0(\langle s\text{-graph:}\ g\rangle, \langle s\text{-table:}\ t_0\rangle, \langle s\text{-control:}\textit{walk}\ (;\text{next-selector, trans})\rangle)$$

where

$$\text{is-source-graph}\ (g) = T$$

and

$$t_0 = \mu_0(\{\langle s:\ N\rangle | s(g) \in M(g)\}).$$

Hence the input specification is

$$\varphi(\xi_0):\ s\text{-table}\ (\xi_0) = \mu_0(\{\langle s:\ N\rangle | s(g) \in M(g)\}) \wedge\ \text{is-source-graph}\ (g) \wedge g = s\text{-graph}\ (g),$$

and our goal is

Goal 2.

$$\{\text{is-target-graph}\ (s\text{-graph}\ (\xi))\}\ \textit{walk}\ (;\ \text{next-selector, trans})\ \{\varphi(\xi_0)\}$$

where the formal specifications of the formal parameters next-selector and trans are disregarded.

In order to synthesize the program, we must find a sequence of transformations to yield an equivalent description of the specification, that can be reduced by apply-

ing one of the strategies given in Section 2. First let us intend to prepare the application of the strategy of iteration.

To produce an appropriate form of the output specification, let us specify the invariable properties of the data structures.

Let

$$\alpha(s, \xi) \equiv s(s\text{-table}\ (\xi)).$$

Our graph walk strategy could be the following: Each node $s'(g)$ with the property

$$\alpha(s', \xi) = \Omega$$

must be reachable from at least one node $s(g)$ that waits for being processed with the property

$$\alpha(s, \xi) = N.$$

Hence, the formal specification of the data components of ξ is

$$Q_1(\xi)\colon\ R_1(\xi, g) \wedge R_2(\xi, g) \wedge R_3(\xi, g) \wedge R_4(\xi, g) \wedge R_5(g) \wedge g = s\text{-graph}\ (\xi),$$

where

$$R_1(\xi, g)\colon\ (\forall s, \alpha(s, \xi) \neq \Omega)(s(g) \neq \Omega) \wedge \text{is-table}\ (s\text{-table}\ (\xi)),$$
$$R_2(\xi, g)\colon\ (\forall s, \alpha(s, \xi) = Y)(\text{is-target}\ (s\text{-value}\ (s(g)))),$$
$$R_3(\xi, g)\colon\ (\forall s, \text{is-target}\ (s\text{-value}\ (s(g))))(\alpha(s, \xi) = Y),$$
$$R_4(\xi, g)\colon\ (\forall s', s'(g) \neq \Omega \wedge \alpha(s', \xi) = \Omega)((\exists s, \alpha(s, \xi) = N)(s(g) \Rightarrow * s'(g))),$$
$$R_5(g)\colon\ \text{is-mixed-graph}\ (g) \wedge \text{is-mixed} = \text{is-source} \vee \text{is-target}.$$

Theorem 4.1.

$$Q_1(\xi_0) \supset \varphi(\xi_0).$$

Theorem 4.2.

$$Q_1(\xi) \wedge \text{next-selector}\ (s\text{-table}\ (\xi)) = \Omega \supset \text{is-target-graph}\ (s\text{-graph}(\xi)).$$

Hence our goal may be

Goal 3.

$$\{Q_1(\xi) \wedge \text{next-selector}\ (s\text{-table}(\xi)) = \Omega\}\ \textit{walk}\ (;\ \text{next-selector, trans})\ \{Q_1(\xi)\}.$$

This suggests achieving Goal 3 with a recursive call applied to the macro *walk* as follows:

walk (; next-selector, trans)=
  next-selector $(s$-table $(\xi))=\Omega \rightarrow$ *null*
  $T \rightarrow$ *walk* (; next-selector, trans);
    process (; next-selector, trans)

which reduces Goal 3 to the subgoal

Goal 4.

$$\{Q_1(\xi)\}\ \textit{process}\ (;\ \text{next-selector, trans})\ \{Q_1(\xi) \wedge \text{next-selector}\ (s\text{-table}\ (\xi)) \neq \Omega\}.$$

Furthermore we must ensure the termination of the iteration. To achieve the termination, we could require that the number of the nodes $s(g)$ with the property

$$\alpha(s, \xi) = Y$$

be strictly increased with each iteration. Let $U(\xi)$ be the number of nodes with the above property, then our goal is

Goal 5.

$$\{Q_1(\xi) \wedge U(\xi) \succ a\}\ \textit{process}\ (;\ \text{next-selector, trans})$$
$$\{Q_1(\xi) \wedge \text{next-selector}\,(s\text{-table}\,(\xi)) \neq \Omega \wedge U(\xi) = a\}.$$

Using the strategy of composition this can be achieved by the macro

process (; next-selector, trans)=
  process-node (s; trans);
  s: *produce-selector*(; next-selector)

reducing Goal 5 to two subgoals

Goal 6.

$$\{Q_1(\xi) \wedge \alpha(s, \xi) = Y \wedge U(\xi) = a+1\}\ \textit{process-node}\ (s;\ \text{trans})$$
$$\{Q_1(\xi) \wedge \alpha(s, \xi) = N \wedge U(\xi) = a\}$$

and

Goal 7.

$$\{Q_1(\xi) \wedge \alpha(s, \xi) = N \wedge U(\xi) = a\}\ s\text{: }\textit{produce-selector}(;\ \text{next-selector})$$
$$\{Q_1(\xi) \wedge \text{next-selector}\,(s\text{-table}\,(\xi)) \neq \Omega \wedge U(\xi) = a\}.$$

Goal 7 can be achieved by the strategy of assignment:

produce-selector (; next-selector)=
  PASS: next-selector (s-table (ξ)).

In order to find a strategy for reducing Goal 5, let us isolate the effect of selector s on predicate $Q_1(\xi)$. Predicate Q_1 has five components. The predicate

$$\alpha(s, \xi) = Y \vee \alpha(s, \xi) = N$$

is of no effect on the first component of Q_1.

Let us consider the components R_2 and R_3.

Theorem 4.3.

$$R_2(\xi, g) \wedge R_3(\xi, g) \equiv R_{21}(\xi, g, s) \wedge R_{31}(\xi, g, s) \wedge Q_{21}(\xi, g, s),$$

where

$R_{21}(\xi, g, s)$: $(\forall s', \alpha(s', \xi) = Y \wedge s' \neq s)$(is-target ($s$-value ($s'(g)$))),
$R_{31}(\xi, g, s)$: $(\forall s',$ is-target (s-value ($s'(g)$))$\wedge s' \neq s)(\alpha(s', \xi) = Y)$,
$Q_{21}(\xi, g, s)$: $\alpha(s, \xi) = Y \wedge$ is-target (s-value ($s(g)$)).

Let us see the component R_4. We have different cases:

1. $s(g) \not\Rightarrow *s'(g)$ ($s'(g)$ is not reachable from $s(g)$),
2. $s(g) \Rightarrow s^*(g) \Rightarrow *s'(g)$ and $\alpha(s^*, \xi) \neq \Omega$,
3. $s(g) \Rightarrow s^*(g) \Rightarrow *s'(g)$ and $\alpha(s^*, \xi) = \Omega$,
4. $s(g) \Rightarrow s'(g)$.

Obviously, we have not to bother with the first two cases. Hence

Theorem 4.4.

$$R_4(\xi', g) \equiv R_{41}(\xi, g, s) \wedge Q_{22}(\xi, \xi', g, s)$$

where

$$R_{41}(\xi, g, s): (\forall s', s'(g) \neq \Omega \wedge \alpha(s', \xi) = \Omega \wedge \neg((s(g) \Rightarrow s^*(g) \Rightarrow *s'(g) \wedge \alpha(s^*, \xi) = \Omega) \vee s(g) \Rightarrow s'(g)))(\exists \bar{s}, \alpha(\bar{s}, \xi) = N \wedge \bar{s} \neq s)(\bar{s}(g) \Rightarrow *s'(g)),$$

$$Q_{22}(\xi, \xi', g, s): (\forall s', s(g) \Rightarrow s^*(g) \Rightarrow s'(g) \wedge \alpha(s^*, \xi) = \alpha(s', \xi) = \Omega)(\alpha(s^*, \xi') = N) \wedge (\forall s', s(g) \Rightarrow s'(g) \wedge \alpha(s', \xi) = \Omega)(\alpha(s', \xi') = N).$$

Theorem 4.5.

$$Q_1(\xi') \wedge \alpha(s, \xi') = Y \wedge U(\xi') = a+1 \equiv Q_2(\xi', s) \wedge Q_{21}(\xi', g, s) \wedge Q_{22}(\xi, \xi', g, s)$$

where

$$Q_2(\xi', s) \equiv R_1(\xi', g) \wedge R_{21}(\xi', g, s) \wedge R_{31}(\xi', g, s) \wedge R_{41}(\xi', g, s) \wedge R_5(g) \wedge g = s\text{-graph}\,(\xi').$$

Theorem 4.6.

$$Q_2(\xi, s) \wedge \alpha(s, \xi) = N \supset Q_1(\xi) \wedge \alpha(s, \xi) = N.$$

Hence our goal is

Goal 8.

$$\{Q_2(\xi', s) \wedge Q_{21}(\xi', g, s) \wedge Q_2(\xi, \xi', g, s)\}\ \textit{process-node}\ (s(s\text{-graph}\,(\xi)); s;\ \text{trans})\ \{Q_2(\xi, s) \wedge \alpha(s, \xi) = N\}.$$

We try to achieve it with the strategy of indeterminism of the form

$$\textit{process-node}\ (n, s;\ \text{trans}) = \textit{process-value}\ (\text{trans}\ (s\text{-value}\ (n)), s), \textit{process-desc}\ (s\text{-desc}\ (n))$$

reducing Goal 8 to the subgoals

Goal 9.

$$\{Q_2(\xi, s) \wedge Q_{21}(\xi, g, s)\}\ \textit{process-value}\ (v, s)\ \{Q_2(\xi, s) \wedge \alpha(s, \xi) = N \wedge v = \text{trans}\ (s\text{-value}\ (s(g)))\}$$

and

Goal 10.

$$\{Q_2(\xi', s) \wedge Q_{22}(\xi', \xi, g, s)\}\ \textit{process-desc}\ (\text{list})\ \{Q_2(\xi, s) \wedge \text{list} = s\text{-desc}\ (s(g))\}$$

provided these are interference-free.

Goal 8 can be achieved by using the strategy of assignment:

process-value $(v, s) =$
s-graph: $\mu(s\text{-graph}(\xi); \langle s\text{-value}_0\, s: v\rangle)$
s-table: $\mu(s\text{-table}(\xi); \langle s: Y\rangle)$.

Let us consider Goal 10. The significant part of the specification is Q_{22}. In order to try to achieve it by iteration, we attempt to apply the following transformation:

Theorem 4.7.

$$Q_{22}(\xi', \xi, g, s) \equiv Q_{221}(w_1, w_2) \wedge \text{length}(w_2) = 0,$$

where

$$Q_{221}(w_1, w_2): (\forall s', s(g) \Rightarrow s^*(g) \Rightarrow s'(g) \wedge \alpha(s^*, \xi) = \alpha(s', \xi) = \Omega \wedge s^* \in w_1)$$
$$(\alpha(s^*, \xi') = N) \wedge (\forall s', s(g) \Rightarrow s'(g) \wedge \alpha(s', \xi) = \Omega \wedge s' \in w_1)$$
$$(\alpha(s', \xi') = N) \wedge \widehat{w_1 w_2} = s\text{-desc}(s(g)).$$

We can now achieve our goal by creating an iteration whose exit is length $(w_2)=0$ and whose invariant assertion is $Q_2 \wedge Q_{221}$. The desired program is

process-desc $(w) =$
length $(w) = 0 \rightarrow$ *null*
$T \rightarrow$ process-desc (tail (w));
set (head (w))

which reduces Goal 10 to the subgoal

Goal 11.

$$\{Q_2(\xi', s) \wedge Q_{221}(\widehat{w_1\langle s^*\rangle}, \text{tail}(w_2))\}\ \textit{set}\ (s^*)$$
$$\{Q_2(\xi, s) \wedge Q_{221}(w_1, w_2) \wedge s^* = \text{head}(w_2) \wedge \text{length}(w_2) \neq 0\}.$$

Obviously, the termination is now ensured.

Goal 11 can be achieved by the conditional strategy:

set $(s) =$
$s(s\text{-table}(\xi)) \neq \Omega \rightarrow$ *null*
$T \rightarrow$ link (s)

which reduces Goal 11 to the subgoal

Goal 12.

$$\{\alpha(s, \xi) = N\}\ \textit{link}\ (s)\ \{\alpha(s, \xi) = \Omega\}.$$

Goal 12 can be achieved by a simple assignment:

link $(s) =$
s-table: $\mu(s\text{-table}(\xi); \langle s: N\rangle)$.

Theorem 4.8. The theorems in Goal 9 and Goal 10 are interference-free. The complete program is

walk (; next-selector, trans)=
  next-selector (*s*-table (ξ))=Ω→*null*
  T→*walk* (; next-selector, trans);
    process (; next-selector, trans)

process (; next-selector, trans)=
  process-node (*s*(*s*-graph (ξ)), *s*; trans);
    s: *produce-selector* (; next-selector)

produce-selector (; next-selector)=
  PASS: next-selector (*s*-table (ξ))

process-node (*n*, *s*; trans)=
  process-value (trans (*s*-value(*n*)), *s*),
  process-desc (*s*-desc (*n*))

process-value (*v*, *s*)=
  s-graph: μ(*s*-graph (ξ̄); ⟨*s*-value.*s*: *v*⟩)
  s-table: μ(*s*-table (ξ); ⟨*s*: *Y*⟩)

process-desc (list)=
  length (list)=0→*null*
  T→*process-desc* (tail (list));
    set (head (list))

set (*s*)=
  s(*s*-table (ξ))≠Ω→*null*
  T→*link* (*s*)

link (*s*)=
  s-table: μ(*s*-table (ξ); ⟨*s*:*N*⟩).

5. An abstract linkage editor

Let us consider a programming system where the segments refer to each other only by the segments name. Then the graph walk algorithm can be applied for defining a linkage editor of this system as follows:

Let

is-*r*/*b*-program=is-segment-code-graph

and

is-select=is-segment-name.

In detail:

is-*r*/*b*-program=({⟨*s*: is-node⟩|is-segment-name (*s*)}),
is-node=(⟨*s*-value: is-segment-code⟩, ⟨*s*-desc: is-segment-name-list⟩).

Let

editor (*t*)

be a function that maps a segment-code to an appropriate form as needed for linking. The actual mapping is not relevant here.

Then an abstract linkage editor can be characterized by the VDL-machine with the initial state:

$$\xi_0=\mu_0(\langle s\text{-input}: p\rangle, \langle s\text{-table}: t_0\rangle, \langle s\text{-ccntrol}: \textit{walk}\ (;\ \text{next-selector, editor})\rangle)$$

where

$$\text{is-}r/b\text{-program}\ (p)=T.$$

A linkage editor model of the system in which the segments may refer to each other by entry names different from the segment name, can be defined by a generalization of the VDL-graph and the graph walk algorithm.

6. An inverse assembler model

Semantics of a class of inverse assemblers can also be defined by the graph walk algorithm.

First of all, let us define the machine code program. A machine code program is an ordered set of codes, where a code according to its function, may be an instruction or a data. That is, those programs are considered where the instructions and the data are not separated.

Definition 6.1. The set of machine code program is given by

$$\{p|\text{is-code-list}\ (p)\}$$

where

$$\text{is-code} = \text{is-data} \vee \text{is-}\textit{stmt}.$$

It is assumed that the program does not alter the instruction code at all and each instruction code contains the address of the next instruction explicitly that should be executed.

The instruction code part of a machine code program is called an actual program. It is assumed, that an actual program has a finite set of entries, and for any instruction at least one entry can be found where starting the program results in the execution of the instruction, that is the flow graph of an actual program is a VDL-graph:

Definition 6.2. The set of actual program is given by

$$\{t|\text{is-instr-graph}\ (t)\}$$

that is

$$\text{is-instr-graph} = (\{\langle s: \text{is-}\textit{stmt}\rangle|\text{is-select}\ (s)\}),$$
$$\text{is-}\textit{stmt} = (\langle s\text{-value}: \text{is-instr}\rangle, \langle s\text{-desc}: \text{is-select-list}\rangle)$$

where the predicate "is-*stmt*" is used instead of "is-node" in the definition of the VDL-graph.

Definition 6.3. Let

$$\text{is-code-list}\ (p) = T$$

and

$$\text{is-instr-graph}\ (t) = T.$$

The actual program t is a part of the program p if and only if

$$(\forall s,\ s(t) \neq \Omega)((\exists i)(\text{elem}\ (i)(p) = s(t)).$$

Definition 6.4. Let

$$\{a|\text{is-ass-instr}\ (a)\}$$

be the set of assembly form of instructions be considered. Let the function

$$\text{translator: } \{v|\text{is-instr}\ (v)\} \to \{a|\text{is-ass-instr}\ (a)\}$$

be given. Then the abstract inverse assembler is specified by the initial state of abstract machine

$$\xi_0 = \mu_0(\langle s\text{-input: } p\rangle, \langle s\text{-table: } t_0\rangle, \langle s\text{-control: } walk\ (;\ \text{next-selector, translator})))$$

where

$$\text{is-code-list}\ (p).$$

7. Conclusions and remarks

This paper can be viewed as a contribution towards the solution of some actual problem of program synthesis. The specifications used in this paper, describe the invariable properties of states of an abstract machine rather than the input-output relationship which is expected to be realized by the desired program. The same techniques can be applied to specify programs, which are never intended to terminate.

Our example demonstrates the application of deductive techniques for deriving program that manipulates the structure of complex data structures like list and graphs.

We have concerned with some aspect of transformation rules for achieving more than one goal simultaneously by checking the protection condition of interference-free.

Abstract

Our purpose in this paper is to illustrate a deductive technique for developing abstract programs systematically from given specifications using the Vienna Definition Language.

The role and the importance of program synthesis within the scope of programming methodology is emphasized. The basic principles and the main steps of a deductive technique for deriving programs systematically from their specifications is summarized.

Programming strategies are formulated for attempting to transform the specifications into a desired VDL-program and the technique is illustrated by the example of an abstract graph walk algorithm. The example includes the definition of an abstract data graph too.

An abstract linkage editor and a general inverse assembler model are given by specifying the graph walk.

KEYWORDS: Abstract data structures, derivation of programs, program verification, program synthesis, programming methodology, Vienna Definition Language.

DEPT. OF MATHEMATICS
L. EÖTVÖS UNIVERSITY
MÚZEUM KRT. 6—8.
BUDAPEST, HUNGARY
H—1088

References

[1] DIJKSTRA, E. W., *A discipline of programming,* Prentice Hall, Englewood Cliffs, New Jersey, 1976.
[2] DERSHOVITZ, N. and Z. MANNA, On automating structured programming, *Proc. IRIA Symp. Proving and Improving Programs,* Arc. et. Senans, France, July 1975, pp. 167—193.
[3] HOARE, C. A. R., An axiomatic basis of computer programming, *Comm. ACM,* v. 12, 1969, pp.576—583.
[4] HOARE, C. A. R., Parallel programming: An axiomatic approach, *Computer Languages,* v. 1, No. 2, 1975, pp. 151—160.
[5] LONDON, R. L., A view of program verification, *Proc. International Conference on Reliable Software,* April 1975, pp. 534—545.
[6] MANNA, Z., *Mathematical theory of computation,* New York, McGraw-Hill, 1974.
[7] MANNA, Z. and R. WALDINGER, The logic of computer programming, *IEEE Trans. Software Engrg.,* v. 4, 1978, pp. 199—229.
[8] MANNA, Z. and R. WALDINGER, The automatic synthesis of recursive programs, *SIGPLAN Notices,* v. 12, No. 8, 1977, pp. 29—36.
[9] MANNA, Z. and R. WALDINGER, The synthesis of structure-changing program, *Proc. 3rd International Conference on Software Engineering,* Atlanta, Georgia, USA, May 10—12, 1978, pp. 175—187.
[10] MILLS, H. D., How to write correct programs and know it, *Proc. International Conference of Reliable Software,* Los Angeles, Calif., April 1975, pp. 363—370.
[11] OWICKI, S. and D. GRIES, Verifying properties of parallel programs: An axiomatic approach, *Comm. ACM,* v. 19, 1976, pp. 279—285.
[12] RAMAMOORTHY, C. V. and S. B. F. Ho, Testing large software with automated software evaluation systems, *IEEE Trans. Software Engrg.,* v. 1, 1975, pp. 46—58.
[13] RAYMOND T. YEH (Ed.), *Current trends in programming methodology,* Vol. 2. Prentice Hall, Inc., Englewood Cliffs, New Yersey, 1977.
[14] WEGNER, P., Research directions in software technology, *Proc. 3rd International Software Engineering Conference,* Atlanta, Georgia, USA, May 10—12, 1978.
[15] WEGNER, P. (Ed.), *Research directions in software technology,* MIT Press, 1979.
[16] WEGNER, P., The Vienna Definition Language, *Comput. Surveys,* v. 4, 1972, pp. 5—63.
[17] WIRTH, N., On the composition of well-structured programs, *Comput. Surveys,* v. 6, 1974, pp. 247—259.

(Received Oct. 24, 1979)

On optimal performance in self-organizing paging algorithms

By R. I. PHELPS* and L. C. THOMAS**

Introduction

A large body of literature has grown up concerned with paging algorithms [e.g. 1, 3, 4, 6, 10]. In the terminology of Arato [2] a program's address space is divided into equal size blocks called page. There are two levels of memory: the first level is a fast access device and the second level is a larger, slower backing store. These levels are each divided into page frames. If a program references a page in the second level a page fault is said to occur. In this case the referenced page is brought into the first level and to make room for it another page is selected by the paging algorithm to be removed from the first to the second level.

The objective of the algorithm is to minimize the expected number of page faults. Most authors have considered this problem when either the probability distribution of the string of program references is known or when the algorithm stores information on the number of times each page has been referenced in order to estimate the distribution [1, 2, 3, 4, 10].

Interest has recently been shown in a different type of paging algorithm, when the distribution is unknown and no information on page references is collected [6, 7]. The page references are assumed to form an independent sequence. These are self-organizing algorithms. Since no information about the reference probabilities is available they can only select the page to be removed from first level memory on the basis of its position in the memory. The advantages of this approach are that no prior knowledge of reference probabilities is required, and no memory needs to be used in collecting information on these probabilities changing, the algorithm will automatically adjust to the new distribution.

The problem of finding the optimal self-organizing paging algorithm is related to another problem that has recently received much attention: there is a linear array of n storage positions $1, \ldots, n$, containing n pages $I_1, I_2, \ldots, I_n$, together with a self-organizing algorithm τ. τ acts so that if the page referenced was in position j it is moved to position $\tau(j)$. If $\tau(j)<j$ then the pages in positions $j-1$ to $\tau(j)$ are all moved back one position each to make room for the page moved from j to $\tau(j)$. If $\tau(j)>j$, then the pages in positions $j+1$ to $\tau(j)$ all move forward one position each. No other pages are moved.

The objective of the algorithm is to minimize the cost, which is the asymptotic expected position of the next page referenced. This is known as the library problem.

In this context the algorithm $\tau(j)=1,\ 1\leqq j\leqq n,$ known as the "move to the front" algorithm, has been studied by several authors [5, 8, 11, 12, 13]. McCabe [13] found an expression for the cost and for its variance. Hendricks [8] gives the stationary distribution of arrangements of pages in the store and Burville and Kingman [5] show that the cost is less than $2m-1$, where m is the cost if the reference distribution is known and the pages arranged optimally. Letac [12] considers the extension of this system in an infinite set of storage positions. Hendricks [9] gives stationary distributions for the algorithms $\tau(j)=k,\ 1\leqq j\leqq n.$

McCabe [13] suggested that the algorithm $\tau(j)=j-1,\ 2\leqq j\leqq n,\ \tau(1)=1,$ the 'transposition' algorithm, would have a cost less than that of the move to the front algorithm. Rivest [14] gives the stationary distribution of the transposition algorithm and shows that the cost of the transposition is always less than or equal to the cost of move to the front. He suggested that transposition is optimal for all reference distributions and supports this with simulation results. Thomas [15] proves that if such an optimal algorithm exists, it must be the transposition.

Returning to the specific paging algorithm setting with two levels of memory, we take the first level to have a capacity of M pages and the second level to have capacity $n-M$ pages. The page frames in the first level are numbered 1 to M and those in the second level $M+1$ to n. We can only consider paging algorithms of form $\tau(i)\leqq M,\ i>M.$

To minimize the number of page faults we take the following cost structure. A cost of 1 is incurred whenever a page in any of the positions $M+1$ to n is referenced. The cost is 0 otherwise. From among the possible paging algorithms, the closest to the transposition algorithm is the paging algorithm 'CLIMB' for which

$$\tau(j)=\begin{cases} M, & j>M \\ j-1, & 2\leqq j\leqq M \\ 1, & j=1 \end{cases}$$

Franaszek and Wagner [6] suggest that CLIMB is the optimal self-organizing paging algorithm for all reference distributions.

The purpose of this paper is to provide supporting analytical evidence for these suggestions by showing that in the special case of any reference distribution of form $p_1=ka, p_2=\ldots=p_n=a,$ where p_i is the reference probability of page I_i, transposition and CLIMB are indeed optimal for their respective cost functions.

1. Optimality for the special case of library problem

The intuitive justification for self-organizing algorithms is that when a page is referenced its posterior reference probability will increase. Since it is clear that it is optimal to have the pages with higher probability in the first positions, it follows that when a page has been referenced it should be moved forward. In this way the pages which are referenced most will tend to be moved into the first positions.

Because of this we restrict attention to algorithms for which $\tau(j)\leqq j,\ 1\leqq j\leqq n,$ and initially we will consider only 'forward moving' algorithms, that is $\tau(j)<j,$ $1\leqq j\leqq n.$ (The results can easily be extended to $\tau(j)\leqq j,$ as will be indicated later.)

With the reference distribution $p_1=ka, p_2=\ldots=p_n=a$, pages $I_2, \ldots, I_n$ have the same probability of reference and hence are equivalent as far as the self-organising system is concerned, and it only has n different states, depending on the position of page I_1. Any algorithm τ then gives rise to a Markov chain with n states, state i corresponding to I_1 being in the i^{th} position. Let P^τ be the one step transition matrix for this algorithm, whose elements p_{ij}^τ are the probabilities that referencing one page and applying τ changes the system from state i to state j. The limiting "steady-state" probabilities for each state under τ are $\underline{\pi}^\tau=(\pi_1^\tau, \pi_2^\tau, \ldots, \pi_n^\tau)$ where $\underline{\pi}^\tau=\underline{\pi}^\tau P^\tau$, provided this exists. Thus the average expected position of the next page referenced is

$$a \sum_{i=1}^{n} \pi_i^\tau \left[\frac{1}{2} n(n+1)+i(\alpha-1)\right].$$

We will show that the transposition algorithm, which we denote as T, where $T(1)=1$, $T(j)=j-1$, $2 \leqq j \leqq n$, minimises this expected position among all algorithms τ, where $\tau(j)<j$, $j \neq 1$ and $\tau(1)=1$.

For any algorithm τ of $n+1$ positions, define an algorithm $D\tau$ on n positions by

$$D\tau(i) = \begin{cases} \tau(i+1)-1 & \text{if } \tau(i+1) \neq 1, \\ 1 & \text{if } \tau(i+1) = 1. \end{cases}$$

As an example, suppose τ is defined on four positions with $\tau(1)=1$, $\tau(2)=1$, $\tau(3)=1$, $\tau(4)=3$. If the probability of referencing I_1 is ka and of referencing the others is a, then

$$P^\tau = \begin{pmatrix} ka+a & 2a & 0 & 0 \\ ka & 2a & a & 0 \\ ka & 0 & 2a & a \\ 0 & 0 & ka & 3a \end{pmatrix}.$$

The corresponding algorithm $D\tau$ on 3 positions satisfied $D\tau(1)=1$, $D\tau(2)=1$, $D\tau(3)=2$, and if the probabilities of referencing I_1, I_2, I_3 are ka', a', a' respectively, then

$$P^{D\tau} = \begin{pmatrix} ka'+a' & a' & 0 \\ ka' & a' & a' \\ 0 & ka' & 2a' \end{pmatrix}.$$

It can be shown (Appendix 1) that

$$\pi_i^\tau = (1-\pi_1^\tau)\pi_{i-1}^{D\tau}, \quad i = 2, \ldots, n+1, \ldots \tag{1}$$

Lemma 1. For any forward moving algorithm τ on n positions with $p_1=ka$, $p_2=p_3=\ldots=p_n=a$,

$$\text{(i)} \quad k \leqq \frac{\pi_i^\tau}{\pi_n^\tau} \leqq k^{n-i} \quad \text{if} \quad k \geqq 1, \tag{2}$$

$$\text{(ii)} \quad k \geqq \frac{\pi_i^\tau}{\pi_n^\tau} \geqq k^{n-i} \quad \text{if} \quad k \leqq 1. \tag{3}$$

Proof. Appendix.

Lemma 2. With the conditions of lemma 1, $\pi_1^T \geqq \pi_i^\tau$, $1 \leqq i \leqq n$.

Proof. Appendix.

Theorem 1. For any reference distribution on pages $I_1, \ldots, I_n$ of the form $p_1 = ka$, $p_2 = \ldots = p_n = a$, then amongst all algorithms τ of the form $\tau(1) = 1$, $\tau(i) < i$, the transposition algorithm minimises the expected position of the next page referenced.

Proof. We proceed by induction on the number of positions. The result is true if there are only two positions, as T is the only forward moving algorithm. So suppose the theorem is true for n-positions and consider a forward moving algorithm on $n+1$ positions. The expected position under algorithm τ is

$$\frac{1}{2}(n+1)(n+2)a + a(k-1)\sum_{i=1}^{n+1} i\pi_i^\tau.$$

Thus we want to show

$$\sum_{i=1}^{n+1} i\pi_i^T \leqq \sum_{i=1}^{n+1} i\pi_i^\tau \quad \text{if} \quad k \geqq 1$$

and

$$\sum_{i=1}^{n+1} i\pi_i^T \geqq \sum_{i=1}^{n+1} i\pi_i^\tau \quad \text{if} \quad k \leqq 1$$

for any algorithm τ. Using (1) we have

$$\sum_{i=1}^{n+1} i\pi_i^\tau = \pi_1^\tau + \sum_{i=2}^{n+1} i(1-\pi_1^\tau)\pi_{i-1}^{D\tau} = \pi_1^\tau + (1-\pi_1^\tau)\sum_{i=1}^{n} \pi_i^{D\tau} + (1-\pi_1^\tau)\sum_{i=1}^{n} i\pi_i^{D\tau}$$

$$= 1 + (1-\pi_1)\sum_{i=1}^{n} i\pi_i^{D\tau}. \tag{4}$$

Assume $k \geqq 1$, then by the induction hypothesis $\sum_{i=1}^{n} i\pi_i^{DT} \leqq \sum_{i=1}^{n} i\pi_i^{D\tau}$, since DT is in fact transportation on $n-1$ items. Also from lemma 2, $\pi_1^T \geqq \pi_1^\tau$. Hence the induction is completed.

2. Optimal paging algorithm for the special case

We can now use these results to prove the optimality of CLIMB as a paging algorithm. We consider a 2-level memory with M page frames in level 1 and $n-M$ page frames in level 2. We denote the algorithm CLIMB by C.

Lemma 3. $\pi_1^C \geqq \pi_i^\tau$, $1 \leqq i \leqq n$, for all τ of form $\tau(1) = 1$, $\tau(i) < i$, $2 \leqq i \leqq M$; $\pi(i) \leqq M$, $i > M$; $M < n$.

Proof. Appendix 4.

Theorem 2. For any reference distribution $p_1=ka$, $p_2=\ldots=p_n=a$, with M page frames in first-level memory, $M<n$, the algorithm CLIMB asymptotically minimizes the number of paging faults among all algorithms of form $\tau(1)=1$, $\tau(i)<i$, $2\leqq i\leqq M$; $\tau(i)\leqq M$, $i>M$.

Proof. We use induction on n. If $n=2$, then independent of whether $M=1$ or 2, CLIMB is the only policy in the class we are dealing with. Moreover, for any n, if $M=1$, then again CLIMB is the only possible policy. Assume the results holds for n positions, and look at the case with $n+1$ positions and first level memory of $M+1$ items, $M>1$. For any algorithm τ in this class, $D\tau$ is an algorithm on n items with a first level memory of M items. The expected paging cost using algorithm τ is

$$\sum_{i=1}^{M+1}\pi_i^{\tau}(n-M)a+\sum_{i=M+2}^{n+1}\pi_i^{\tau}(n-M+k-1)a=(n-M)a+(k-1)a\sum_{i=M+2}^{n+1}\pi_i^{\tau}.$$

Hence we need to show that

$$\sum_{i=M+2}^{n+1}\tau_i^C\leqq\sum_{i=M+2}^{n+1}\pi_i^{\tau}.$$

By (1)

$$\sum_{i=M+2}^{n+1}\pi_i^{\tau}=(1-\pi_1^{\tau})\sum_{i=M+1}^{n}\pi_i^{D\tau}.$$

The induction hypothesis implies that $\sum_{i=M+1}^{n}\pi_i^{DC}\leqq\sum_{i=M+1}^{n}\pi_i^{D\tau}$, and lemma 3 gives $\pi_1^C\geqq\pi_1^{\tau}$, so $\sum_{i=M+1}^{n+1}\pi_i^C\leqq\sum_{i=M+2}^{n+1}\pi_i^{\tau}$.

Other algorithms. Two other paging algorithms which can be run in a self organizing manner are [6] namely:

Least Recently Used — where the page which is moved from first-level memory is the one least recently referenced. This corresponds to an algorithm $\tau(j)=1$, $1\leqq j\leqq n$, which is forward moving and hence inferior to CLIMB at least for this reference distribution.

First In, First Out — where the one to leave is the one which arrived first, of those presently in the first level. This is an algorithm where

$$\tau(i)=1,\quad i=1,\ldots,M,\quad \tau(j)=1,\quad j>M.$$

This algorithm does not seem to be covered by the theorem. However, all the above results can be extended to include the cases where $\tau(j)\leqq j$. The difficulty is that one can then have algorithms which give rise to a Markov chain in which it is not possible to get from some states to others. Thus the 'steady state' probabilities depend on the initial ordering of the pages. However, if we assume that the initial ordering is equally likely to be any of the possible orderings, the above results still hold. This is because any set of connected states corresponds to the page with reference probability ka being in a set of consecutive positions, and an overall transposition algorithm is better than one which is a transposition algorithm on

each set individually. Thus Theorems 1 and 2 can be extended to allow algorithms where $\tau(j) \leqq j$.

APPENDIX 1. Proof of $\pi_i^\tau = (1-\pi_1^\tau)\pi_{i-1}^{D\tau}$, $i=2, \ldots, n+1$.

For a forward moving τ, let $S(i|\tau) = |\{r | r > i, \tau(r) \leqq i\}|$, where $|A|$ denotes the cardinality of the set A. Let

$$T(i|\tau) = \{r | r > i, \tau(r) = i\}.$$

The $i+1^{\text{th}}$ component of the equation $\underline{\pi}^\tau = \underline{\pi}^\tau P^\tau$ then reads

$$\pi_{i+1}^\tau = \Big(ka \sum_{r \in T(i+1|\tau)} \pi_r^\tau\Big) + \pi_{i+1}^\tau\big(1-ka-aS(i+1|\tau)\big) + \pi_i^\tau aS(i|\tau), \quad i \geqq 1. \tag{1.1}$$

This follows because as τ is forward moving a page will only move if it is referenced or if the referenced page moves from behind it to in front of it. Thus I_1 is in the $i+1^{\text{th}}$ position, because either it was in the r^{th} position previously and was referenced, where $\tau(r)=i+1$, or it was in the $i+1^{\text{th}}$ position and the page referenced did not move it, or was in the i^{th} position and the page referenced moved in front of it. Thus 1.1 becomes

$$S(i|\tau)\pi_i^\tau + \pi_{i+1}^\tau\big(k+S(i+1|\tau)\big) - k \sum_{r \in T(i+1|\tau)} \pi_r^\tau. \tag{1.2}$$

Applying the same procedure to the i^{th} component of $\underline{\pi}^D = \underline{\pi}^{D\tau} P^{D\tau}$, when the probabilities of referencing the pages are $(ka', a', \ldots, a')$ gives

$$S(i-1|D\tau)\pi_{i-1}^{D\tau} = \pi_i^{D\tau}\big(k+S(i|D\tau)\big) - k \sum_{r \in T(i|D\tau)} \pi_r^{D\tau}, \quad i \geqq 2. \tag{1.3}$$

By the definition of D it is obvious that

$$T(i|D\tau) = \{j-1 | j \in T(i+1|\tau)\}, \quad S(i|D\tau) = S(i+1|\tau), \quad i = 2, \ldots, n. \tag{1.4}$$

Thus if we identified $\pi_{i-1}^{D\tau}$ with π_i^τ, equations 1.2 and 1.3 would be the same.

Consider 1.2 for the case $i=n$. The right hand side of the equation can only contain terms in π_{n+1}^τ since τ is forward moving and so $T(n+1|\tau)$ is empty, while $S(n|\tau)$ is 1. We thus have a linear equation $\pi_n^\tau = K_n \pi_{n+1}^\tau$ for some constant K_n. Since 1.3 is identical with 1.2 except that $\pi_{i-1}^{D\tau}$ replaces π_i^τ throughout, in the corresponding case it becomes $\pi_{n-1}^{D\tau} = K_n \pi_n^{D\tau}$.

Now consider 1.2 with $i=n-1$. The right hand side can now only contain terms in π_n^τ and π_{n+1}^τ. We can substitute $K_n \pi_{n+1}$ for π_n^τ and so obtain another linear equation $\pi_{n-1}^\tau = K_{n-1}\pi_{n+1}^\tau$, for some K_{n-1}. The same argument implies that 1.3 will give $\pi_{n-2}^{D\tau} = K_{n-1}\pi_n^{D\tau}$. Repeating the procedure gives

$$\frac{\pi_{i-1}^{D\tau}}{\pi_n^{D\tau}} = \frac{\pi_i^\tau}{\pi_{n+1}^\tau} = K_i, \quad i = 2, \ldots, n. \tag{1.5}$$

Thus

$$\sum_{i=1}^{n} \pi_i^{D\tau} = \Big(\sum_{i=2}^{n} K_i + 1\Big)\pi_n^{D\tau} = 1$$

and

$$\sum_{i=2}^{n+1} \pi_i^\tau = \Big(\sum_{i=2}^{n} K_i + 1\Big)\pi_{n+1}^\tau = 1 - \pi_1^\tau$$

so $\pi^{\tau}_{n+1}=(1-\pi^{\tau}_1)\pi^{D\tau}_n$ and hence

$$\pi^{\tau}_i = (1-\pi^{\tau}_1)\pi^{D}_{i-1}, \quad i = 2, \ldots, n+1. \tag{1.6}$$

APPENDIX 2. Proof of Lemma 1.

Proof. Proceed by induction on n. The result is true when $n=2$ for the only algorithm satisfying the requirements is the transposition algorithm T where $T(1)=T(2)=1$, and $\pi^T_1/\pi^T_2=\alpha$. Assume the bounds hold for forward moving algorithms on n items, and look at τ, a forward moving algorithm on $n+1$ items. Since $D\tau$ is a forward moving algorithm on n items, (1) gives

$$\frac{\pi^{\tau}_i}{\pi^{\tau}_{n+1}} = \frac{(1-\pi^{\tau}_1)\pi^{D\tau}_{i-1}}{(1-\pi^{\tau}_1)\pi^{D\tau}_n} = \frac{\pi^{D}_{i-1}}{\pi^{D}_n}, \quad i = 2, \ldots, n \tag{2.1}$$

and so, because of the induction hypothesis,

$$k \leqq \frac{\pi^{\tau}_i}{\pi^{\tau}_{n+1}} \leqq k^{n-(i-1)} = k^{n+1-i} \quad \text{if} \quad k \geqq 1, \quad i = 2, \ldots, n. \tag{2.2}$$

Thus we only have to prove that $k \leqq \frac{\pi^{\tau}_1}{\pi^{\tau}_{n+1}} \leqq k^n$.

Consider the first component of the equation $\underline{\pi}^{\tau}=\underline{\pi}^{\tau}P$. Since $T(1|\tau)$ is the set of positions that are mapped on the first position by τ, page I_1 can only be in the first position if it was in one of the positions of $T(1|\tau)$ and was referenced, or if it was in the first position and the page referenced was not in the set $T(1|\tau)$. Then

$$\pi^{\tau}_1 = \Big(ka \sum_{r\in T(1|\tau)} \pi^{\tau}_r\Big)+\pi^{\tau}_1(1-a|T(1|\tau)|) \tag{2.3}$$

where $|T(1|\tau)|$ denotes the number of positions in the set $\{j|\tau(j)=1, j>1\}$. Thus $\frac{\pi^{\tau}_1}{\pi^{\tau}_{n+1}} = \frac{k}{|T(1|\tau)|} \sum_{r\in T(1|\tau)} \pi^{\tau}_r/\pi^{\tau}_{n+1}$. If $k\geqq 1$, (2.2) already gives $1 \leqq k \leqq \frac{\pi^{\tau}_r}{\pi^{\tau}_{n+1}} \leqq k^{n+1-r} \leqq$ $\leqq k^{r+1-2}$. Thus $k^n \geqq \frac{\pi^{\tau}_1}{\pi^{\tau}_{n+1}} \geqq k$ for $k\geqq 1$. The induction is complete and a similar proof works for $k<1$.

APPENDIX 3. Proof of Lemma 2.

Proof. Assume $k\geqq 1$ (the coresponding result for $k<1$ follows similarly), and assume that $\pi^T_1 \geqq \pi^{\tau}_i$ for algorithms acting on n positions. The result is trivially true for $n=2$, and so we proceed by induction. For $n+1$ positions, using (2.3) and (1) gives

$$\pi^{\tau}_1 = \Big(k \sum_{r\in T(1|\tau)} \pi^{\tau}_r\Big)/|T(1|\tau)| = \Big(k \sum_{r\in T(1|\tau)} \pi^{D\tau}_{r-1}(1-\pi^{\tau}_1)\Big)/|T(1|\tau)|. \tag{3.1}$$

By the inductive hypothesis, $\pi^{D\tau}_{r-1} \leqq \pi^{DT}_1$, so

$$\pi^{\tau}_1 \leqq \alpha\pi^{DT}_1(1-\pi^{\tau}_1). \tag{3.2}$$

Rivest's result [4] is that $\frac{\pi^T_i}{\pi^T_n}=k^{n-1}$, so $\pi^T_1=k\pi^T_2$, while (1) gives $\pi^T_2=\pi^{DT}_1(1-\pi^T_1)$.

Thus

$$\pi_1^T = k\pi_1^{DT}(1-\pi_1^T). \tag{3.3}$$

Thus we have

$$\frac{\pi_1^\tau}{1-\pi_1^\tau} \leqq \frac{\pi_1^T}{1-\pi_1^T}$$

and so

$$\pi_1 \leqq \pi_1^T. \tag{3.4}$$

For π_i^τ, $i=2, \ldots, n+1$ (1) gives

$$\pi_i^\tau = \pi_{i-1}^{D\tau}(1-\pi_1^\tau) \leqq \pi_1^{DT}(1-\pi_1^\tau) = \frac{\pi_1^T}{k}\frac{(1-\pi_1)}{(1-\pi_1^T)}. \tag{3.5}$$

So it is sufficient to show that

$$\frac{1-\pi_1}{1-\pi_1^T} \leqq k.$$

Writing $\pi_i = K_i\pi_{n+1}^\tau$ as before, we get

$$\pi_{n+1}^\tau = \left(1+\sum_{i=1}^{n} K_i\right)^{-1}.$$

Thus

$$1-\pi_1^\tau = \sum_{i=2}^{n+1}\pi_i^\tau = \pi_{n+1}^\tau\left(1+\sum_{i=2}^{n} K_i\right) = \frac{1+\sum_{i=2}^{n} K_i}{1+\sum_{i=1}^{n} K_i} \tag{3.6}$$

$$\leqq \frac{1+\max\sum_{i=2}^{n} K_i}{1+k+\max\sum_{i=2}^{n} K_i}. \tag{3.7}$$

The inequality follows since (3.6) is a maximum when K_1 is as small as it can be, which is k from Lemma 1, and the sum of the rest of the K_i are as large as they can be.

Using the inequalities of Lemma 1

$$\frac{1+\max\sum_{i=2}^{n} K_i}{1+k+\max\sum_{i=2}^{n} K_i} = \frac{1+\sum_{i=2}^{n} k^i}{1+k+\sum_{i=2}^{n} k^i} = \frac{k^n-1}{k^n+k^2-k-1}.$$

From Rivest's result $(1-\pi_1^T)=\frac{k^n-1}{k^{n+1}-1}$, so we have $(1-\pi_1)\leq k(1-\pi_1^T)$, since $\frac{k^n-1}{k^n+k^2-k-1}\leqq\frac{k(k^n-1)}{(k^{n+1}-1)}$ for k positive. This completes the induction

Appendix 4. Proof of Lemma 3.

Proof. The lemma is trivially true for $n=2$. Assume that it is true for up to n positions, and consider C and another such algorithm τ on $n+1$ positions. Since C acts as the transposition algorithm T, on the first $M+1$ positions, it follows that $\pi_1^C=k\pi_2^C$ just as $\pi_1^T=k\pi_2^T$. Thus (3.2), (3.3), (3.4) follow as in Appendix 3, replacing

T by C, giving $\pi_1^\tau \leqq \pi_1^C$. For $i=2, \ldots, n+1$, we have

$$\pi_i^\tau = \pi_{i-1}^{p\tau}(1-\pi_1^\tau) \leqq \pi_1^{pC}(1-\pi_1^\tau) = \frac{\pi_1^C}{k}\frac{(1-\pi_1^\tau)}{(1-\pi_1^C)}.$$

It is sufficient to show $\dfrac{1-\pi_1^\tau}{1-\pi_1^C} \leqq k$. As C is a forward moving algorithm the induction of theorem 1 gives $\pi_1^T \geqq \pi_1^C$ and so

$$\frac{1}{1-\pi_1^T} \geqq \frac{1}{1-\pi_1^C}.$$

So

$$\frac{1-\pi_1^\tau}{1-\pi_1^C} \leqq \frac{1-\pi_1^\tau}{1-\pi_1^T} \leqq k.$$

Abstract

A brief survey is given of developments in the study of self organizing paging algorithms and the associated library problem. It has been conjectured that two related algorithms, transposition and climb, are optimal in these fields and we establish this optimality for a specific distribution of page references.

* DEPARTMENT OF MATHEMATICS
POLYTECHNIC OF THE SOUTH BANK
LONDON

** DEPARTMENT OF DECISION THEORY
UNIVERSITY OF MANCHASTER
MANCHESTER

References

[1] AHO, A. V., P. J. DINNING, J. D. ULLMANN, Principles of optimal page replacement, *J. Assoc. Comput. Mach.*, v. 18, 1971, pp. 80—93.
[2] ARATÓ, M., A note on optimal performance of page storage, *Acta Cybernet.*, v. 2, 1976, pp. 25—30.
[3] BÉLÁDY, L. A., A study of replacement algorithms for a virtual storage computer, *IBM Systems J.*, v. 5, 1966, pp. 78—101.
[4] BENCZUR, A., A. KRÁMLI, J. PERGEL, On the Bayesian approach to optima performance of page storage hierarchies, *Acta Cybernet.*, v. 3, 1976, pp. 79—89.
[5] BURVILLE, P. J. and J. F. C. KINGMAN, On a model for storage and search, *J. Appl. Probab.*, v. 10, 1973, pp. 697—701.
[6] FRANASZEK, P. A. and T. J. WAGNER, Some distribution-free aspects of paging algorithm performance, *J. Assoc. Comput. Mach.*, v. 21, 1974, pp. 31—39.
[7] GELENBE, E., A unified approach to the evaluation of a class of replacement algorithms, *IEEE. Trans. Comput.*, v. 22, 1973, pp. 611—617.
[8] HENDRICKS, W. J., The stationary distribution of an interesting Markov chain, *J. Appl. Probab.*, v. 9, 1972, pp. 231—233.
[9] HENDRICKS, W. J., An extension of a theorem concerning an interesting Markov chain, *J. Appl. Probab.*, v. 10, 1973, pp. 886—890.
[10] INGARGIOLA, G. and J. F. KORSH, Finding optimal demand paging algorithms, *J. Assoc. Comput. Mach.*, v. 21, 1974, pp. 40—53.
[11] KNUTH, D. E., *The art of computer programming* (Volume 3: Sorting and Searching), Addison—Wesley Publishing Co. Reading, Mass., 1973.
[12] LETAC, G., Transcience and recurrence of an interesting Markov chain, *J. Appl. Probab.*, v. 11, 1974, pp. 818—824.
[13] MCCABE, J., On serial files with relocatable records, *Oper. Res.*, v. 13, 1965, pp. 609—618.
[14] RIVEST, R. L., On self-organising sequential search heuristics, *Comm. ACM*, v. 19, 1976, pp. 63—67.
[15] THOMAS, L. C., Optimal replacement of library type Markov chains, University of Manchester, Department of Decision Theory, Note No. 22, 1975.

(Received Oct. 24, 1979)

Dominant schedules of a steady job-flow pair*

By J. TANKÓ

A specific approach to some non-finite deterministic scheduling problems is the scheduling of a steady job-flow pair model. Its non-preemptive scheduling problem was discussed earlier [4]. The more general preemptive scheduling is discussed below. A very simple scheduling discipline leads to the dominant set of the so-called consistent economical schedules (CESs). The proof of dominance is the main goal of this article. An algorithm to evaluate the dominant schedules and choose an optimal one is given as well.

1. Introduction

In an earlier article [4] we defined the general scheduling model of steady job-flow pairs as a new approach to some non-finite deterministic scheduling problems. There we referred to the study [2] and to the dissertation [3] of the author dealing with this problem and to other works dealing with scheduling problems related to our problem. Some practical cases the model may be applicable in are mentioned there.

Some statements below bear some resemblance to those of non-preemptive scheduling [4] but, for example the cardinal of the dominant set, is not bounded as in the non-preemptive case. The task of determining the optimal schedule under the restriction of non-preemption is simpler than without this restriction. In a non-preemptive case the dominant set of the so-called consistent natural schedules have six elements maximum. These elements can be evaluated at once, e.g., by the method of reduction [4]. The general problem of determining or producing an optimal schedule (preemptive if necessary) for any steady job-flow pair is not completely solved until now.

We reduce below the set of feasible schedules to a dominant set of consistent economical schedules containing optimal schedules and give an algorithm to choose an optimal schedule by evaluation of the whole set if it is finite.

* This article reports on some results of a study of the author supported by the Computer and Automation Institute of the Hungarian Academy of Sciences.

2. Definitions

The scheduling problem of steady job-flow pairs is to schedule three processors $\mathscr{P}=(P_A, P_{B1}, P_{B2})$ to service, without conflicts, pairs $Q=(Q^{(1)}, Q^{(2)})$ of steady job-flows $Q^{(i)}=\{C_{ij}, j=1, 2, \ldots\}$ consisting of task-pairs $C_{ij}=(A_{ij}, B_{ij})$ with service demands η_i and ϑ_i on processor P_A and P_{Bi}, respectively. The order of servicing the tasks is strictly serial within job-flows but it is not restricted among job-flows. Conflicts might only be on the processor P_A and the efficiency of a scheduling R is measured by the utilization of the processor P_A. Define *P_A-utilization* of a section from time t_1 to time t_2 of a scheduling R by $\lambda(t_1, t_2)/(t_2-t_1)$ with *P_A-usage* $\lambda(t_1, t_2)$ as the sum of activity durations of P_A in the while from t_1 to t_2. Let $\lambda(t)=\lambda(0, t)$. The *efficiency* of a scheduling R is defined by the limit

$$\gamma = \gamma(R) = \lim_{t\to\infty} \frac{\lambda(t)}{t}. \tag{1}$$

The efficiency of any scheduling cannot be greater than 1 or the sum $\gamma^{(1)}+\gamma^{(2)}$ of the P_A-utilizations of the job-flows $Q^{(1)}$ and $Q^{(2)}$ which are given by $\gamma^{(i)}=\eta_i/\tau_i$, $i=1, 2$. We use the notations

$$\tau_i = \eta_i+\vartheta_i, \quad i = 1, 2, \quad \eta = \eta_1+\eta_2, \quad \vartheta = \vartheta_1+\vartheta_2.$$

The scheduling procedure is a decision process determining for all moment $t\geqq 0$ and state of processors and job-flows the way of continuation of the servicing process. The plan or result of a scheduling procedure is a *schedule* R as an ordered set of situations σ. The *situation* σ characterises the state of processors, the state of demand cycles under service, if any, of both job-flows and the duration of these states in a given phase of the scheduling.

Two components of σ are the functions $\beta^{(i)}(t)$, $i=1, 2$, $t\geqq 0$, the value of $\beta^{(i)}(t)$ being the demand not served yet from the demand cycle started but not finished (active), if it exists, of the job-flow $Q^{(i)}$, and 0 otherwise.

A schedule is *consistent* if the scheduling decision is the same when the situation σ has the same value. A schedule is *tight* if processor is never idle when demand it could serve exists. A schedule is *non-preemptive* if the service of every task finishes without breaks after its beginning. The specific class of non-preemptive schedules is discussed in [4]. Here now we allow the service of a task to be preempted and resumed later on the same processor.

The instance of a scheduling problem is fully determined by the values $Q=(\eta_1; \vartheta_1; \eta_2; \vartheta_2)$ of the service demands of tasks type A_1, B_1, A_2, B_2, respectively. $\eta_1, \vartheta_1, \eta_2, \vartheta_2$ are called *parameters* and the quaternaries Q are called *configurations*. The non-negative sixteenth $\mathscr{Q}$ of the four-dimensional Cartesian space constitutes the *configuration space*. The goal of the study of the model defined is to find a method for choosing a schedule R^* for every configuration $Q\in\mathscr{Q}$ for which $\gamma(R^*)$ exists and has the maximum value among all the feasible schedules. This schedule is called an *optimal schedule*. Simple method for finding optimal schedule for all $Q\in\mathscr{Q}$ i.e. an optimal scheduling strategy is not found yet.

Two schedules R and R' are *essentially-the-same* and denoted by $R\approx R'$ if they are congruent after some finite initial sections of them. $\gamma(R)=\gamma(R')$ if $R\approx R'$. The schedule R' *dominates* the schedule R if for the efficiency values $\gamma(R')$ and $\gamma(R)$

defined by (1) the relation $\gamma(R') \geqq \gamma(R)$ is true. The set $\mathscr{R}'$ of schedules is a *dominant set* if for every feasible schedule R there exists an $R' \in \mathscr{R}'$ dominating it.

Looking for an optimal schedule the investigation of a dominant set $\mathscr{R}'$ is enough for. We obtain a dominant set of schedules by means of the concept of the dominant decision.

The scheduling *decision* s' *dominates* s in a situation σ if the minimal next following cycle-finishes of both job-flows are not later by s' than by s. A *decision* s is *economical* if decision s' dominating it does not exist (see Fig. 2 below). A schedule R is an *economical schedule* (ES) if the scheduling decisions in its every situation are economical. Let $\mathscr{R}(Q)$ denote the class of all economical schedules for the configuration $Q \in \mathscr{Q}$. Let $\mathscr{R} = \bigcup_{Q \in \mathscr{Q}} \mathscr{R}(Q)$. We will show that $\mathscr{R}$ is a dominant set of schedules.

3. Economical schedules

The importance of the economical schedules (ESs) lies in their dominance which we show below.

Theorem 1. *The class* $\mathscr{R}$ *of economical schedules constitutes a dominant set.*

Proof. Let R be any feasible schedule having scheduling decisions not economical. Let s be a not economical decision in the situation σ of R. There exists an economical decision s' in σ dominating s because s would be economical decision otherwise. By exchanging s for s' both the next following cycle-ends could come forward and this eventually makes possible to anticipate all cycle-ends. This transformation does not diminish the function $\lambda(t)$ and, consequently, γ in (1). The new schedule R' obtained by this transformation dominates R as a result. Starting from $t=0$ and initial situation $\sigma=\sigma_0$, we can construct a dominating ES R' for any feasible schedule R. This was to be proven. □

The class $\mathscr{R}$ is a true part of the set of all feasible schedules but it can be very big to choose an optimal schedule by direct evaluations. To show this and to look for further reduction of the dominant set we investigate the characteristics of the ESs.

It is easy to be seen that the economical decision is unique in all situations σ except an enumerable set of situations for every ES. The exceptional situations are called *critical situations*. The economical decisions made in this situations are defined as *critical decisions*. The initial situation σ_0 of every schedule and the initial decision s_i, $i=1, 2$, for servicing the task A_{i1} first, are always critical but we mean by first critical situation of an ES the next one if it exists. Fig. 1 shows the types of critical situations and the possible alternative critical decisions. These and their conditions are the following:

Type	Decisions	Conditions
σ_0	s_1, s_2	$\beta^{(1)}(t) = \beta^{(2)}(t) = 0$
$\sigma_{i,1}$	s_0, s_i	$\beta^{(i)}(t) = 0, \quad \vartheta_{3-i} < \beta^{(3-i)}(t) < \tau_{3-i}, \quad i = 1, 2$

Fig. 2 illustrates the dominance of scheduling decisions. The graphs (a) and (b) illustrate that the idleness of a processor cannot be a dominating decision if

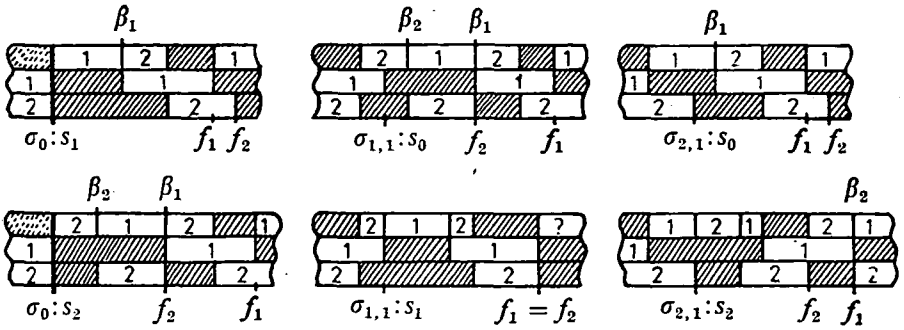


Fig. 1
Critical situations and decisions

demand waiting for service does exist. The graphs (c)—(d) show that the decisions s_i' causing preemption for not a complete service of the preempting task are not dominant as well. The graph (e) shows the non-dominance of the preemption of a preempting task.

It follows that the ESs are tight, usually preemptive schedules but have no superfluous preemptions. Only cycle-ends f_i can be critical situations and they really are if the processor P_A is busy or demanded simultaneously by the other

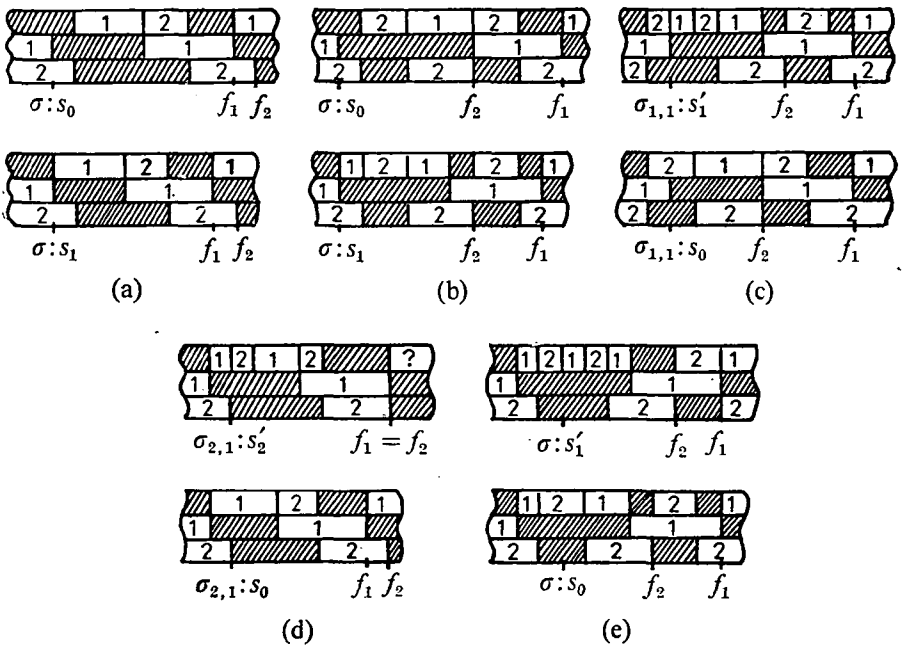


Fig. 2
Dominating decisions

job-flow. Preemption can only occur in critical situations and every critical decision causes a delay of the service of the job-flow not preferred by the decision. Delay is not caused by decisions other than critical. Between critical situations the sections of any ES are uniquely determined by the initial situation and decision. These sections are, therefore, called *determined sections*. The infinite section starting with the last critical situation if it exists, is the *last* determined section.

All ESs start with the service of the task A_{i1} without preemption in the interval $(0, \eta_i)$ in accordance with the initial decision s_i, $i=1, 2$. Accordingly, the class $\mathscr{R}$ bursts into two subclasses $\mathscr{R}^{(i)}$, $i=1, 2$, consisting of ESs with the initial decisions s_i, $i=1, 2$, respectively. The initial decision s_i uniquely determines the first determined section together with the closing critical situation — the first — if it exists. It follows that all elements of $\mathscr{R}^{(i)}(Q)$ have the same first determined sections and critical situations σ_i' if the latters exist at all. Let T_i' be the length of the first determined section. There is no preemption and delay on the first determined section except the initial delay of $Q^{(3-i)}$ in the interval $(0, \eta_i)$. Use the notation $\sigma^{(i)}$ for the situation of schedules $R \in \mathscr{R}^{(i)}$ in the point $t_i'=\eta_i$.

The concepts of critical situation and decision were introduced for the natural schedules defined in [4] as well. The types of critical situations were σ_0 and $\sigma_{i,0}$, $i=1, 2$, and the conditions for σ_0 were the same as here. The conditions of $\sigma_{i,0}$ there and the Fig. 1 show that a situation type $\sigma_{i,1}$ in ESs is always preceded by a situation type $\sigma_{3-i,0}$ being critical situation of a natural schedule but not of an economical one. This simultaneousness of $\sigma_{3-i,0}$ and $\sigma_{i,1}$ has a particular importance at the first determined sections playing a central role in the discussion of ECs (see Theorem 2). Out of types σ_0, $\sigma_{i,0}$ and $\sigma_{i,1}$ the natural and economical decisions are the same for every situation and cause no preemptions or delays. The first determined sections for the ESs are, therefore, almost the same as for the natural schedules. The differences are only in the last subsections of the ESs starting with $\sigma_{3-i,0}$ and ending with $\sigma_{i,1}$. The processor P_A is busy throughout the subsections. If the first critical situation does not exist, the set $\mathscr{R}^{(i)}$ consists of a single schedule R_{i0} being natural schedule, simultaneously.

The connection between the first critical situations of the natural and economical schedules allow us to simply prove an important theorem concerning typical situations by reference. *Typical situations* of an ES are defined as its critical situations and the β_i-situations which are not $\sigma^{(i)}$ situations directly following critical situations [4]. *β_i-situation* is a situation in which an A_i-task finishes and an A_{3-i}-task starts at the same moment. Let σ_i^* denote the first typical situation of the ESs of $\mathscr{R}^{(i)}(Q)$ if it exists. The possible first typical situations are illustrated in Fig. 3. We also use the wording *characteristic situations* for the critical and every β_i-situations.

Theorem 2. *In one and the same cases all elements of $\mathscr{R}^{(a)}(Q)$ have a first typical situation σ_a^* iff the simultaneous inequalities*

$$0 \leqq \Delta_a \leqq \eta, \quad \omega_a \geqq (1, 0) \tag{2}$$

have a solution, where $\omega_a=(B_a, A_a)$ are integers and $\Delta_a \equiv B_a\tau_a - A_a\tau_{3-a}$, $a=1, 2$.

When (2) has no solution, $\mathscr{R}^{(a)}(Q)$ consists of the single (non-preemptive and consistent) schedule R_{a0}. This occurs in the cases

$$\eta=0, \ \vartheta_1 \text{ and } \vartheta_2 \text{ are rationally independent} \tag{3}$$

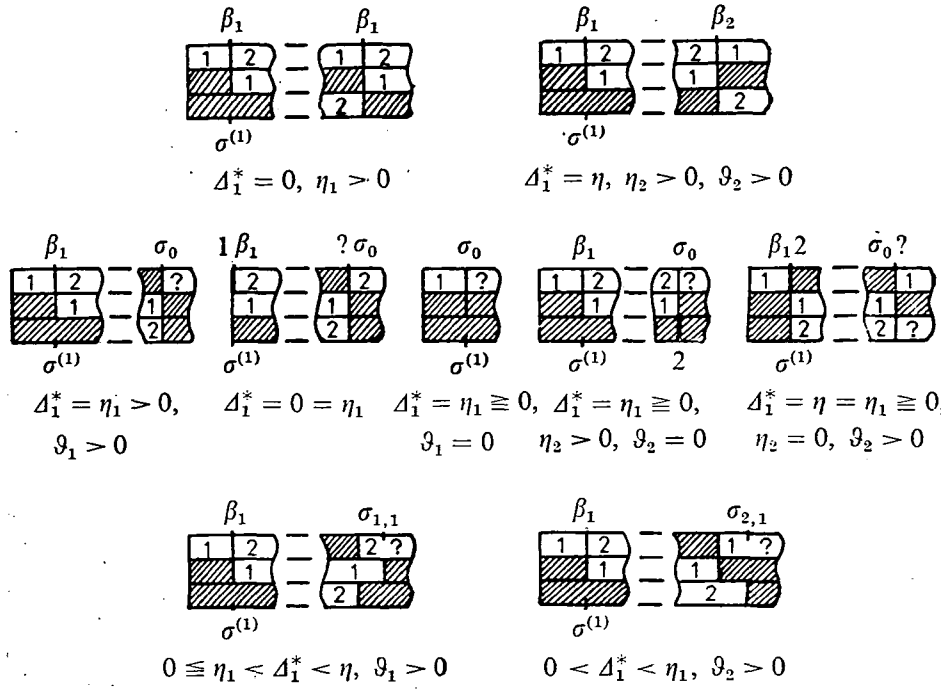


Fig. 3
First typical situations and their conditions ($\Delta_1^* \equiv B_1^* \tau_1 - A_1^* \tau_2$)

and

$$\vartheta_a > 0, \quad \tau_{3-a} = 0. \tag{4}$$

When (2) *has a solution, the type (and place) of* σ_a^* *is determined by the error* Δ_a^* *of the least solution* $\omega_a^* = (B_a^*, A_a^*)$ *of* (2) *according to the table*

σ_a^*	Conditions
β_a	$\Delta_a^* = 0 < \eta_a$
β_{3-a}	$\Delta_a^* = \eta > \eta_a, \quad \vartheta_{3-a} > 0$
σ_0	$\Delta_a^* = \eta_a$ *or* $\Delta_a^* = \eta > \eta_a$ *but* $\vartheta_{3-a} = 0$
$\sigma_{a,1}$	$\eta_a < \Delta_a^* < \eta$
$\sigma_{3-a,1}$	$0 < \Delta_a^* < \eta_a$

Proof. The assertions of the theorem follow from Theorem 4 of the article [4] and the comments made above. □

The problem of finding the least solution of (2) is a coincidence problem [2].

If $\sigma^{(a)}$ is not a critical situation, it is always a β_a-situation. It follows that β_a returns periodically and σ_a' does not exist if $\sigma_a^* = \beta_a$. If $\sigma_a^* = \beta_{3-a}$ then the first

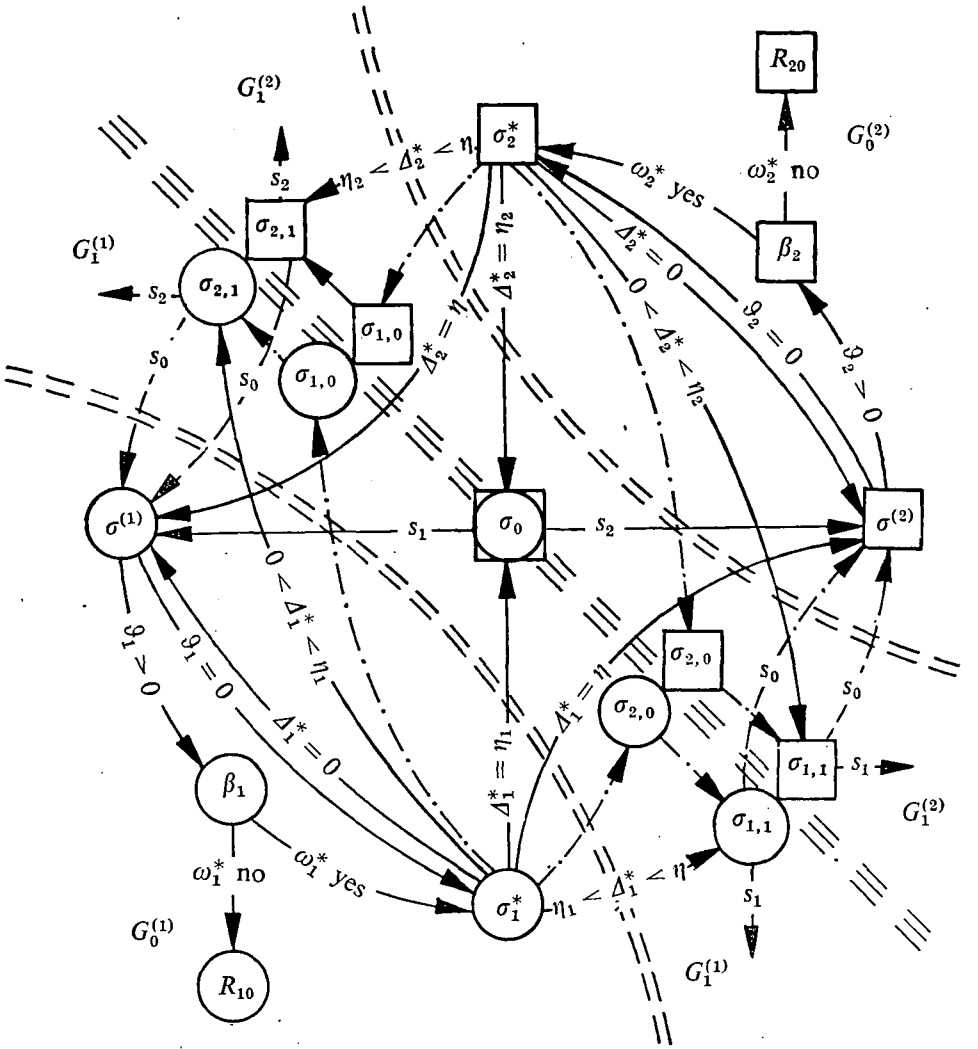


Fig. 4
The cyclic graph G_0 of the first determined sections

determined section of $\mathscr{R}^{(a)}(Q)$ from its β_{3-a}-situation on is congruent with the first determined section of $\mathscr{R}^{(3-a)}(Q)$ from its $\sigma^{(3-a)}=\beta_{3-a}$-situation on.

The assertions of Theorem 2 are well illustrated by the cyclical graph G_0 of Fig. 4 showing the possible characteristic situations of the first determined sections of ESs. The vertices of the graph represent situations and the (directed) arcs successions or identities. The arcs are labeled by critical decisions after critical situations and by conditions for Δ_a^* and the parameters after other vertices. The vertices framed by circles or squares can be the situations of $\mathscr{R}^{(1)}$ and $\mathscr{R}^{(2)}$, respectively, until the

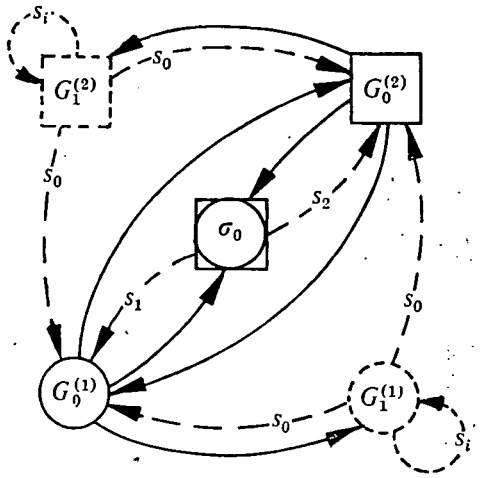


Fig. 5
The partitioning of the graph G_0

first typical situations. The graph G_0 represents all the possible cases for the whole configuration space $\mathcal{Q}$. For every $Q \in \mathcal{Q}$ only one arc going from a not critical situation is right. The graph can be partitioned into four subgraphs by Fig. 5. On the graphs the results of the decisions in the first critical situations are drawn by broken arcs.

Before we investigate further characteristic situations of the ESs, we show an example by Fig. 6. The part (a) shows the Gantt-chart of an $R \in \mathcal{R}^{(1)}(Q)$, the part (b) is the graph $G_0(Q)$ and the part (c) illustrates the graph $G(Q)$ of the ESs of $\mathcal{R}(Q)$.

EXAMPLE. $Q=(4.5;\ 3.5;\ 1;\ 2)$, $\tau_1=8$; $\tau_2=3$, $\eta=5.5$, $\vartheta=5.5$.

$\omega_1^*=(1, 1)$, $\Delta_1^*=5 \in (4.5;\ 5.5)$ and so $\sigma_1^*=\sigma_{1,1}$.

$\omega_2^*=(1, 0)$, $\Delta_2^*=3 \in (1;\ 5.5)$ and so $\sigma_2^*=\sigma_{2,1}$.

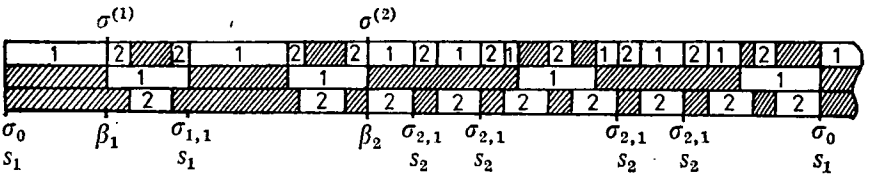
It is seen that always the characteristic situation $\sigma^{(3-a)} \in G_0$ occurs after the critical decision s_0 in a critical situation type $\sigma_{a,1}$. This means that new characteristic situation value can only be generated by decision s_i in a situation type $\sigma_{i,1}$. The type of the generated critical situation can be either of $\sigma_{j,1}$, $j=1, 2$, σ_0 and β_j, $j=1, 2$. The situations except type $\sigma_{j,1}$ are not new and lead back into the subgraph G_0. But the generated critical situation value must be new if its type is $\sigma_{j,1}$, $j=1, 2$. This is the consequence of the fact that determined sections are determined by their closing critical situations as well. Returning of an earlier $\sigma_{j,1}$ value after $\sigma_{i,1}$ would contradict this fact.

All the possibilities of the ES elements $R \in \mathcal{R}$ can well be illustrated by G_0 and the further critical situations according to the graph G on Fig. 7. The vertices $\sigma_{i,1}$ all illustrate different values of critical situations of type $\sigma_{1,1}$ and $\sigma_{2,1}$ independently of each other. The graph G is composed from five subgraphs by Fig. 7/b. $G_1^{(a)}$, $a=1, 2$, are the *branches* of G. The number of different vertices of G is infinite as we show below.

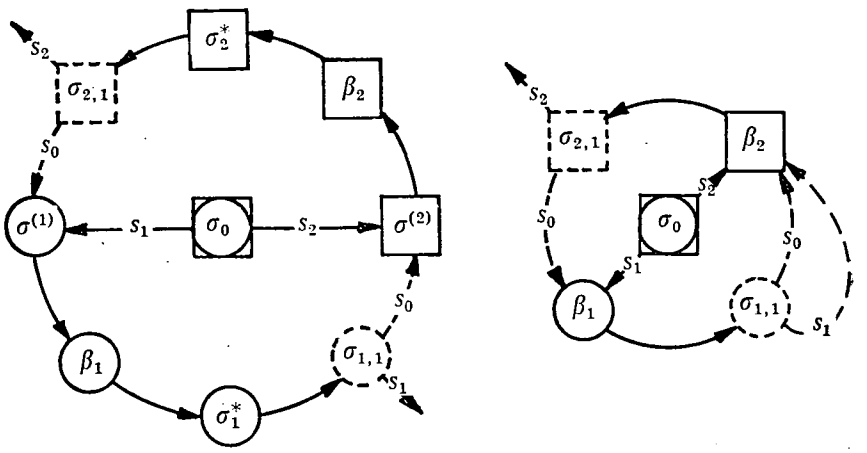
For any given configuration $Q \in \mathcal{Q}$ the elements $R \in \mathcal{R}(Q)$ can similarly be illustrated by a graph $G(Q)$ which is the subgraph of G (see Fig. 6/c). The dotted arcs on Fig. 7/a, b may be present only of a branch of $G(Q)$ is finite or missing. From the arcs going out from $G_0^{(a)}$ at most one can be present in any $G(Q)$. The number of vertices of $G(Q)$ can be infinite. Examples for infinity are the configurations with

$$\eta_a \vartheta_{3-a}=0,\ \vartheta_a \text{ and } \tau_{3-a} \text{ rationally independent} \qquad (5)$$

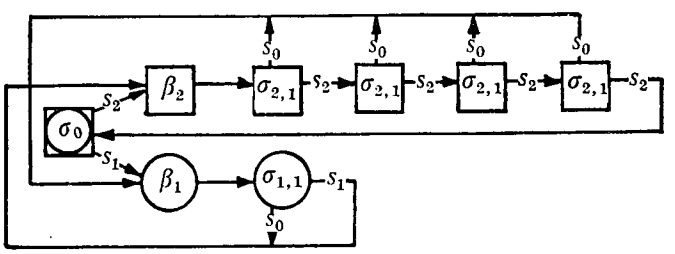
(see Fig. 8/b, c). The general conditions of the infinite vertices of $G(Q)$ is an open question. Perhaps, the above conditions are necessary.



(a) Gantt-chart



(b) Graphs $G_0(Q)$



(c) Graph $G(Q)$

Fig. 6
Graphical illustrations of the ESs for the configuration $Q=(4.5;\ 3.5;\ 1;\ 2)$

For any $Q\in\mathcal{Q}$ every $R\in\mathcal{R}(Q)$ can well be illustrated by a subgraph $G(R)$ of $G(Q)$. The configurations $Q\in\mathcal{Q}$ and the schedules $R\in\mathcal{R}(Q)$ can be classified e.g. by some significant characteristics of their graphs as well. Such characteristics can be the existence and number (one or two) of the branches $G_1^{(a)}(R)$, the finiteness, the number of loops in $G(R)$, etc. We will use some classifications below.

Let $R\in\mathcal{R}(Q)$ be an ES and $G(R)$ the graph representing it. $G(R)$ may have finite or infinite vertices. Let us call the *tour of* R the passage along the arcs and

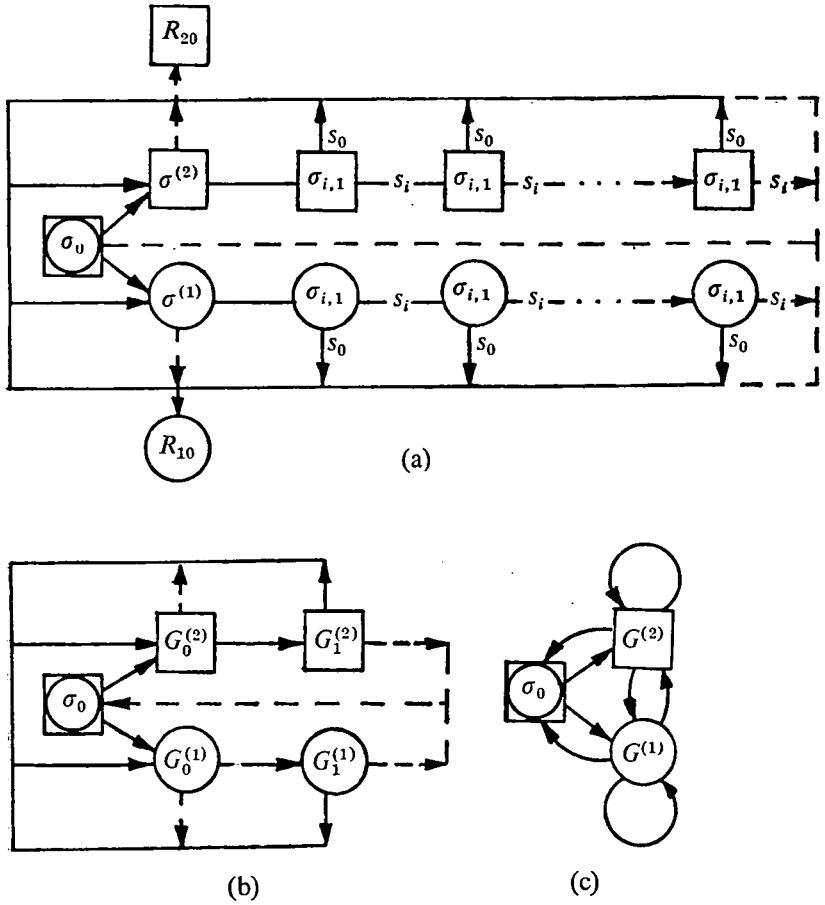


Fig. 7
The graph G of the elements of $\mathscr{R}$ and its partitions

vertices of $G(R)$ in accordance with all the characteristic situations of R. The passage of R may be finite ending in a vertex R_{i0} or infinite with finite or infinite number of loops. A *simple loop* in any graph is a loop having no other loops as its part. For any loop in $G(Q)$ there is at least one path from the vertex σ_0 to the loop without any other loop. The first vertex of the loop reached by the path from σ_0 to the loop is called a *root* of the loop.

For some reasons it may be necessary to allow demands of tasks to be zeros. The job-flow $Q^{(i)}$ is *defective* if one of η_i and ϑ_i is zero and is *degenerate* if both are zeros. For degenerate configurations (for which $\tau_1=0$ or $\tau_2=0$) we can impose specific restrictions to better model practical cases in which demands of one job-flow are negligible with respect to others. In such cases our methods could lead to optimal schedule not reasonable with regard to other optimal schedules. A re-

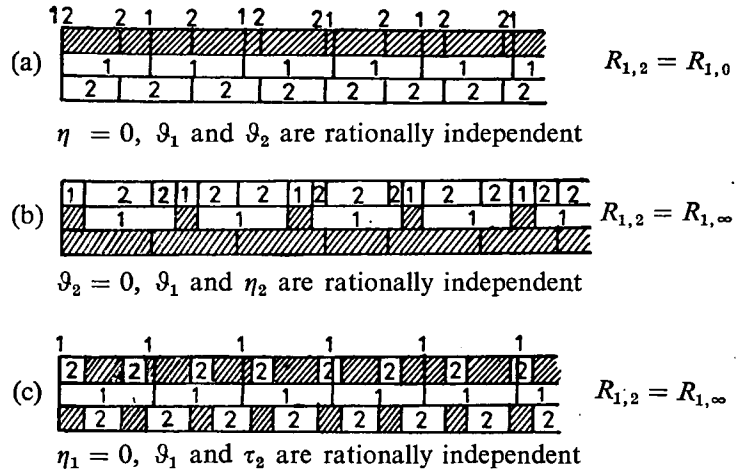


Fig. 8
Examples for CESs not periodic and having infinitely many different critical situation values

striction may be the prohibition of servicing repeatedly the cycles of the same degenerate job-flow alone [2, 3]. Such restrictions further complicate the discussion of the schedules. In degenerate cases the ESs are non-preemptive and are discussed in the course of non-preemptive scheduling of steady job-flow pairs [3].

4. Consistent economical schedules

After the preparations made in the previous paragraph, we are near to be able to prove our most important assertion: the class of consistent economical schedules is a dominant set.

An ES is a *consistent economical schedule* (CES) if its critical decisions are consistent: they are the same in every occurrence of the same critical situation values. Note that two situations of the same type, $\sigma_{i,1}$ say, may well have different values by having different values of $\beta^{(1)}(t)$ or $\beta^{(2)}(t)$, for instance. Let $\bar{\mathcal{R}}(Q) \subset \mathcal{R}(Q)$ be the class of CESs for Q and $\bar{\mathcal{R}} = \bigcup_{Q \in \mathcal{Q}} \bar{\mathcal{R}}(Q)$.

The graphs $G(R')$ of CESs $R' \in \bar{\mathcal{R}}$ have specific characteristics. It can only have one out-arc from any vertex except the vertex R_{i0}, $i=1, 2$, if it is in $G(R')$. R_{i0} has no out-arc. Any vertex has only one in-arc except eventually the vertex σ_0 and one more. σ_0 has no in-arc if R_{i0} is in $G(R')$ or $G(R')$ is infinite. In case of a finite number of vertices and without R_{i0}, $G(R')$ has exactly one simple loop with root σ_0 if σ_0 has an in-arc or with another root which has two in-arcs then. The CES R' is said constructed from this loop. For any simple loop of $G(Q)$ there is at least one $G(R')$ composed from the loop and a path leading from σ_0 to the root of the loop. The tour of R' is the path from σ_0 to the root and infinitely many repetitions of the loop after. The efficiency of the CES so constructed is the P_A-

utilization of the constituent loop. This CES is *periodic* with *periods* represented by the loop. If $G(Q)$ is infinite, let $R_{a,\infty}$ denote the CES with a tour from σ_0 through $\sigma^{(a)}$ and vertices $\sigma_{i,1}$ to the infinity without any loop.

Theorem 3. *The class $\overline{\mathscr{R}}$ of the consistent economical schedules is a dominant set.*

Proof. Let $R \in \mathscr{R}$ be any ES with efficiency $\gamma(R)$. We will show a CES $R' \in \overline{\mathscr{R}}$ dominating R. The dominance follows if R is CES or is essentially-the-same as a CES R'.

If the graph $G(Q)$ does not have loops, all ESs are consistent and R may not be other as well. If the P_A-utilizations of the simple loops of $G(Q)$ have a maximum, the R' constructed from a simple loop with maximal P_A-utilization will dominate every other ESs except eventually those which are essentially-the-same as R_{i0} or $R_{i,\infty}$, $i=1, 2$.

The only crucial $G(Q)$ is that in which the P_A-utilizations of simple loops have no maximum. But if the $G(R) \subset G(Q)$ has a simple loop with P_A-utilization not less than $\gamma(R)$, the CES R' constructed from this loop will dominate R. Thus the dominatedness of R with finite $G(R)$ by CESs is proved. If $G(R)$ is infinite but with a finite number of simple loops, the tour of R cannot have a loop after a finite initial section and is essentially-the-same as an $R_{i,\infty}$.

The only crucial $G(R)$ is, therefore, that which has infinitely many simple loops without one having maximum P_A-utilization. Whether such a $G(R)$ does or does not exist is an open but irrelevant question now. The length of loops cannot be bounded in this case. The schedule R is composed from two kinds of simple loops represented by Fig. 9.

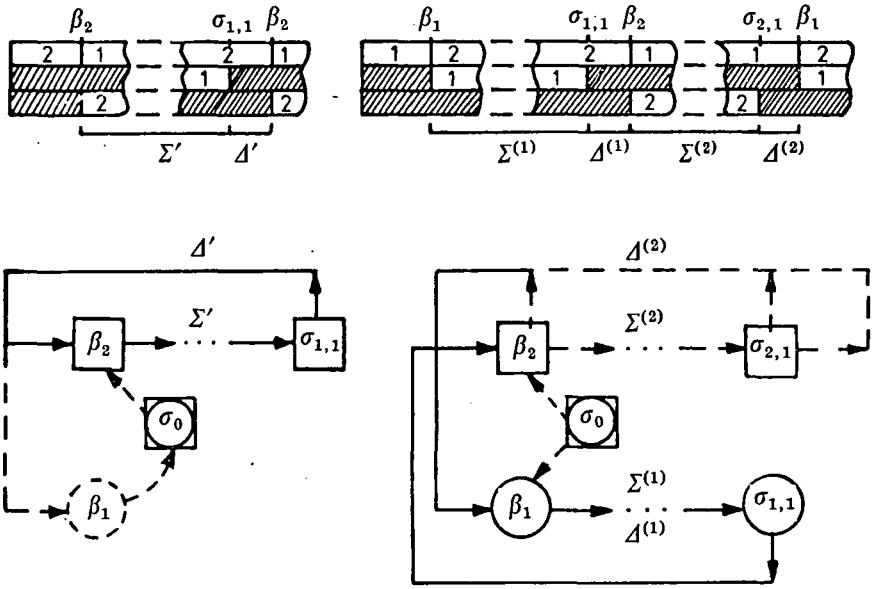


Fig. 9
The two possibilities of simple loops

By definition (1) of $\gamma(R)$ we can choose a sequence $\Sigma_1, \Sigma_2, \ldots, \Sigma_n, \ldots$ of initial sections of R which are ending with simple loops and for which

$$\gamma(R) = \lim_{n\to\infty} \frac{\lambda(\Sigma_n)}{t(\Sigma_n)}$$

where $\lambda(\Sigma_n)$ and $t(\Sigma_n)$ are the P_A-usage and length, respectively, of the section Σ_n. But

$$\gamma(\Sigma_n) = \frac{\lambda(\Sigma_n)}{t(\Sigma_n)}$$

is the weighted mean of the P_A-utilizations of the finite many simple loops composing Σ_n. Let $\Delta\Sigma_1, \Delta\Sigma_2, \ldots$ a sequence of simple loops carved out of $\Sigma_1, \Sigma_2, \ldots$, respectively, with maximal P_A-utilizations. By assumptions

$$\gamma(\Sigma_n) \leq \gamma(\Delta\Sigma_n) < \gamma(R)$$

and so the convergence $\gamma(\Delta\Sigma_n)\to\gamma(R)$ is true. The sequence $\Delta\Sigma_1, \Delta\Sigma_2, \ldots$ must have a subsequence with monotonically increasing length and P_A-utilization because the contrary would lead to contradiction with the assumptions $\gamma(\Delta\Sigma_n)\to\gamma(R)$ and no finite loop with $\gamma(\Delta\Sigma_n)\geqq\gamma(R)$ exists. Let $\Delta\Sigma_1, \Delta\Sigma_2, \ldots$ be this subsequence already. Clearly $\gamma(\Delta\Sigma_n)\to\gamma(R)$. Every $\Delta\Sigma_n$ could be composed either from an initial section Σ'_n of an $R_{i,\infty}$, $i=1, 2$, and a section Δ'_n of bounded length or from an initial section $\Sigma_n^{(1)}$ of $R_{1,\infty}$, an initial section $\Sigma_n^{(2)}$ of $R_{2,\infty}$, a section $\Delta_n^{(1)}$ and a section $\Delta_n^{(2)}$ of bounded lengths, as in Fig. 9. Because of boundedness of sections Δ'_n, $\Delta_n^{(1)}$ and $\Delta_n^{(2)}$ they do not influence the limit of $\gamma(\Delta\Sigma_n)$ and

$$\lim_{n\to\infty} \gamma(\Delta\Sigma_n) = \lim_{n\to\infty} \gamma(\Sigma_n^{(1)} \cup \Sigma_n^{(2)})$$

allowing one of $\Sigma_n^{(1)}$ and $\Sigma_n^{(2)}$ to be missing. In the sequence $\Delta\Sigma_1, \Delta\Sigma_2, \ldots$ at least one of $\Sigma_n^{(1)}$ and $\Sigma_n^{(2)}$ tends to $R_{1,\infty}$ or $R_{2,\infty}$, respectively. $\gamma(\Delta\Sigma_n)$ cannot be greater in limit than the maximum of limits of $\gamma(\Sigma_n^{(1)})$ and $\gamma(\Sigma_n^{(2)})$. Therefore, the maximum of $\gamma(R_{1,\infty})$ and $\gamma(R_{2,\infty})$ will not be less than $\gamma(R)$ and the corresponding CES $R_{i,\infty}$ dominates R. This concludes our proof. □

The set $\bar{\mathscr{R}}(Q)$ of CESs can have fairly many — if not infinite — elements in general. Methods for reducing further the dominant set or a simple algorithm to choose an optimal schedule from $\bar{\mathscr{R}}(Q)$ are not known. A direct method to determine the optimal schedule is to survey the whole set $\bar{\mathscr{R}}$ and compare the efficiencies of the elements. In some cases this is a feasible arrangement. To judge better the amount of work on this way we can use the *number* $N_L(Q)$ *of simple loops* in $G(Q)$ and the *number* $\bar{N}(Q)$ *of elements* of $\bar{\mathscr{R}}(Q)$. To determine these we need the graph $G(Q)$ or at least some data of it.

Let us define the following data (see Fig. 6 and Fig. 7 as illustration):

$$\begin{aligned} &n_0 \text{ is the number of } R_{i0} \text{ vertices in } G(Q) \\ &n_{aj} \text{ is the number of vertices } \sigma_{j,1} \text{ of the branch } G_1^{(a)}(Q) \end{aligned} \tag{6}$$

for $a = 1, 2, \quad j = 1, 2$

$$\delta_{aj} = \begin{cases} 1 & \text{if the last arc of } G^{(a)} \text{ leads to vertex } \sigma^{(j)} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

for $a = 1, 2 \quad j = 0, 1, 2$ and $\sigma^{(0)} = \sigma_0$.

Use the notations

$$n_a = n_{a1} + n_{a2}, \quad a = 1, 2. \tag{8}$$

n_a is the number of vertices in the branch $G_1^{(a)}(Q)$. All the data can be read from two schedule-sections $\Sigma^{(a)}$, $a=1, 2$, constructed in the following way. For $\Sigma^{(a)}$ schedule Q economically with critical decisions $s(0)=s_a$ and $s(\sigma_{i,1})=s_i$, $i=1, 2$, until the first typical situation other than $\sigma_{i,1}$ occurs. This procedure is finite iff $G(Q)$ is finite. From these two schedule-sections we can read the P_A-usages $\lambda(\Delta\Sigma)$ and lengths $t(\Delta\Sigma)$ of determined sections $\Delta\Sigma$ which are necessary to evaluate the CESs of Q. These two schedule-sections enable us to draw simply the graph $G(Q)$ and determine the data (6)—(8). To illustrate this method, Fig. 12 below can be considered. The way to use the data to determine $N_L(Q)$ and $\overline{N}(Q)$ is stated by the following lemma.

Lemma 1. *The number N_L of the simple loops of $G(Q)$ and the number $\overline{N}$ of the elements of $\overline{\mathcal{R}}(Q)$ can be expressed as*

$$N_L = (n_{11}+\delta_{10}+\delta_{12})(n_{22}+\delta_{20}+\delta_{21})+(n_{12}+\delta_{10}+\delta_{11})+(n_{21}+\delta_{20}+\delta_{22})-\delta_{10}\delta_{20} \tag{9}$$

$$\overline{N} = (n_{11}+\delta_{12})(n_2+\delta_{20}+\delta_{21}+\delta_{22})+(n_{22}+\delta_{21})(n_1+\delta_{10}+\delta_{11}+\delta_{12})+$$
$$+(n_{12}+\delta_{10}+\delta_{11})+(n_{21}+\delta_{20}+\delta_{22})+n_0 \tag{10}$$

where n_j, n_{aj} and δ_{aj} are defined by (6)—(8).

Proof. Consider Fig. 7 as illustration. We count the number of simple loops of the graph $G(Q)$ and the number of different paths from σ_0 to the loop without other loops.

The number $N_L^{(aa)}$ of loops not leading out from the subgraph $G^{(a)}$ is the number of vertices $\sigma_{3-a,1}$ plus one if the last arc of $G^{(a)}$ leads to the vertex $\sigma^{(a)}$. This gives $N_L^{(aa)}=n_{a,3-a}+\delta_{aa}$. The root $\sigma^{(a)}$ of these loops can be reached directly from σ_0 or through $\sigma^{(3-a)}$ if arcs connect $G^{(3-a)}$ to $\sigma^{(a)}$. The number of the latter arcs is the number of vertices $\sigma_{a,1}$ in $G^{(3-a)}$ plus one if the last arc of $G^{(3-a)}$ leads to $\sigma^{(a)}$. This gives the number of paths from σ_0 to $\sigma^{(a)}$ as $1+n_{3-a,3-a}+\delta_{3-a,a}$ and the number $\overline{N}^{(aa)}$ of the CESs as $\overline{N}^{(aa)}=(n_{a,3-a}+\delta_{aa})(1+n_{3-a,3-a}+\delta_{3-a,a})$. Further loops arise from arcs leading from $G^{(1)}$ to $\sigma^{(2)}$ and back from $G^{(2)}$ to $\sigma^{(1)}$. The number of arcs leading from $G^{(a)}$ to $\sigma^{(3-a)}$ is the number of vertices $\sigma_{a,1}$ in the branch $G_1^{(a)}$ plus one if the last arc of $G^{(a)}$ leads to $\sigma^{(3-a)}$ as well. This gives the number $N_L^{(0)}$ of simple loops as $N_L^{(0)}=(n_{11}+\delta_{12})(n_{22}+\delta_{21})$. Any of these loops can be reached directly through $\sigma^{(1)}$ or $\sigma^{(2)}$ giving the number of CESs as $\overline{N}^{(0)}=2(n_{11}+\delta_{12})(n_{22}+\delta_{21})$. There are loops between σ_0 and $G^{(a)}$ if the last arc in $G^{(a)}$ leads to σ_0. Because the vertex σ_0 is the component of the loop, one or other of the paths $\sigma_0\to\sigma^{(1)}$ and $\sigma_0\to\sigma^{(2)}$ is an arc of the loop and determine the possible loops. The arc $\sigma_0\to\sigma^{(a)}$ is the part of only one loop if $\delta_{a0}=1$. The arc $\sigma_0\to\sigma^{(3-a)}$ is the part of loops

$\sigma_0 \to \sigma^{(3-a)} \to \sigma_{3-a,3-a} \to \sigma^{(a)} \to \sigma_0$ the number of which is $n_{3-a,3-a}+\delta_{3-a,a}$. These give the number of loops $N_L^{(a)}=(1+n_{3-a,3-a}+\delta_{3-a,a})\delta_{a0}$. Each loop is the constituent of exactly one CES and this fact gives the number $\bar{N}^{(a)}=(1+n_{3-a,3-a}+$ $+\delta_{3-a,a})\delta_{a0}$.

Adding up the numbers for $a=1$ and $a=2$, we have

$$N_L = N_L^{(11)}+N_L^{(22)}+N_L^{(0)}+N_L^{(1)}+N_L^{(2)} =$$

$$= n_{12}+\delta_{11}+n_{21}+\delta_{22}+(n_{11}+\delta_{12})(n_{22}+\delta_{21})+(1+n_{11}+\delta_{12})\delta_{20}+(1+n_{22}+\delta_{21})\delta_{10}$$

and

$$\bar{N}'=\bar{N}^{(11)}+\bar{N}^{(22)}+\bar{N}^{(0)}+\bar{N}^{(1)}+\bar{N}^{(2)}=(n_{12}+\delta_{11})(1+n_{22}+\delta_{21})+$$

$$+(n_{21}+\delta_{22})(1+n_{11}+\delta_{12})+2(n_{11}+\delta_{12})(n_{22}+\delta_{21})+(1+n_{11}+\delta_{12})\delta_{20}+(1+n_{22}+\delta_{21})\delta_{10}.$$

If $G_0^{(a)}(Q)$ contains the vertex R_{a0}, the subgraph $G_0^{(a)}$ in Fig. 7/b has no out-arc and cannot take part in any cycle but represents a CES the path of which ends in vertex R_{a0}. This means that the value $\bar{N}'$ obtained above must be corrected by adding n_0 to the number of CESs generated by loops. The identity of the so obtained expressions of N_L and $\bar{N}'+n_0$ with (9) and (10) is obvious. □

For the *example* of Fig. 6 we get

$$n_{11}=1, \quad n_{12}=0, \quad \delta_{10}=0, \quad \delta_{11}=0, \quad \delta_{12}=1$$
$$n_{21}=0, \quad n_{22}=4, \quad \delta_{20}=1, \quad \delta_{21}=0, \quad \delta_{22}=0.$$

From these data the numbers are

$$N_L = 11 \quad \text{and} \quad \bar{N}=19.$$

If $G(Q)$ has no branches, i.e. $n_{ai}=0$, $a=1, 2$, $i=1, 2$, then the particular formulae are

$$N_L(Q) = (\delta_{10}+\delta_{12})(\delta_{20}+\delta_{21})+(\delta_{10}+\delta_{11})+(\delta_{20}+\delta_{22})-\delta_{10}\delta_{20} \leqq 2 \qquad (9')$$

$$\bar{N}(Q) \equiv 2. \qquad (10')$$

The relations can be proved simply by taking the possible values of n_0 and every δ_{aj}.

The CESs having the same simple loop as their constituent (period) are essentially-the-same. The number of essentially different CESs is N_L and $\bar{\mathcal{R}}(Q)$ represents at most N_L different efficiency values.

Except the trivial cases of existence of a vertex R_{a0} in $G_0(Q)$ — which can only be in the defective cases (3) and (4) — the relations

$$\delta_{a0}+\delta_{a1}+\delta_{a2}=1, \quad a=1, 2, \quad n_0=0 \qquad (11)$$

are always true and the expressions (9) and (10) can be written in the simpler forms

$$N_L = (n_{11}+1-\delta_{11})(n_{22}+1-\delta_{22})+(n_{12}+1-\delta_{12})+(n_{21}+1-\delta_{21})-\delta_{10}\delta_{20} \qquad (9'')$$

$$\bar{N} = (n_{11}+\delta_{12})(n_2+1)+(n_{22}+\delta_{21})(n_1+1)+(n_{12}+1-\delta_{12})+(n_{21}+1-\delta_{21}). \qquad (10'')$$

The expressions (9) and (9″) show how the number N_L of the possible CESs representing different values of efficiency depends on the numbers n_{aj}, $a, j=1, 2,$

of the vertices in the branches of $G(Q)$. N_L is finite if all n_{aj} are finite and if $n_{ii}=\infty$ but $n_{i,3-i}, n_{3-i,i}$ are finite and $n_{3-i,3-i}+\delta_{3-i,0}+\delta_{3-i,i}=0$ (provided that this last case is possible for some configuration Q).

For the sake of reference, we have to identify the elements of $\bar{\mathcal{R}}(Q)$. In view of evaluation, the identification of the simple loops is enough. We introduce a symbolism for this purpose.

We identify the vertices of the branches $G_1^{(a)}$, $a=1, 2$, by numbering them serially with 1, 2, ..., n_a in the order of occurrences in $G_1^{(a)}$. Let the vertex $\sigma^{(a)}$ have the serial number 0 and the vertex of $G(Q)$ the last arc of $G^{(a)}(Q)$ leads to the serial number n_a+1. This last vertex can be either σ_0 or $\sigma^{(1)}$ or $\sigma^{(2)}$. The serial numbers of vertices of $G^{(1)}$ and $G^{(2)}$ of our example in Fig. 6 will be 0, 1, 2 and 0, 1, 2, 3, 4, 5, respectively. The last number of $G^{(1)}$ represents the vertex $\sigma^{(2)}$ and the last number of $G^{(2)}$ represents the vertex σ_0. Every simple loop is composed from one or two sections belonging to subgraphs $G^{(1)}$ and $G^{(2)}$, respectively. Every *loop-section* of $G^{(a)}$ starts with the vertex $\sigma^{(a)}$, goes through some further vertices of $G_1^{(a)}$ if they exist, and finishes in σ_0, $\sigma^{(1)}$ or $\sigma^{(2)}$. A loop-section of a given $G^{(a)}(Q)$ can be identified by the maximum of serial numbers of its vertices. The character of a loop-section can well be given by a code (abc) constructed from the number "a" of the subgraph it belongs to, from the maximal serial number "b" of its vertices and from the code "c" of its last vertex by the coding:

type	σ_0	$\sigma^{(1)}$	$\sigma^{(2)}$
c-code	0	1	2

The code (ac) identifies the shape of the loop-section which can be symbolized in the following way:

a \ c	0	1	2	$\sigma_0 \to \sigma^{(a)}$
1	⊐	⊟	⊏	╲
2	⊐	⊏	⊏	╱

The simple loops are composed from one or two sections directly or by means of a section $\sigma_0 \to \sigma^{(1)}$ or $\sigma_0 \to \sigma^{(2)}$ symbolized by ╲ and ╱.

To identify a simple loop we can use the b-codes of its component loop-sections. The loop identified with $(b_1 b_2)$ has vertices from $G^{(1)}$ and $G^{(2)}$ with maximum serial number b_1 and b_2, respectively. If a loop has no vertex from $G^{(a)}$, the component b_a is zero.

The elements R of $\bar{\mathcal{R}}$ can be characterized by the code $(b_1 b_2)$ of its simple loop. The CESs R_{a0} for degenerate configurations (3) and (4) will be characterized by the code (00). The code $(b_1 b_2)$ of a CES is called its *type*. The code $(b_1 b_2)$ represents an essentially-the-same class of $\bar{\mathcal{R}}(Q)$, the number of which was counted in the proof of Lemma 1.

Not every code $(b_1 b_2)$ can represent an existing loop in $G(Q)$. In Table 1 we marked by sign + or − that a loop of code $(b_1 b_2)$ composed from the existing

loop-section pair $(1b_1c_1)$, $(2b_2c_2)$ did or did not exist, respectively. The code (00) is possible if at least one vertex R_{a0} of $G(Q)$ exists (and $n_a=0$, of course). In this case the only possible value of b_a is 0. The other (b_1b_2) entries of Table 1 for given (abc) codes can be easily made. We put sign $-$ in every entries of rows with $c_1=1$ and of columns with $c_2=2$ except their first entries. In row $b_1=0$ we put $-$ in entries with heading $c_2=1$ and in column $b_2=0$ we put $-$ in entries with heading $c_1=2$. If an entry with $c_1=c_2=0$ existed, we put $-$ in it. In the remaining entries we put signs $+$.

Table 1. The existing codes (b_1b_2) of simple loops.

b_1	c_1 \ c_2	(R_{20})	b_2: 2	1	0	2	1
0	(R_{10})	(+)	+	−	+	+	−
b_1	1	+	−	−	−	−	−
	2	−	−	+	+	−	+
n_1+1	0	+	−	+	−	−	+
	1	+	−	−	−	−	−
	2	−	−	+	+	−	+

Example of Fig. 6

b_1	c_1 \ b_2	0	1	2	3	4	5
	c_2	−	1	1	1	1	0
0	−	−	−	−	−	−	⊕
1	2	−	⊕	+	+	+	+
2	2	−	⊕	+	+	+	+

Table 1 says which loops have to be evaluated for determining the optimal one. The possibilities for some specific types of CESs are represented by Fig. 10/a.

The set $\bar{\mathcal{R}}(Q)$ always contains exactly two non-preemptive schedules $R_{a,0}$, $a=1, 2$, which are the two tight consistent natural schedules defined in [4]. These are the *non-preemptive priority schedules,* at the same time [2]. Two other remarkable elements of $\bar{\mathcal{R}}(Q)$ are the *priority schedules* $R_{a,3-a}$, $a=1, 2$. $R_{a,3-a}$ is defined as the CES in which the job-flow $Q^{(a)}$ has absolute priority against $Q^{(3-a)}$ which means that every task A_{aj}, $j=1, 2, \ldots$, is serviced by P_A at the moment it is ready for service, independently of the state of P_A. The priority schedules are schedules of great practical importance. With the help of Table 1 it is easy to determine the types $(b'_1b'_2)$ of the priority schedules $R_{a,0}$ and $R_{a,3-a}$, $a=1, 2$, by their definitions.

$R_{a,0}$ is determined by the restriction that no preemption is allowed and $s(0)=s_a$. This means that $R_{a,0}=R_{a0}$ and has type (00) if the vertex R_{a0} exists. Otherwise, $b'_a=1$, $b'_{3-a}=0$ except if $c_{a,1}=3-a$ when $b'_{3-a}=1$, and $c_{3-a,1}=c_{a,1}=3-a$ when $b'_a=0$, moreover.

$R_{a,3-a}$ is determined by the fact that any task type A_{3-a} must and any task type A_a must not be preempted in conflicting situations $\sigma_{i,1}$, $i=1, 2$. This means that $s(\sigma_{a,1})=s_a$ and $s(\sigma_{3-a,1})=s_0$. The possibilities are illustrated by Fig. 10/b. If the vertex R_{a0} exists, then $R_{a,3-a}=R_{a,0}=R_{a0}$ with type (00). Otherwise, b'_a of

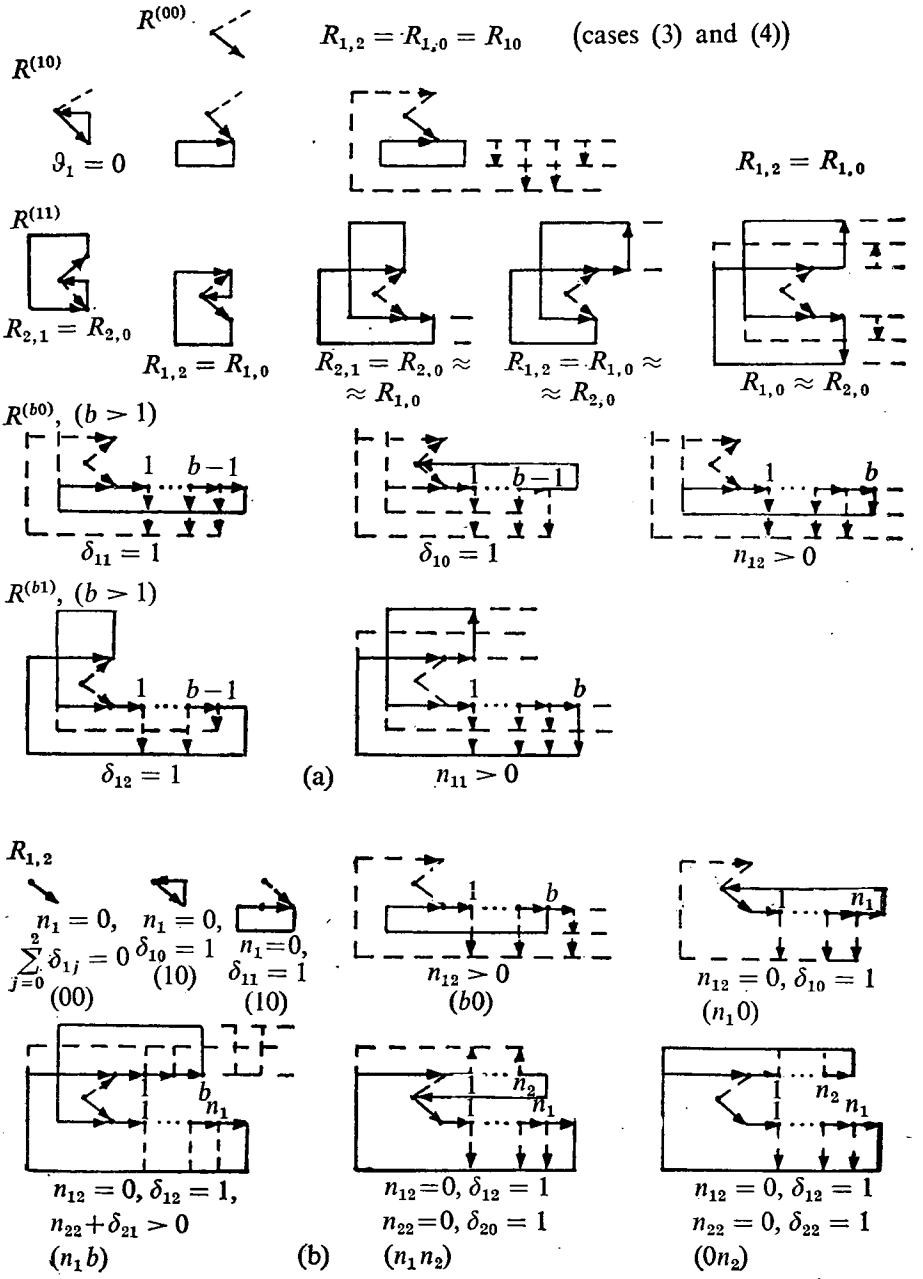


Fig. 10
Special types $R^{(b_1 b_2)}$ and types of $R_{1,2}$

$R_{a,3-a}$ is the serial number of the first vertex type $\sigma_{3-a,1}$ in the branch $G_1^{(a)}$, if it exists ($n_{a,3-a}>0$) and $b'_a=n_a+1$ or $b'_a=0$ otherwise (when $n_{a,3-a}=0$). $b'_a=n_a+1$ if $\delta_{a,3-a}=0$ or $\delta_{a,3-a}=1$ and $n_{3-a,3-a}+1-\delta_{3-a,3-a}>0$. $b'_a=0$ if $\delta_{a,3-a}=1$ and $n_{3-a,3-a}+1-\delta_{3-a,3-a}=0$. The value of b'_{3-a} of $R_{a,3-a}$ is 0 when $n_{a,3-a}+1-\delta_{a,3-a}>0$, the serial number of the first vertex $\delta_{3-a,1}$ in the branch $G_1^{(3-a)}$, if it exists ($n_{3-a,3-a}>0$) and $n_{3-a}+1$, otherwise, when $n_{a,3-a}+1-\delta_{a,3-a}=0$.

In the completed Table 1 we can pick out the types of $R_{a,0}$ and $R_{a,3-a}$ as follows. $R_{a,0}$ is represented by the sign + encounters first in counter-clockwise for $a=1$ and clockwise for $a=2$ in the left upper 2×2 subtable and $R_{a,3-a}$ is represented by the first + encounters on the border of the whole table counter-clockwise for $a=1$ and clockwise for $a=2$ starting from the entry (00). If Table 1 consists only from one row then $R_{1,0}=R_{1,2}=R_{10}$ and if it consists only from one column then $R_{2,0}=R_{2,1}=R_{20}$.

Let $\mathscr{R}_0(Q)=\{R_{1,2}, R_{2,1}\}$ be the pair of priority schedules. This is a subset of $\bar{\mathscr{R}}(Q)$. If $\bar{\mathscr{R}}(Q)=\mathscr{R}_0(Q)$ then $\mathscr{R}_0(Q)$ is a dominant set. In this case $R_{a,3-a}=R_{a,0}$, $a=1, 2$. An example for this is the configuration $Q=(1; 4; 2; 5)$ with $R_{1,2}(Q)$ optimal. If $\bar{\mathscr{R}}(Q)\neq\mathscr{R}_0(Q)$, the set $\mathscr{R}_0(Q)$ is not necessarily dominant. Trivial examples for this are the configurations Q with $\vartheta_i<\eta_{3-i}<2\vartheta_i$, $i=1, 2$. For these configurations the CESs $R_{1,0}\approx R_{2,0}$ are optimal with efficiency $\gamma=1$. A non-trivial example is the configuration $Q=(4.5; 3.5; 1; 2)$ in Fig. 6 as we will see in the next paragraph.

Though the priority schedules are not dominant, they are interesting on their own, because they are often used in practice and can be produced by simple rules. They are investigated in the study [2]. The evaluation of $R_{1,2}$ and $R_{2,1}$ is not a trivial task at all. The priority schedules were investigated also for the stochastic version of job-flow pairs [1, 5].

5. Evaluation of the CESs

Though the cardinal of the dominant set $\bar{\mathscr{R}}(Q)$ of the consistent economical schedules is not necessarily finite, we give an algorithm for the direct evaluation of the CESs. This is applicable only when $\bar{\mathscr{R}}(Q)$ is finite. $\bar{\mathscr{R}}(Q)$ is finite exactly then when the graph $G(Q)$ is finite. For some cases the automatic application of the given algorithm can be superfluously complicate. Four such cases will be mentioned below as cases (i)—(iv). These cases contain the configurations we know as having $G(Q)$ with infinite vertices. By *general case* non-defective configurations are meant. The special cases (i)—(iv) are illustrated by Fig. 11.

Case (i). $\tau_1\tau_2=0$, degenerate configurations (see (4)). The CESs are the $R_{a,0}$, $a=1, 2$, and $\gamma_{a,0}=0$. If the number of cycles of the same degenerate job-flow scheduled directly after each other is restricted, the maximal efficiency $\gamma^{(1)}+\gamma^{(2)}$ can be achieved.

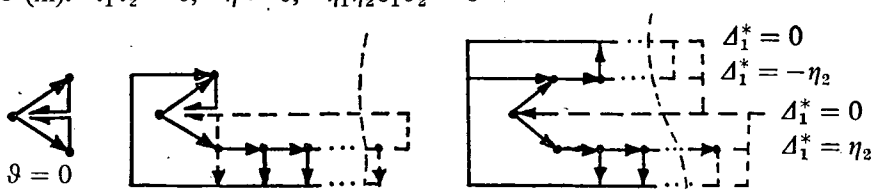
Case (ii). $\eta=0$, $\vartheta_1\vartheta_2>0$. $R_{a,0}$, $a=1, 2$, are the only CESs with $\gamma=0$. $R_{a,0}=R_{a0}$ and has no typical situations for the configurations (3).

Case (iii). $\tau_1\tau_2>0$, $\eta>0$ but Q is defective. If $\eta_a\vartheta_{3-a}=0$ then $R_{a,3-a}$ has the maximum efficiency of $\gamma=\gamma^{(3-a)}$ (see Fig. 8). The shape of the graph $G(Q)$ depends

Case (i). $\tau_1\tau_2 = 0$

Case (ii). $\eta = 0, \quad \vartheta_1\vartheta_2 > 0$

Case (iii). $\tau_1\tau_2 > 0, \quad \eta > 0, \quad \eta_1\eta_2\vartheta_1\vartheta_2 = 0$



Case (iv). $\eta_1 > \vartheta_2 > 0, \quad \eta_2 > \vartheta_1 > 0$

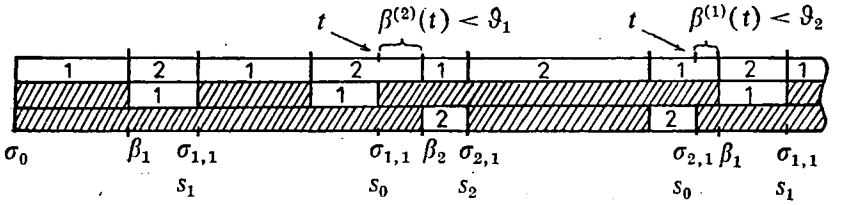


Fig. 11
Trivial cases for optimal schedule

on the existence and relations of the least non-trivial non-negative integer solutions (X_1^*, X_2^*) of the equations

$$\Delta_a \equiv X_a\vartheta_a - X_{3-a}\tau_{3-a} = \begin{cases} 0 \\ \pm\eta_{3-a} \end{cases} \quad a = 1, 2 \qquad (*)$$

but this fact is irrelevant from the point of view of optimality. There is no solution of (*) in cases (5).

Case (iv). $\eta_i \geqq \vartheta_{3-a} > 0$, $i = 1, 2$. The maximal efficiency of the CESs is $\gamma = 1$ and any $R \in \bar{\mathcal{R}}(Q)$ with decisions $s(\sigma_{i,1}) = s_0$ if only $\beta^{(3-i)}(t) - \vartheta_{3-i} < \vartheta_i$, is optimal. E.g. also the $R_{a,0}$, $a = 1, 2$, are optimal with $\gamma_{a,0} = 1$.

Before we give an algorithm for the general case, we show the evaluation of the CESs of the example configuration $Q = (4.5;\ 3.5;\ 1;\ 2)$.

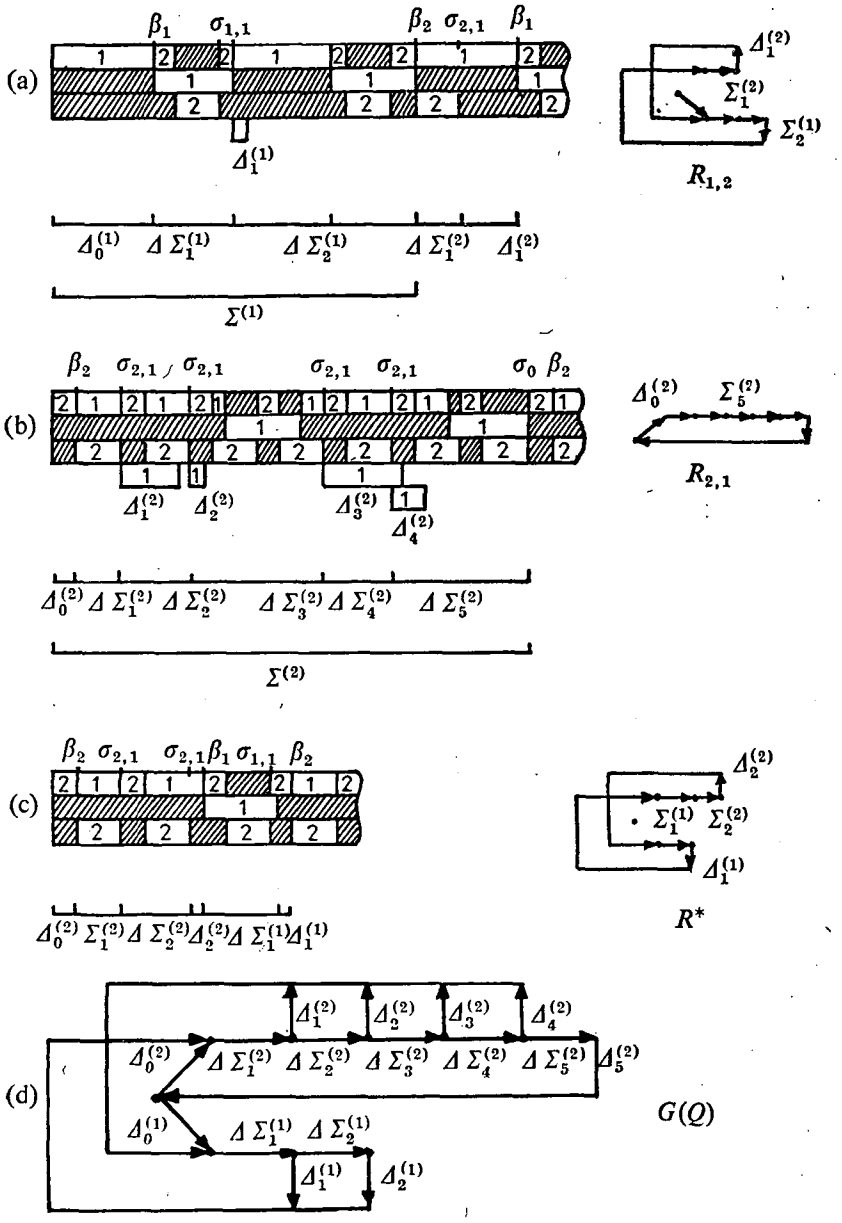


Fig. 12
The sections $\Sigma^{(a)}$, $a=1,2$, the priority and the optimal schedules of the example $Q=(4.5;\ 3.5;\ 1;\ 2)$

In Fig. 12/a, b we show the Gantt-charts of the two schedule sections $\Sigma^{(1)}$ and $\Sigma^{(2)}$ expanded here to provide $R_{1,2}$ and $R_{2,1}$ at the same time. It can be realized that every loop-section is composed from consecutive subsections $\Delta\Sigma_j^{(a)}$, $j=1, \ldots, b$, of $\Sigma^{(a)}$ and a section $\Delta_b^{(a)}$ of full P_A-utilization as $\Sigma_b^{(a)} \cup \Delta_b^{(a)}$. This fact is illustrated by Fig. 12/d. The lengths and P_A-usages of the subsections can be read from $\Sigma^{(1)}$ and $\Sigma^{(2)}$ and are given in Table 2. The data (lengths and P_A-usages) of loop-sections are

$$t(\Sigma_b^{(a)})+\Delta_b^{(a)} \quad \text{and} \quad \lambda(\Sigma_b^{(a)})+\Delta_b^{(a)}$$

with

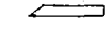
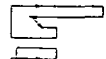
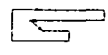
$$\Sigma_b^{(a)} = \bigcup_{j=1}^{b} \Delta\Sigma_j^{(a)}, \quad b = 1, 2, \ldots, n_a+1, \quad a = 1, 2.$$

These data are given in Table 2 as well.

Table 2. The data of loop-sections of the example of Fig. 12

a	b	Type of sect.	c	$\lambda(\Delta\Sigma)$	$t(\Delta\Sigma)$	Δ	$\lambda(\Sigma+\Delta)$	$t(\Sigma+\Delta)$
1	0	$\sigma^{(1)}$	–	–	–	4.5	–	–
	1	$\sigma_{1,1}$	2	1.5	3.5	0.5	2	4
	2	$\sigma^{(2)}$	2	6	8	0	7.6	11.5
2	0	$\sigma^{(2)}$	–	–	–	1	–	–
	1	$\sigma_{2,1}$	1	2	2	2.5	4.5	4.5
	2	$\sigma_{2,1}$	1	3	3	0.5	5.5	5.5
	3	$\sigma_{2,1}$	1	3.5	6	3.5	12	14.5
	4	$\sigma_{2,1}$	1	3	3	1.5	13	15.5
	5	σ_0	0	3.5	6	0	15	20

Table 3. The simple loops and their characteristics for the example of Fig. 12

No.	$(b_1 b_2)$	$G(R)$	Composition	$\lambda(\Sigma)$	$t(\Sigma)$	$\gamma(\Sigma)$	Rmk.
1	(0, 5)		$\Sigma_5^{(2)} \cup \Delta_0^{(2)}$	16	21	0.762	$R_{2,1}$
2	(1, 1)		$\Sigma_1^{(1)} \cup \Delta_1^{(1)} \cup \Sigma_1^{(2)} \cup \Delta_1^{(2)}$	6.5	8.5	0.765	$R_{1,0} \approx R_{2,0}$
3	(1, 2)		$\Sigma_1^{(1)} \cup \Delta_1^{(1)} \cup \Sigma_2^{(2)} \cup \Delta_2^{(2)}$	7.5	9.5	0.789	R^*
4	(1, 3)		$\Sigma_1^{(1)} \cup \Delta_1^{(1)} \cup \Sigma_3^{(2)} \cup \Delta_3^{(2)}$	14	18.5	0.757	
5	(1, 4)		$\Sigma_1^{(1)} \cup \Delta_1^{(1)} \cup \Sigma_4^{(2)} \cup \Delta_4^{(2)}$	15	19.5	0.769	
6	(1, 5)		$\Sigma_1^{(1)} \cup \Delta_1^{(1)} \cup \Sigma_5^{(2)} \cup \Delta_0^{(1)}$	21.5	28.5	0.754	
7	(2, 1)		$\Sigma_2^{(1)} \cup \Sigma_1^{(2)} \cup \Delta_1^{(2)}$	12	16	0.750	$R_{1,2}$
8	(2, 2)		$\Sigma_2^{(1)} \cup \Sigma_2^{(2)} \cup \Delta_2^{(2)}$	13	17	0.765	
9	(2, 3)		$\Sigma_2^{(1)} \cup \Sigma_3^{(2)} \cup \Delta_3^{(2)}$	19.5	26	0.750	
10	(2, 4)		$\Sigma_2^{(1)} \cup \Sigma_4^{(2)} \cup \Delta_4^{(2)}$	20.5	27	0.759	
11	(2, 5)		$\Sigma_2^{(1)} \cup \Sigma_5^{(2)} \cup \Delta_0^{(1)}$	27	36	0.750	

The c-codes of the loop-sections are easy to determine from the fact that the result of the decision $s(\sigma_{i,1})=s_0$ is $\sigma^{(3-i)}$, $i=1, 2$, and the vertex the last arc of $G^{(a)}$ leads to can be obtained as the last typical situation of $\Sigma^{(a)}$ with $\beta_i=\sigma^{(i)}$, $i=1, 2$.

From the possible (abc) codes Table 1 of the possible types (b_1b_2) of the simple loops can be completed. The data of the simple loops can be obtained from those loop-sections which are shown in Table 3. The last datum is $\gamma(\Sigma)$, the efficiency of the corresponding simple loop. Comparing these data we can choose the maximum value as 0.789. The type of the optimal schedule R^* is (1, 2) and its Gantt-chart can be seen in Fig. 12/c.

The table

R	(b_1b_2)	R	$100\gamma/\gamma^*$
R^*	(1, 2)	0.789	100
$R_{1,0}$	(1, 1)	0.765	96.9
$R_{2,0}$	(1, 1)	0.765	96.9
$R_{1,2}$	(2, 1)	0.750	95.0
$R_{2,1}$	(0, 5)	0.762	96.5

shows that the priority schedules are not optimal. The efficiency γ^* of the optimal schedule is 88% of the sum $\gamma^{(1)}+\gamma^{(2)}=4.5/8+1/3=0.896$ and the efficiency of every priority schedule is less than γ^*. $\gamma_{1,2}$ is the minimum of the efficiency values of the CESs. This is 95% of the value γ^*. To find a good estimation for the $\min_{R\in\bar{\mathcal{R}}(Q)} \gamma(R)/\gamma^*$ is an open question. A trivial estimation is clearly $\max_{i=1,2} \gamma^{(i)}/(\gamma^{(1)}+\gamma^{(2)})$. In the example $\gamma_{2,1}$ is not minimal but there are 8 other CESs with greater efficiency. Also $R_{1,0}\approx R_{2,0}$ have better efficiency.

Fig. 12/c shows that the economic decisions in the optimal schedule are chosen such that the delay d caused by the decision be minimal. This heuristic scheduling strategy can often give a not bad schedule but not optimal in general. One can argue that a unit delay of the job-flow with a higher P_A-utilization $\gamma^{(i)}=\eta_i/\tau_i$ is worse than a unit delay of the other job-flow. Therefore, we can expect better schedule by the strategy which decides such that the *loss of utilization* $D_i=\gamma^{(i)}d_i$ by the delay d_i of $Q^{(i)}$ be minimum. For our example the critical situations of R^*, the delays d_i, the losses D_i and the decisions s^* are from the Fig. 12/c as follows:

σ'	d_1	D_1	d_2	D_2	s^*
σ_0	1	0.56	4.5	1.50	s_2
$\sigma_{2,1}$	1	0.56	2.5	0.83	s_2
$\sigma_{2,1}$	1	0.56	0.5	0.17	s_0
$\sigma_{1,1}$	0.5	0.28	4.5	1.50	s_0

The table shows that the optimal decisions correspond to the *strategy of minimizing local losses* of utilization. This strategy is not optimal in every cases either. We show this by the example configuration $Q=(1; 3.5; 2; 1.5)$ in Fig. 13. The graph

$G(Q)$ with the data $\lambda(\Delta\Sigma)$, $t(\Delta\Sigma)$ and Δ is the part (d). The data (6)—(10) are

$$n_0 = 0, \quad n_{11} = 0, \quad n_{12} = 0, \quad \delta_{10} = 1, \quad \delta_{11} = 0, \quad \delta_{12} = 0, \quad n_1 = 0,$$
$$n_{21} = 0, \quad n_{22} = 1, \quad \delta_{20} = 0, \quad \delta_{21} = 1, \quad \delta_{22} = 0, \quad n_2 = 1,$$
$$N_L = 3, \quad \bar{N} = 3.$$

The possible three CESs are $R_{1,2}=R_{1,0}$, $R_{2,1}$ and $R_{2,0}$ by Fig. 13/a, b, c. The efficiency values are $\gamma_{1,2}=\gamma_{1,0}=0.667$, $\gamma_{2,1}=0.743$, $\gamma_{2,0}=0.727$. $R^*=R_{2,1}$ is the

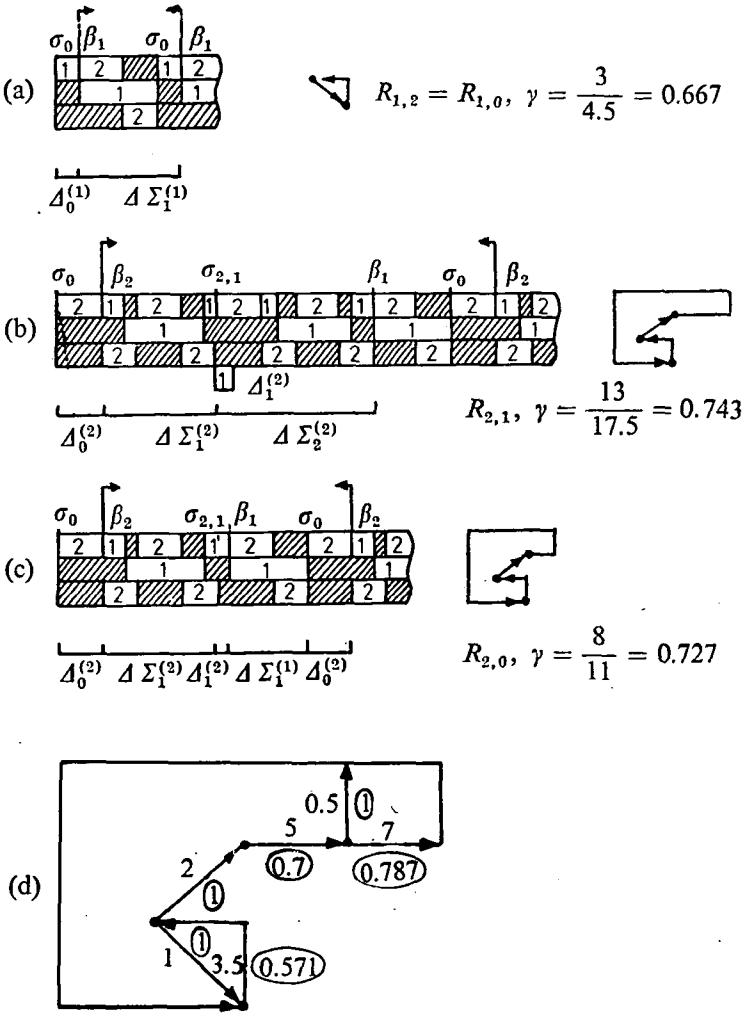


Fig. 13
Example configuration for no optimal minimum local losses strategy $Q=(1;\ 3.5;\ 2;\ 1.5)$

optimal schedule. The delays, losses and optimal decisions are the following ($\gamma^{(1)}=0.222$, $\gamma^{(2)}=0.571$):

σ'	d_1	D_1	d_2	D_2	s^*
σ_0	2	0.444	1	0.571	s_2
$\sigma_{2,1}$	2	0.444	0.5	0.285	s_2

The preemption $s(\sigma_{2,1})=s_2$ causes a greater delay (2) and local loss (0.444) than the decision $s(\sigma_{2,1})=s_0$ would but it is, nevertheless, optimal. The decision $s(\sigma_{2,1})=s_0$ results in $R_{2,0}$ which is not an optimal schedule (see Fig. 13/c). This example shows that the "locally optimal" decisions are not "totally optimal". An evident problem is the ratio γ/γ^* of the efficiency of the schedule with minimal local losses and the efficiency of the optimal schedule.

After the examples we give, now, an algorithm to determine an optimal schedule by direct evaluation and comparison of the CESs in finite cases. Formally we divide the algorithm into two parts and formulate the parts as the *S*-algorithm and the *E*-algorithm.

The *S*-algorithm produce the series of vectors

$$Z_{ab} = (\lambda_{ab}, t_{ab}, c_{ab}), \quad b = 1, 2, \ldots, n_a+1, \quad a = 1, 2$$

with components

$$\lambda_{ab} = \lambda(\Sigma_b^{(a)})+\Delta_b^{(a)}, \quad t_{ab} = t(\Sigma_b^{(a)})+\Delta_b^{(a)}, \quad c_{ab}$$

as P_A-usage, length and c-code of the loop-section with code (ab). An auxiliary variable is in the algorithm $X=(\lambda, t, \Delta)$ as P_A-usage, length of subsections of $\Sigma^{(a)}$ and the length of a next section which will be inspected afterwards. Another auxiliary variable is $\overline{Y}=(\bar{\lambda}, \bar{t})$ the cumulated P_A-usages and lengths of the subsections. The algorithm supplies also the data n_{aj}, δ_{aj} defined by (6)—(7) and used in (9)—(10).

S-algorithm. *Input data:* $Q=(\eta_1; \vartheta_1; \eta_2; \vartheta_2)$;

Output data: n_a, n_{aj}, $j=1, 2$, δ_{aj}, $j=0, 1, 2$, $a=1, 2$, $Z_{ab}=(\lambda_{ab}, t_{ab}, c_{ab})$, $b=1, \ldots, n_a+1$, $a=1, 2$;

Step 0: $\tau_1:=\eta_1+\vartheta_1$; $\tau_2:=\eta_2+\vartheta_2$; $a:=1$; $n:=1$; $i:=2$;
Step 1: $X:=(0, 0, \vartheta_a)$; $\overline{Y}:=(0, 0)$;
Step 2: $l:=[\Delta/\tau_i]$; $\Delta':=\Delta-l\tau_i$;
Step 3: If $\Delta'>\eta_i$ then $X:=(\lambda+(l+1)\eta_i, t+\Delta, \tau_i-\Delta')$, $i:=3-i$ and go to *Step 2*;
If $\Delta'=\eta_i$ then $\overline{Y}:=(\bar{\lambda}+\lambda+(l+1)\eta_i, \bar{t}+t+\Delta)$, $Z_{an}:=(\bar{\lambda}, \bar{t}, i)$, $\delta_{ai}:=1$ and go to *Step 4*;
If $\Delta'=0$ then $\overline{Y}:=(\bar{\lambda}+\lambda+l\eta_i, \bar{t}+t+\Delta)$, $Z_{an}:=(\bar{\lambda}, \bar{t}, 0)$, $\delta_{a0}:=1$ and go to *Step 4*;
$\overline{Y}:=(\bar{\lambda}+\lambda+l\eta_i+\Delta', \bar{t}+t+\Delta)$; $\Delta:=\eta_i-\Delta'$; $Z_{an}:=(\bar{\lambda}+\Delta, \bar{t}+\Delta, i)$;
$n_{a,3-a}:=n_{a,3-a}+1$; $i:=3-i$; $k:=[\Delta/\vartheta_i]$; $\Delta':=\Delta-k\vartheta_i$;
If $k>0$ then $\overline{Y}:=(\bar{\lambda}+\tau_i, \bar{t}+\tau_i)$, $Z_{an+j}:=(\bar{\lambda}+\Delta-j\vartheta_i, \bar{t}+\Delta-j\vartheta_i, 3-i)$, $j=1, \ldots, k$ and $n_{ai}:=n_{ai}+k$;
$n:=n+k$;

If $\Delta'=0$ then $\bar{Y}:=(\bar{\lambda}+\tau_i, \bar{t}+\tau_i)$; $Z_{an}:=(\bar{\lambda}, \bar{t}, 3-i)$; $n_{ai}:=n_{ai}-1$;
$n:=n-1$; $\delta_{a,3-i}:=1$ and go to *Step 4;*
$n:=n+1$;
If $\vartheta_{3-i}+\Delta'-\vartheta_i \geqq 0$ then $X:=(\eta_i+\Delta', \tau_i, \vartheta_{3-i}+\Delta'-\vartheta_i)$ and go to *Step 2;*
$X:=(\eta_i+\Delta', \eta_i+\Delta'+\vartheta_{3-i}, \vartheta_i-\vartheta_{3-i}-\Delta')$; $i:=3-i$; go to *Step 2;*

Step 4: If $a=2$ then $n_2:=n$ and go to *End;*
$n_1:=n$; $n:=1$; $a:=2$; $i:=1$; go to *Step 1;*

End.

The output data of the *S*-algorithm corresponds to the data of Table 2 and the data (6)—(7). From these data the efficiency values of the possible simple loops can be determined by the *E*-algorithm. The flow-chart of the *S*-algorithm is shown in Fig. 14.

The *E*-algorithm uses the output data n_a, $a=1, 2,$ and Z_{ab_a}, $b_a=1, \ldots, n_a+1$, $a=1, 2,$ of the *S*-algorithm and determines the efficiency values γ of the simple loops and provides the type $(b_1^* b_2^*)$ and efficiency γ^* of a simple loop with maximum efficiency. The order of evaluation of the simple loops will determine which of the possibly more than one simple loops with maximum efficiency will be chosen. This order can be seen in Table 1: the + entries of the first column with increasing b_1, the + entries of the first row with increasing b_2 and the other + entries by rows after.

E-algorithm. *Input data:* $\eta_1, \eta_2, n_1, n_2, Z_{ab}=(\lambda_{ab}, t_{ab}, c_{ab})$, $b=1, 2, \ldots, n_a+1$, $a=1, 2;$

Output data: $b_1^*, b_2^*, \gamma^*;$

Definition of operation F: If $\gamma>\gamma^*$ then $b_1^*:=b_1$, $b_2^*:=b_2$ and $\gamma^*:=\gamma;$

Begin: $b_1^*:=b_2^*:=\gamma^*:=b_2:=0;$

For $b_1:=1$ step 1 until n_n+1 do if $c_{1b_1}=1$ then $\gamma:=\lambda_{1b_1}/t_{1b_1}$ and F;
If $c_{1b_1}=0$ then $\gamma:=(\lambda_{1b_1}+\eta_1)/(t_{1b_1}+\eta_1)$ and F; $b_1:=0$;
For $b_2:=1$ step 1 until n_2+1 do if $c_{2b_2}=2$ then $\gamma:=\lambda_{2b_2}/t_{2b_2}$ and F;
If $c_{2b_2}=0$ then $\gamma:=(\lambda_{2b_2}+\eta_2)/(t_{2b_2}+\eta_2)$ and F;
For $b_1:=1$ step 1 until n_1+1 do if $c_{1b_1}=2$ then
begin For $b_2:=1$ step 1 until n_2+1 do
if $c_{2b_2}=1$ then $\gamma:=(\lambda_{1b_1}+\lambda_{2b_2})/(t_{1b_1}+t_{2b_2})$ and F;
If $c_{2b_2}=0$ then $\gamma:=(\lambda_{1b_1}+\lambda_{2b_2}+\eta_1)/(t_{1b_1}+t_{2b_2}+\eta_1)$ and F;
end;
If $c_{1b_1}=0$ then for $b_2:=1$ step 1 until n_2+1 do
if $c_{2b_2}=1$ then $\gamma:=(\lambda_{1b_1}+\lambda_{2b_2}+\eta_2)/(t_{1b_1}+t_{2b_2}+\eta_2)$ and F;

End.

Fig. 15 shows the flow-chart of the *E*-algorithm. This clarifies the meaning of the "for-step-until-do" cycles used in the algoritm.

The verification of the *S*-algorithm is easy e.g. by following its operations graphically on the Gantt-charts of some configurations as of $Q=(4.5; 3.5; 1; 2)$ in Fig. 12. The *E*-algorithm does not need further verification.

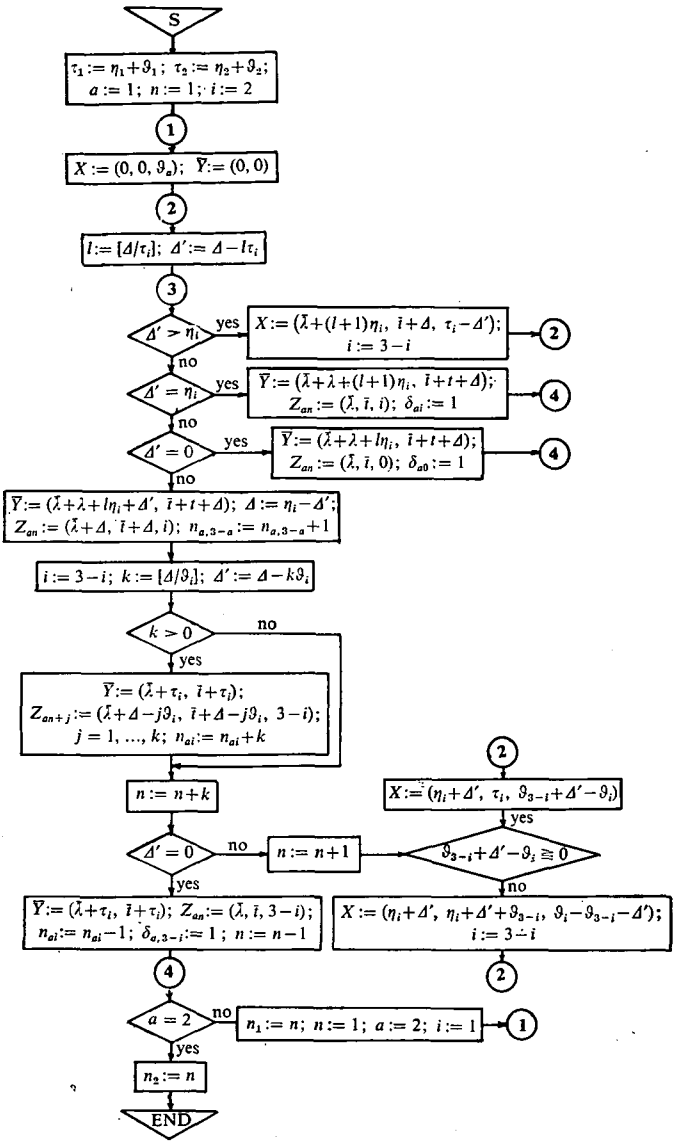


Fig. 14
The flow-chart of the *S*-algorithm

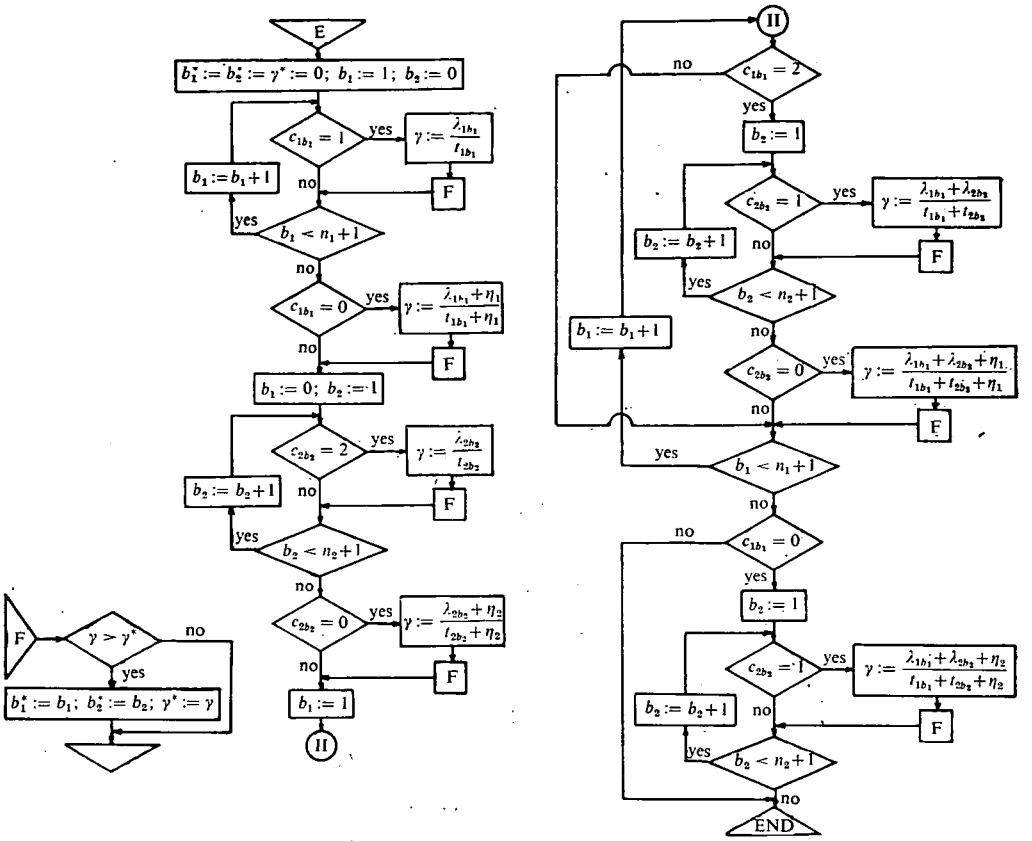


Fig. 15
The flow-chart of the *E*-algorithm

6. Summary

No simple rule to produce nor any simple method to choose an optimal schedule $R^*(Q)$ of any job-flow pair configuration Q is known. The dominance of the class of the consistent economical schedules (CESs) is proven here. We investigated the structure of the CESs and gave a classification for them. This is based upon the graph $G(Q)$ of the typical (critical) situations of two schedule sections $\Sigma^{(a)}$, $a=1, 2$. The information necessary to obtain $G(Q)$ and its data can be got by the *S*-algorithm if only $G(Q)$ is finite. In this case the *E*-algorithm supplies an optimal schedule and its efficiency. The discussion has shown the importance of some open problems which require further investigation. Such problems are: necessary and sufficient conditions for $G(Q)$ to be finite; estimations for the ratio of the efficiency values of CESs to the maximum value; detailed information about

some heuristic strategies such as priority schedules and the schedule with minimum local losses.

KEYWORDS: steady job-flow pairs, preemptive scheduling, economic schedules, dominance.

COMPUTER SERVICE FOR
STATE ADMINISTRATION
CSALOGÁNY U. 30—32.
BUDAPEST, HUNGARY
H—1015

References

[1] ARATÓ, M., Diffusion approximation for multiprogrammed computer systems, *Comput. Math. Appl.*, v. 1, 1975, pp. 315—326.
[2] TANKÓ, J., A study on scheduling steady job-flow pairs, *Tanulmányok — MTA Számítástech. Automat. Kutató Int. Budapest*, v. 82, 83, 1978 (in Hungarian).
[3] TANKÓ, J., On non-preemptive scheduling of steady job-flow pairs, Dissertation, Hungarian Academy of Sciences, Budapest, 1979 (in Hungarian).
[4] TANKÓ, J., Non-preemptive scheduling of steady job-flow pairs, *Found. Control Engrg.*, to appear. Reduction method for non-preemtive scheduling steady job-flow pairs, *Found. Control Engrg.*, to appear.
[5] TOMKÓ, J., Processor utilization study, *Comput. Math. Appl.*, v. 1, 1975, pp. 337—344.

(Received Oct. 24, 1979)

INDEX—TARTALOM

ISSN 0324—721 X

Kiadja a Szegedi József Attila Tudományegyetem
Felelős szerkesztő és kiadó: Gécseg Ferenc

80-2403 Szegedi Nyomda — Felelős vezető: Dobó József igazgató

A kézirat a nyomdába érkezett: 1980. június 5.
Megjelent: 1980. december hó
Példányszám: 1000. Terjedelem: 10,12 (A/5) ív
Készült monószedéssel, íves magasnyomással
az MSZ 5601 és az MSZ 5602—55 szabvány szerint