Tomus 6. Fasciculus 4.

ACTA CYBERNETICA

FORUM CENTRALE PUBLICATIONUM
CYBERNETICARUM HUNGARICUM

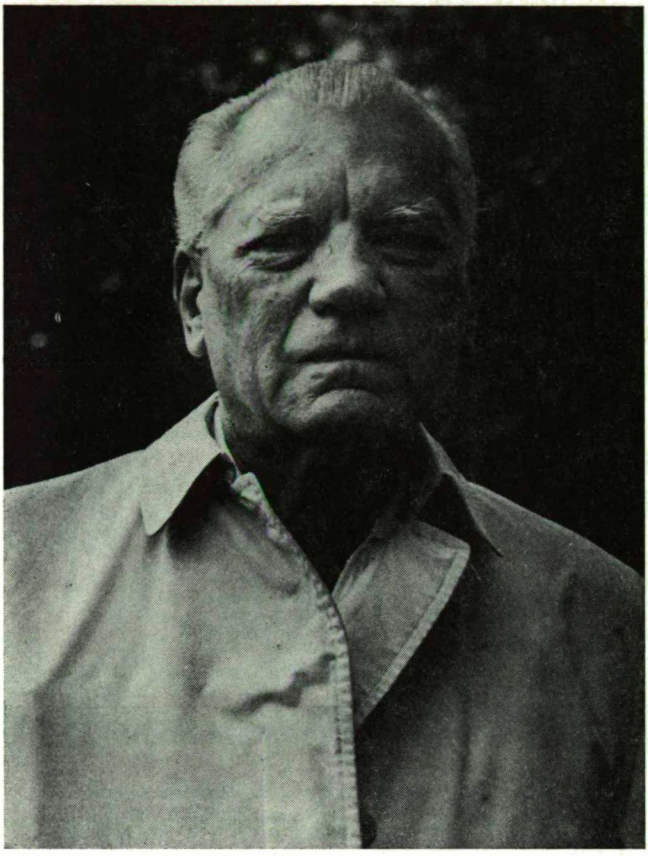
Szeged, 1984
Curat: Universitas Szegediensis de Attila József nominata

6. kötet 4. füzet

ACTA CYBERNETICA

A HAZAI KIBERNETIKAI KUTATÁSOK
KÖZPONTI PUBLIKÁCIÓS FÓRUMA

Szeged, 1984. augusztus
A Szegedi József Attila Tudományegyetem gondozásában



KÁLMÁN LISSÁK
(1908—1982)

Professor Lissák, the doyen of Hungarian physiologists and member of the Academy of Sciences died on 22 June 1982. He started his scientific career in 1933 at the Institute of Physiology of the Medical School in Debrecen, in 1943 became the director of the Institute of Physiology in Pécs and kept this post until his retirement in 1978. The establishment of the highest standards in the education of physiology and the introduction of modern neuro- and electrophysiological research methods in his country belonged to his dearest endeavours which he accomplished with much success. As a coworker of W. Cannon in 1938—39 Dr. Lissák was fortunate to become a witness of the historical discussions among Rosenblueth, Wiener and Bigelow which finally led to the foundation of cybernetics. Dr. Lissák became an early and arduous supporter of cybernetics in Hungary and faithfully served its cause until his death.

On certain partitions of finite directed graphs and of finite automata

By A. ÁDÁM

I. Introduction and basic terminology

§ 1.

The main aim of this paper is to study the partitions π of the vertex set of a finite directed graph G such that π satisfies the following condition: if the vertices a, b are in a common class modulo π and the edges $\vec{ac}$, $\vec{bd}$ exist in G, then c, d are also in a common class. These partitions will be called partitions having property P in the paper.

My attention was called to studying these partitions by the automaton-theoretical articles [7], [9], [10]. The majority of the present paper is written, however, from a graph-theoretical point of view.

Sections 3—5 are devoted to introducing the notions which are basic for the paper, and to exposing a few simple consequences of the definitions.

In Chapter II a description of the P-partitions of functional graphs will be given. The results of Chapter II will be generalized in Chapter III into an overview of the P-partitions of all (finite, directed) connected graphs in which no vertices with out-degree zero occur.

Chapter IV contains comments of several types. The extension of the former results to non-connected graphs is sketched, their extension to graphs with sinks is questioned and examples answering some arising questions will be given. § 12 is an appendix to the paper; it starts with lemmas on a sequence of partitions of the state set of a Moore automaton, later these facts lead to a proof solving a problem[1] on the complexity and state number of Moore automata.

[1] Conjecture 1 in [4].

§ 2.

The idea of studying the P-partitions by graph-theoretic methods was suggested by the articles [7], [9], [10] of Gill, Flexer and Hwang. They have dealt with questions concerning automata. The mentioned partition type is the same as the "partitions with substitution property" in their papers.[2]

Gill, Flexer and Hwang discussed mainly the partitions of the state set of an automaton such that the factor automaton (modulo the partition in question) exists and is a cycle. It turns out from their articles that the overview of these partitions has a certain technological significance.[3]

Yoeli and Ginzburg [14] introduced the P-partitions under the name "admissible partitions". They investigated chiefly the atoms[4] in the lattice of these partitions.

Dvořák, Gerbrich and Novotný deal in their most recent paper [5], essentially, with connected directed graphs in which no out-degree exceeds one. They describe all the possible homomorphisms (if there exists any) of a graph onto another.

The question, to whose solution Chapter II of the present paper is devoted, is the same (apart from terminological differences) as the problem of describing the congruences of connected finite unary algebras with one operation. The papers [11], [15], [16], [17] of Kopeček, Egorova and Skornjakov deal with somewhat related questions.

The autonomus semiautomata (possibly infinite ones) which are investigated by Machner and Strassner in [12] are essentially the same as the functional graphs in our terminology. Theorem 3 and Corollary 6′ in [12] concern to finite functional graphs, these results correspond to certain considerations in our Chapter II. The mentioned results are derived by Machner and Strassner as consequences of their investigations dealing with the infinite case.

§ 3.

By a graph, we mean a connected directed finite graph. Parallel edges with the same orientation are not permitted. We allow, however, loops and oppositely oriented parallel edges. Sometimes we regard a graph G as a relational structure, this means that we say "the relation $\alpha_G(a, b)$ holds" instead of saying "the edge from the vertex a to the vertex b exists in G".[5]

The most familiar notions of the theory of directed graphs are supposed to be known; especially, the notions of *path* and *cycle*. These are understood always in directed sense, and with pairwise different vertices. (Of course, the first vertex of a cycle and the last one are the same.)

The notion of *circuit* originates from the notion of cycle by the modification that the edges are considered as non-directed ones.

---

[2] In [7], [9], [10] automata without output signs are considered. Actually, the graph of an automaton is studied rather than the graph itself.

[3] See the middle of Section 1 in [7] and Section VII of [9].

[4] π is called an atom if $\pi \supset o$ holds and $\pi \supseteq \pi' \supset o$ implies $\pi = \pi'$ (where π, π' are P-partitions).

[5] The subscript G is possibly dropped in α_G if its absence cannot cause a misunderstanding. Similar notational simplifications may occur in other cases, too.

A vertex with in-degree zero is called a *source*. A vertex with out-degree zero is called a *sink*.

The lattice of all partitions of the vertex set V of a graph G is denoted by $L(V)$; as usual, $\pi_1 \subseteqq \pi_2$ means that $\pi_1 (\in L(V))$ is a (proper or non-proper) refinement of $\pi_2 (\in L(V))$. ι is the partition having one class only, and o is the partition each class of which consists of a single element.

Consider a partition $\pi (\in L(V))$. We say that π *possesses the property* P (or, simply, that π is a P-partition) if

$$(a \equiv b \,(\mathrm{mod}\, \pi) \,\&\, \alpha(a, c) \,\&\, \alpha(b, d)) \Rightarrow c \equiv d \,(\mathrm{mod}\, \pi)$$

holds universally (i.e., for every choice of the vertices a, b, c, d).

Denote by $[a]_\pi$ (or simply by $[a]$) the class (modulo π) containing a vertex a. The *factor graph* $G^* = G/\pi$ is defined in the following manner:

the vertices of G^* are the classes of V modulo π,

$\alpha^*([a], [b])$ holds[6] if and only if there exist two vertices $a' (\in V), b' (\in V)$ such that $a' \in [a], b' \in [b]$ and $\alpha(a', b')$.

It is clear that G/π can have loops even if G is loop-free.

We end this § by asserting two obvious statements concerning the above notion of the factor graph. The first of them is an analogon of one of the general isomorphism theorems of universal algebras.

Lemma 1. *Let π_1, π_2 be two partitions of the vertex set V of a graph G. Suppose $\pi_1 \subseteqq \pi_2$; denote by π_2' the following partition of the vertex set of G/π_1: $[a]_{\pi_1} \equiv [b]_{\pi_1}$ (mod π_2') if and only if $a \equiv b$ (mod π_2). Then G/π_2 and $(G/\pi_1)/\pi_2'$ are isomorphic.*

Lemma 2. *Let π be a partition of the vertex set of a graph G. If there is no source (or no sink) in G, then there is no source (or no sink, resp.) in G/π, too.*

§ 4.

A graph G is called a *functional graph* if the out-degree of each vertex of G is one. A simple structural description of the finite functional graphs is due to Ore (see [13], § 4.4; [1], Chapter I); his theorem states that *a connected graph G is functional if and only if*

G has precisely one circuit,

the circuit in G is a cycle, and

each other edge of G is directed towards the cycle.

By its definition, a functional graph G does not contain a sink: G contains at least one source unless G is a cycle.

The vertices and edges of the cycle of a functional graph G are called *cyclic*. Each other vertex and edge of G is said *acyclic*. (A source is always acyclic.)

If a is a vertex of a functional graph G, then we denote by $\varphi_G(a)$ the (uniquely determined) vertex b for which $a_G(a, b)$ is true. We define $\varphi^i(a)$ by $\varphi^i(a) = \varphi(\varphi^{i-1}(a))$ recursively; we agree that $\varphi^0(a) = a$.

[6] We write α^* instead of α_{G^*}.

Let a, b be two vertices of a functional graph. If there is a number $i(\geqq 0)$ such that $\varphi^i(a)=b$, then we denote by $\chi(a, b)$ the smallest of these numbers.

Let a path in a functional graph G be considered whose vertices are

$$a_1, a_2, \ldots, a_s \quad (s \geqq 2). \tag{4.1}$$

If $a_1, a_2, \ldots, a_{s-1}$ are acyclic vertices and a_s is cyclic, then we call (4.1) a *principal path*. To each acyclic vertex a_1, there is exactly one principal path starting from a_1.

Let B be a subset of V such that every element of B is an acyclic vertex. B is called a *basic set* if, to each $b(\in B)$, the principal path starting with b contains no other element of B than b. The empty set is regarded to be basic, too. (Thus each functional graph — even a cycle — has at least one basic set.) The set of all sources of a functional graph is always basic. For any basic set B, a principal path may contain at most one element of B.

Consider a basic set B of a functional graph. A vertex a is called *outer with respect to* B if a is acyclic and the principal path starting with a contains an element of B. The remaining vertices are called *inner (with respect to B)*. The following lemma is obvious:

Lemma 3. *Let B be a basic set in a functional graph. Then*
(i) *each element of B is outer with respect to B,*
(ii) *each cyclic vertex of the graph is inner with respect to B, and*
(iii) *if a is inner with respect to B, then $\varphi(a)$ is also inner.*

In the last assertion of this § we state a connection between the P-partitions and a slight extension of the class of functional graphs.

Proposition 1. *Let G be a graph and π be a partition of its vertex set. π has the property* P *if and only if each out-degree in the factor graph G/π is either zero or one.*

Proof. The out-degree of a vertex $[a]$ of G/π is at least two if and only if there exist four vertices b_1, b_2, c, d in G such that $b_1 \in [a]$, $b_2 \in [a]$, $\alpha(b_1, c)$, $\alpha(b_2, d)$ and $c \not\equiv d \pmod{\pi}$. This condition is precisely the negation of the property P.

§ 5.

In this last section of Chapter I, a few concepts of the theory of automata will be recalled or introduced. These notions are referred to in § 2 and § 12 only.

The notion of the Moore automaton is well-known, we denote such an automaton by $\mathbf{A}=(A, X, Y, \delta, \lambda)$.

Let a, b be two states; the length of a shortest (input) word p such that $\lambda(\delta(a, p)) \neq \lambda(\delta(b, p))$ is denoted by $\omega(a, b)$. The maximum of $\omega(a, b)$ (taken for pairs of different states) is called the *complexity* of $\mathbf{A}$.

Let us define the partitions[7] η_k in its state set A in such a manner that $a \equiv b$

[7] It follows from Proposition 16 of [2] that each η_k is really a partition. In [2], I have written R_k instead of η_k.

(mod η_k) holds exactly when $\omega(a, b) \geqq k$. It is obvious that

$$\eta_0 \supseteq \eta_1 \supseteq \eta_2 \supseteq \eta_3 \supseteq \cdots \tag{5.1}$$

and η_0 equals the maximal partition of A.

If we consider an automaton **A** such that the output set Y and the output function λ are not taken into account, then we speak of an *automaton without output signs*.

Let **A** be an automaton. Let us construct a directed graph G in the following way:

the states of **A** are the vertices of G, and

$\alpha_G(a, b)$ holds if and only if there is at least one $x(\in X)$ satisfying $\delta(a, x)=b$.

Then G is called the *graph of the automaton* **A**. (It is clear that we have regarded **A** as an automaton without output signs in this definition.)

II. Partitions having the property P in functional graphs

§ 6.

Construction I. Let G be a functional graph, B a basic set in G and d a divisor of the length of the cycle of G.

We form an augmenting sequence

$$G_1, G_2, \ldots, G_t \tag{6.1}$$

of induced subgraphs of G such that

(α) the vertex set V_1 of G_1 equals the set of inner vertices (with respect to B),

(β) the vertex set V_i of G_i consists of the vertices a which satisfy $\varphi(a) \in V_{i-1}$ (V_{i-1} is the vertex set of G_{i-1}; $2 \leqq i \leqq t$),

(γ) the sequence (6.1) terminates when we reach G (in the form of G_t).[8]

Let us construct a sequence

$$\pi_1, \pi_2, \ldots, \pi_t = \pi \tag{6.2}$$

(of partitions) according to the following rules (A)—(F):

(A) π_i is a partition of V_i (where $1 \leqq i \leqq t$).

(B) *(Initial step)* Choose a cyclic vertex c of G. Let

$$a \equiv b \pmod{\pi_1}$$

hold for $a(\in V_1)$ and $b(\in V_1)$ exactly when

$$\chi(a, c) \equiv \chi(b, c) \pmod{d}.$$

(C) Suppose that the partition π_{i-1} (of V_{i-1}) has already been defined (where $2 \leqq i \leqq t$). Denote by τ_i the following partition of V_i: $a \equiv b \pmod{\tau_i}$ precisely if

either $a=b$,

or $a \in V_i - V_{i-1}$, $b \in V_i - V_{i-1}$ and $\varphi(a) \equiv \varphi(b) \pmod{\pi_{i-1}}$.

[8] It is clear that $V_2 = B \cup V_1$.

(D) Assume that the partition π_{i-1} (of V_{i-1}) has already been defined (where $2 \leqq i \leqq t$). Denote by π_i' the subsequent partition of V_i: $a \equiv b \pmod{\pi_i'}$ exactly if either $a=b$,
or $a \in V_{i-1}, b \in V_{i-1}$ and $a \equiv b \pmod{\pi_{i-1}}$.

(E) *(Ordinary step)* Choose an arbitrary partition π_i^* of V_i such that $\pi_i^* \subseteqq \tau_i$. Form the union $\pi_i^* \cup \pi_i'$ and denote it by π_i.

(F) The construction of the sequence (6.2) contains an initial step and $t-1$ ordinary steps.

Remarks. If $a \in V_{i-1}$ and $b \in V_i - V_{i-1}$, then $a \not\equiv b$ modulo any of the partitions $\pi_i, \pi_{i+1}, \ldots, \pi_t (=\pi)$. — If $a \in V_i$ and $b \in V_i$, then either the congruence $a \equiv b$ is true for all of the partitions $\pi_i, \pi_{i+1}, \ldots, \pi_t$ or it is false for all of them. — The following three assertions are equivalent (for a performance of Construction I):

(1) B is empty,

(2) every vertex is inner and the construction collapses to the initial step,

(3) G/π is a cycle.

If G is a cycle, then the assertions (1), (2), (3) are true. — If B is empty and $d=1$, then π equals the maximal partition ι of V. — If B is chosen as the set of all acyclic vertices a fulfilling the statement that $\varphi(a)$ is cyclic, the number d is chosen as the cycle length of G and each π_i^* is the minimal partition of V_i, then π equals the minimal partition o of V.

Lemma 4. *The initial step of Construction I is independent of the choice of the cyclic vertex c.*

Proof. Apply the initial step with $c^{(1)}$ and $c^{(2)}$, resp. (instead of c). Denote $\chi(c^{(1)}, c^{(2)})$ by q. Let the originating partitions be $\pi_1^{(1)}$ and $\pi_1^{(2)}$.

Suppose $a \equiv b \pmod{\pi_1^{(1)}}$. Denote

$$\frac{\chi(a, c^{(1)}) - \chi(b, c^{(1)})}{d}$$

by k and p/d by m where p is the length of the cycle of G. It is easy to see that $\chi(a, c^{(2)})$ equals either $\chi(a, c^{(1)})+q$ or $\chi(a, c^{(1)})-(p-q)$ and a similar assertion holds with b (instead of a). A discussion shows that $\chi(a, c^{(2)})-\chi(b, c^{(2)})$ is equal to one of $kd, (k+m)d, (k-m)d$. Hence $\chi(a, c^{(2)}) \equiv \chi(b, c^{(2)}) \pmod{d}$ and $a \equiv b \pmod{\pi_1^{(2)}}$.

An analogous inference shows that $a \equiv b \pmod{\pi_1^{(2)}}$ implies $a \equiv b \pmod{\pi_1^{(1)}}$.

Proposition 2. *Consider two performances of Construction I for a graph G; suppose that we start with the pairs $(B^{(1)}, d^{(1)})$ and $(B^{(2)}, d^{(2)})$, respectively. Denote the obtained partitions by $\pi^{(1)}$ and $\pi^{(2)}$. If $\pi^{(1)}=\pi^{(2)}$, then $B^{(1)}=B^{(2)}$, $d^{(1)}=d^{(2)}$ and the two performances are stepwise coinciding.*

Proof. We verify the statement indirectly.

If $B^{(1)} \neq B^{(2)}$, then there is a vertex a which is inner with respect to one of $B^{(1)}, B^{(2)}$ (e.g. to $B^{(1)}$) and outer with respect to the other one. Thus $a \equiv b \pmod{\pi^{(1)}}$ is satisfiable with at least one cyclic vertex b, but $a \equiv b \pmod{\pi^{(2)}}$ is not satisfiable by any cyclic b.

Let $d^{(1)}, d^{(2)}$ be different, we can suppose $d^{(1)}<d^{(2)}$. Choose a cyclic vertex a. a and $\varphi^{d^{(1)}}(a)$ are congruent modulo $\pi^{(1)}$ but they are incongruent modulo $\pi^{(2)}$.

Finally, we consider the case when $B^{(1)}=B^{(2)}$, $d^{(1)}=d^{(2)}$ and the two performances of Construction I differ from each other. The first difference between them will appear in the following manner: in two ordinary steps (corresponding to each other in the performances), $\pi_i^{*(1)}, \pi_i^{*(2)}$ act differently on the set V_i-V_{i-1}. It is evident (by the second sentence of the remarks above) that the partitions $\pi^{(1)}, \pi^{(2)}$ act on V_i-V_{i-1} in the same manner as $\pi_i^{*(1)}$ and $\pi_i^{*(2)}$, respectively.

We have got $\pi^{(1)}\neq\pi^{(2)}$ when the two performances do not agree with each other completely.

§ 7.

Theorem 1. *The following three assertions are equivalent for a partition π of the vertex set V of a functional graph G:*

(I) *G/π is a functional graph,*

(II) *π has the property* P,

(III) *π can be obtained by Construction* I.

Proof. The equivalence of (I) and (II) follows immediately from Proposition 1 and Lemma 2. In what follows, we strive to show the equivalence of (II) and (III).

(II)⇒(III). Let us start with a P-partition π of V. Our aim is to determine a performance of Construction I such that π is obtained by this performance. In details the determination of the performance will consist of the following phases (α)—(ε):

(α) we determine a basic set B,

(β) we determine a divisor d of the length of the cycle of G,

(γ) we prove that if we choose two vertices a_1, a_2 and two elements b_1, b_2 of B such that the numbers $\chi(a_1, b_1)$ and $\chi(a_2, b_2)$ are defined and they do not coincide, then $a_1\not\equiv a_2 \pmod{\pi}$,

(δ) we determine the partitions $\pi_2^*, \pi_3^*, \pi_4^*, \ldots$,

(ε) we show that each π_i^* is a refinement of τ_i.

We turn to elaborate the parts of the proof (of (II)⇒(III)) exposed above.

(α) Denote by C the set of all vertices a of G such that $[a]_\pi$ contains at least one cyclic vertex. Denote by B the set of vertices b such that $b\notin C$ and $\varphi(b)\in C$ are valid. It is clear that B consists of acyclic vertices. We are going to show that to any $b(\in B)$ no positive i can satisfy $b\equiv\varphi^i(b) \pmod{\pi}$. Suppose the contrary. It is easy to see (by the property P) that $b, \varphi^i(b), \varphi^{2i}(b), \varphi^{3i}(b), \ldots$ belong to a common class modulo π, this is impossible since $[b]_\pi$ cannot contain a cyclic vertex.

(β) Let a be an element of C, denote by $\eta(a)$ the smallest positive integer i such that $a\equiv\varphi^i(a) \pmod{\pi}$.

Consider a vertex $a(\in C)$, let i be the (minimal) number occurring in the definition of $\eta(a)$. Then

$$\varphi(a) \equiv \varphi(\varphi^i(a)) = \varphi^i(\varphi(a)) \pmod{\pi},$$

hence

$$\eta(a) \geqq \eta(\varphi(a)). \tag{7.1}$$

If (7.1) is applied for the vertices of the cycle of the graph, we get easily that $\eta(a)$ is common for the *cyclic* vertices. Denote this common value by d and the cycle length by p.

Our next aim is to verify that d is a divisor of p. Let k be the smallest integer such that $kd>p$. Since the deduction

$$c \equiv \varphi^d(c) \equiv \varphi^{2d}(c) \equiv \varphi^{3d}(c) \equiv \ldots \equiv \varphi^{kd}(c) = \varphi^{kd-p}(c) \pmod{\pi}$$

holds for an arbitrary cyclic vertex c, we have $kd-p \geqq \eta(c)(=d)$ by the minimality condition in the definition of η. On the other hand, the minimality condition in the definition of k implies $kd-p \leqq d$. Consequently $kd-p=d$, thus $(k-1)d=p$ and $d|p$.

Consider now an acyclic vertex $a(\in C)$, let c be a cyclic vertex such that $a \equiv c \pmod{\pi}$. We have $\varphi^i(a) \equiv \varphi^i(c) \pmod{\pi}$ for every i, this fact implies $\eta(a) \equiv \eta(c) = d$.

(γ) We can suppose $\chi(a_1, b_1) < \chi(a_2, b_2)$ without an essential restriction of the generality. Denote $\chi(a_1, b_1)$ by j. It is clear that $\varphi^{j+1}(a_1) \in C$ and $\varphi^{j+1}(a_2) \notin C$, hence $a_1 \not\equiv a_2 \pmod{\pi}$.

(δ) Denote by V_i (where $i \geqq 1$) the set of vertices a satisfying $\varphi^{i-1}(a) \in C$. (It is clear that $V_i \supseteq V_{i-1}$ if $i \geqq 2$.) Let π_i^* be a partition of V_i defined by what follows: $a \equiv b \pmod{\pi_i^*}$ (where $a \in V_i$, $b \in V_i$) if and only if

$$\text{either} \quad a = b$$

$$\text{or} \quad a \notin V_{i-1}, \ b \notin V_{i-1} \quad \text{and} \quad a \equiv b \pmod{\pi}.$$

(ε) is obviously true with the above definition of the partitions π_i^*.

We have completed the determination of the "parameters" B, d and $\pi_2^*, \pi_3^*, \ldots$ occurring in Construction I. A routine inference shows (together with (γ)) that we obtain just π if we perform the construction with these "parameters".

(III)$\Rightarrow$(II). Consider a partition π which has been obtained by Construction I. Similarly to the preceding part of the proof, we denote by C the set of those vertices a for which $[a]_\pi$ contains a cyclic vertex.

Suppose $a \equiv b \pmod{\pi}$ where $a \neq b$.

If $a \in C$, then clearly $b \in C$. Let us choose an arbitrary cyclic vertex c. Either

$$\chi(\varphi(a), c) = \chi(a, c) - 1$$

or

$$\chi(\varphi(a), c) = p - 1 \equiv -1 = \chi(a, c) - 1 \pmod{d}$$

(according as $a \neq c$ or $a = c$), and the analogous statement holds for b (instead of a). Therefore we have

$$\chi(\varphi(a), c) \equiv \chi(a, c) - 1 \equiv \chi(b, c) - 1 \equiv \chi(\varphi(b), c) \pmod{d},$$

thus $\varphi(a) \equiv \varphi(b) \pmod{\pi}$.

If a and b do not belong to C, then they are necessarily contained in the same difference set $V_i - V_{i-1}$. $a \equiv b$ is valid modulo each of $\pi = \pi_t, \pi_{t-1}, \pi_{t-2}, \ldots, \ldots, \pi_i, \pi_i^*$ and τ_i (by the construction). We get $\varphi(a) \equiv \varphi(b) \pmod{\pi_{i-1}}$ by the rule (C), hence the elements $\varphi(a)$ and $\varphi(b)$ of V_{i-1} are congruent modulo each of $\pi_i, \pi_{i+1}, \ldots, \pi_t = \pi$, too.

The fulfilment of the property P is proved.

The next assertion is an easy consequence of the procedure described in Construction I and of the notion of factor graph:

Proposition 3. *Let G be a functional graph and π be a partition (in G) produced by Construction I. The cycle length of the factor graph G/π equals d. G/π is a cycle if and only if B is empty.*

III. Partitions having the property P in arbitrary sink-free graphs

§ 8.

Let G be a directed graph. We introduce a quaternary relation $\varkappa$ and some binary relations in the set V of vertices of G.

Let $\varkappa(a, b, c, d)$ hold for the (not necessarily different) vertices a, b, c, d if there is a positive integer k and there exist $2k$ vertices $f_1, f_2, \ldots, f_k, g_1, g_2, \ldots, g_k$ such that the equalities

$$a = f_1,\ b = f_k,\ c = g_1,\ d = g_k \tag{8.1}$$

and the $2k-2$ relations

$$\alpha(f_1, f_2),\ \alpha(f_2, f_3),\ \ldots,\ \alpha(f_{k-1}, f_k), \tag{8.2}$$

$$\alpha(g_1, g_2),\ \alpha(g_2, g_3),\ \ldots,\ \alpha(g_{k-1}, g_k) \tag{8.3}$$

are true. $\varkappa(a, a, c, c)$ is regarded to be always valid (with the choice $k=1$) both when $a=c$ and when $a \neq c$. It is clear that $\varkappa(a, b, c, d)$ and $\varkappa(c, d, a, b)$ are equivalent.

Let $\varrho(a, b)$ be true if there is a $c(\in V)$ such that $\varkappa(c, a, c, b)$. Denote the transitive extension of ϱ by ε.

In Chapter III, our aim is to characterize the P-partitions of the sink-free graphs by use of the partition ε.

Remark. If G is a functional graph, then $\varepsilon = o$.

§ 9.

Lemma 5. *If π_1 and π_2 are partitions with property P, then $\pi_1 \cap \pi_2$ is a P-partition, too.*

Proof. If $a \equiv b \pmod{\pi_1 \cap \pi_2}$, $\alpha(a, c)$ and $\alpha(b, d)$ are true, then both of $c \equiv d \pmod{\pi_1}$, $c \equiv d \pmod{\pi_2}$ hold.

Proposition 4. *There is a (uniquely determined) P-partition π^* such that $\pi^* \subseteqq \pi$ for each P-partition.*

Proof. G is a finite graph, hence the intersection π^* of all P-partitions possesses property P by a successive application of Lemma 5.

Proposition 5. *We have* $\pi^* \supseteq \varepsilon$.

Proof. Let a, b be two vertices such that $\varrho(a, b)$. There is a vertex c such that $\varkappa(c, a, c, b)$. Consider the $2k$ vertices occurring in (8.2), (8.3). (These vertices fulfil now $c=f_1=g_1$, $a=f_k$, $b=g_k$ instead of (8.1).) Since π^* has property P, $f_i \equiv g_i \pmod{\pi^*}$ follows inductively; especially,

$$a = f_k \equiv g_k = b \pmod{\pi^*}.$$

We have shown that $\varrho(a, b)$ implies $a \equiv b \pmod{\pi^*}$. Consequently, $a \equiv b \pmod{\varepsilon}$ implies $a \equiv b \pmod{\pi^*}$ (because ε is the transitive extension of ϱ).

Lemma 6. *If G has no sink, then ε is a* P*-partition.*

Proof. Assume $\varrho(a, b)$, $\alpha(a, c)$ and $\alpha(b, d)$ for some vertices a, b, c, d. Then there is a vertex h such that $\varkappa(h, a, h, b)$, hence $\varkappa(h, c, h, d)$, thus $\varrho(c, d)$.

Suppose $a \equiv b \pmod{\varepsilon}$, $\alpha(a, c)$, $\alpha(b, d)$ for an arbitrary quadruple a, b, c, d. There exist vertices $a_1, a_2, \ldots, a_k$ such that $\varrho(a_{i-1}, a_i)$ for each i $(2 \leqq i \leqq k)$ and $a_1=a$, $a_k=b$. We can choose $k-2$ vertices $c_2, c_3, \ldots, c_{k-1}$ such that $\alpha(a_i, c_i)$ holds $(2 \leqq i \leqq k-1)$. By the beginning sentences of the proof, $\varrho(c_{i-1}, c_i)$ if $3 \leqq i \leqq k-1$; furthermore, $\varrho(c, c_2)$ and $\varrho(c_{k-1}, d)$. Therefore $\varepsilon(c, d)$.

Proposition 6. *If the directed graph G has no sink, then $\pi^* = \varepsilon$.*

Proof. $\pi^* \supseteq \varepsilon$ was stated in Proposition 5. $\pi^* \subseteq \varepsilon$ is an immediate consequence of Proposition 4 and Lemma 6.

Propositions 1, 6 and Lemma 2 imply

Corollary 1. *If G has no sink, then G/ε is a functional graph.*

Construction II. Let G be a graph without sinks. Denote the factor graph G/ε by G^*. Choose a partition π' of the vertex set of G^* such that π' is obtained by Construction I. Define a partition π in the vertex set V of G in the following manner: $a \equiv b \pmod{\pi}$ holds for $a(\in V), b(\in V)$ exactly when $[a]_\varepsilon \equiv [b]_\varepsilon \pmod{\pi'}$.

Theorem 2. *Let G be a directed graph without sinks. The following three assertions are equivalent for a partition π of the vertex set of G:*

(i) *G/π is a functional graph,*

(ii) *π has the property* P,

(iii) *π can be obtained by Construction II.*

Proof. The theorem becomes clear by comparing the following earlier results: Theorem 1, Propositions 1, 4, 6, Corollary 1, Lemmas 1 and 2. (Now Lemma 1 is applied for ε and π instead of π_1 and π_2, resp.)

Proposition 7. *Let G be a graph without sinks. The length p of the cycle of the functional graph G/ε divides the greatest common divisor p^* of all cycle lengths of G.*

Sketch of the *proof.* Choose an arbitrary cycle Z' in G, denote the length of Z' by p'. Let us start with a vertex of Z' and pass through all the vertices of Z'; consider the corresponding vertices of G/ε. We have passed through the

cycle Z of G/ε either one or more times; in any case, the number of surroundings of Z is an *integer*. Thus $p|p'$.

Since the same assertion holds for each choice of Z', we have $p|p^*$.

Corollary 2. *Let G be a graph without sinks. Then the following two numbers are equal:*

(a) *the number of partitions π such that G/π is a cycle,*

(b) *the number of divisors of the cycle length p of G/ε (including 1 and p).*

Proof. Recall Construction II, Theorem 2 and Proposition 3. It is clear that G/π is a cycle if and only if π' is constructed (in G/ε) by such a performance of Construction I that B is empty. This means that we have (precisely) the freedom of choosing a divisor of the cycle length of G/ε arbitrarily.

IV. Remarks, examples; an appendix

§ 10.

1. In the previous sections, a complete description of the partitions having property P of connected finite directed graphs without sinks was obtained. In the present remark, we shall outline how this description can be extended to non-connected graphs.

Let G be a non-connected directed graph containing no sink. Then G can be represented (in at least one manner) as the disjoint union of two graphs[9] G_1, G_2. Consider a partition π of the vertex set V of G; denote by π_i (where i can be 1 or 2) the restriction of π to the vertex set V_i of G_i. Let $[a]_\pi$ be an arbitrary π-class; evidently, either $[a]_\pi=[a]_{\pi_1}$ or $[a]_\pi=[a]_{\pi_2}$ (where necessarily $a\in V_1$ or $a\in V_2$, resp.) or $[a]_\pi=[b_1]_{\pi_1}\cup[b_2]_{\pi_2}$ with suitable vertices $b_1(\in V_1)$ and $b_2(\in V_2)$. It is easy to see the validity of the following assertion:

Proposition 8. *A partition π of V has property P if and only if*

π_1, π_2 are P-partitions, and

whenever $\alpha(a, b)$ holds in G and $[a]_\pi$ is the union of a π_1-class and a π_2-class, then the same statement holds for $[b]_\pi$, too.

The above idea can be utilized in such a way that first we form G/ε (which is clearly the disjoint union of G_1/ε and G_2/ε), we apply the proposition for $G/\varepsilon, G_1/\varepsilon$ and G_2/ε (instead of G, G_1, G_2, resp.), and we form the P-partitions of G by using the P-partitions of G/ε (analogously to Construction II).

2. The exposed theory admits a dualization with respect to reversing the orientation of edges. (The dual of a functional graph is a graph in which all in-degrees are one. Sources and sinks are dual to each other. The duals of the P-partitions are the partitions satisfying

$$(c \equiv d \,(\mathrm{mod}\,\pi)\ \&\ \alpha(a, c)\ \&\ \alpha(b, d)) \Rightarrow a \equiv b \,(\mathrm{mod}\,\pi).$$

[9] Each connected component of G is either a connected component of G_1 or a connected component of G_2.

The dual of $\varrho(a, b)$ is true exactly if there is a c such that $\varkappa(a, c, b, c)$ holds. And so on.)

3. It can be shown that the P-partitions of a sink-free graph form a lattice. The maximal element of this lattice is ι, its minimal element is ε.

4. In [7], [9], [10] also "input-independent partitions" have been studied. This notion is a slight modification of the concept of "partition with substitution property" (i.e., with property P). In our terminology, a partition π is called input-independent when

$$(\alpha(a, c) \ \& \ \alpha(a, d)) \Rightarrow c \equiv d \,(\mathrm{mod}\, \pi)$$

is universally true.

It is easy to see that this property is satisfied exactly when $\pi \supseteq \varepsilon^*$ holds where ε^* is the transitive extension of the following relation ϱ^*: $\varrho^*(a, b)$ is valid if either $a=b$ or there exists a c such that $\alpha(c, a)$ & $\alpha(c, b)$.

5. We finish the section with exposing two open questions.

Problem 1. Let an overview of the P-partitions of the finite directed graphs containing sinks be given.

Problem 2. When does $p=p^*$ hold in Proposition 7?

§ 11.

In this section, we shall see some examples. The first example is used for illustrating how Constructions I, II are performed. This example and the two subsequent ones will serve for deciding the following questions:

(A) Is the relation ϱ always transitive, or is it really needed that it should be extended transitively? (Cf. § 8.)

(B) Can it happen that $\pi \supset \varepsilon$ for a sink-free graph, but π does not possess property P? (Cf. Propositions 4, 6.)

(C) Is the condition that sinks are not allowed indispensable in Proposition 6?

(D) Is $p<p^*$ possible in Proposition 7?

First, let us consider the graph G_1 seen on Fig. 1a. Since $\varrho(c, f)$ and $\varrho(f, g)$ are valid but $\varrho(c, g)$ does not hold, the transitive extension is a proper step when ε is formed. The classes modulo ε are:

$$\{a\},\ \{b, d\},\ \{c, f, g\},\ \{e\}.$$

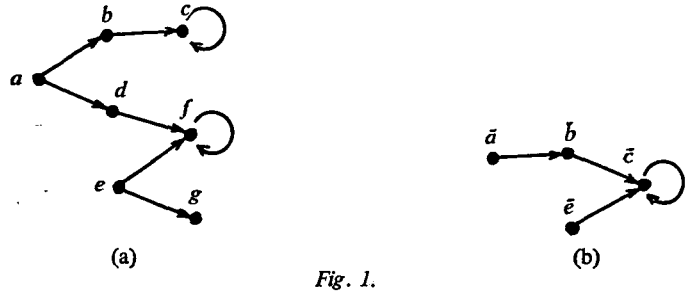


Fig. 1.

Fig. 1b shows the factor graph G_1/ε. (We write e.g. $\bar{a}$ instead of $[a]_\varepsilon$.) In G_1/ε, there is only one choice for d, namely $d=1$. We have six possibilities for choosing B, and Construction I can be performed in eight manners, the resulting partitions are seen on Table 1. (If $|B|=2$, then we have two possibilities for the choice of π_2^*, because τ_2 is the maximal partition of B.) The vertex set of G_1/ε has fifteen partitions; the remaining seven ones — among these,

$$\langle\{\bar{a}, \bar{c}\},\ \{\bar{b}\},\ \{\bar{e}\}\rangle \tag{11.1}$$

— do not have property P.

Table 1.

The elements of B	The classes modulo π
—	$\{\bar{a}, \bar{b}, \bar{c}, \bar{e}\}$
$\bar{a}$	$\{\bar{a}\}, \{\bar{b}, \bar{c}, \bar{e}\}$
$\bar{b}$	$\{\bar{a}\}, \{\bar{b}\}, \{\bar{c}, \bar{e}\}$
$\bar{e}$	$\{\bar{a}, \bar{b}, \bar{c}\}, \{\bar{e}\}$
$\bar{a}, \bar{e}$	$\{\bar{a}\}, \{\bar{e}\}, \{\bar{b}, \bar{c}\}$
	$\{\bar{a}, \bar{e}\}, \{\bar{b}, \bar{c}\}$
$\bar{b}, \bar{e}$	$\{\bar{a}\}, \{\bar{b}\}, \{\bar{e}\}, \{\bar{c}\}$
	$\{\bar{a}\}, \{\bar{b}, \bar{e}\}, \{\bar{c}\}$

Let us apply Construction II (with the partitions π of G_1/ε in the role of π'), we get that G_1 has eight P-partitions (from among the 15 partitions π fulfilling $\pi \supseteq \varepsilon$). E.g.,

$$\langle\{a\},\ \{b, c, d, f, g\},\ \{e\}\rangle$$

is a P-partition of G_1 (obtained from the fifth row of Table 1), but

$$\langle\{a, c, f, g\},\ \{b, d\},\ \{e\}\rangle \tag{11.2}$$

(got from (11.1)) is not a P-partition; in fact, $\alpha(a, b)$, $\alpha(c, c)$ hold and $a \equiv c$ but $b \not\equiv c$ modulo the partition (11.2).

The relation ε for the graph G_2 in Fig. 2 (containing three sinks) has the following equivalence classes:

$$\{a\},\ \{b\},\ \{c, d, e\},\ \{f\},\ \{g\}.$$

ε does not possess property P because $c \equiv e$ but $f \not\equiv g \pmod{\varepsilon}$. Therefore $\varepsilon \neq \pi^*$ in G_2.

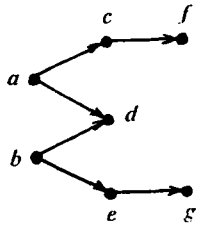


Fig. 2.

Consider the graph G_3 in Fig. 3a. ε has (on G_3) the equivalence classes

$$\{a\},\ \{b, c, d, e, f, g\}.$$

G_3/ε is seen in Fig. 3b. We can observe that $p=1<6=p^*$ (with the notations of Proposition 7).

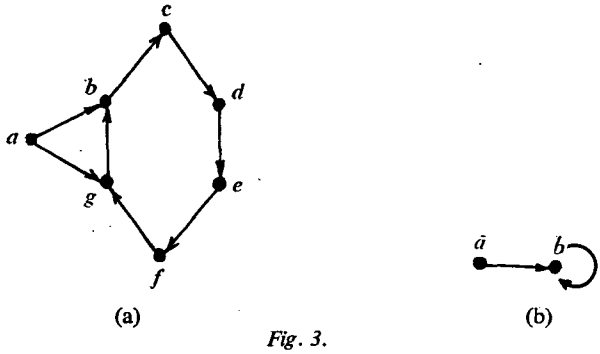


Fig. 3.

Summarizing, the examples show that the answers to the questions (A), (B), (C), (D) are: "the transitive extension is really needed", "yes", "yes", "yes", respectively.

Each counter-example given above contains a source.

Problem 3. Do the above answers to the questions (A)—(D) remain unchanged when we restrict ourselves to graphs without sources?

§ 12. (Appendix)

In this section our aim is to give a simple proof[10] for Conjecture 1 posed in [4].

Lemma 7. *Consider the sequence* $\eta_0, \eta_1, \eta_2, \ldots$ *of partitions of the state set* A *of a finite Moore automaton* $\mathbf{A}=(A, X, Y, \delta, \lambda)$. *If* $\eta_{i-1}=\eta_i$ *for some positive* i, *then* $\eta_i=\eta_{i+1}$.

[10] It should be noted that the idea of the present considerations is similar to a thought occurring in [8], p. 14.

Proof. Suppose $\eta_i \supset \eta_{i+1}$, we are going to show $\eta_{i-1} \supset \eta_i$. The supposition means that there are two states a, b such that $\omega(a, b)=i$. We have

$$\omega(\delta(a, x),\ \delta(b, x)) \geqq i-1 \tag{12.1}$$

for each choice of $x(\in X)$ and there is an $x^*(\in X)$ for which equality holds in (12.1). The state pair $\delta(a, x^*), \delta(b, x^*)$ is congruent modulo η_{i-1} but incongruent modulo η_i.

The next assertion is an easy consequence of Lemma 7.

Lemma 8. *Let* A *be as in the preceding lemma, denote* $|A|$ *by* v. *Let* m *be the smallest number such that* $\eta_m=\eta_{m+1}$. *Then* $m \leqq v-1$.

Lemma 9. *Let* A, v, m *be as in Lemmas* 7, 8. *If two states* a, b *satisfy* $\omega(a, b) \geqq v-1$, *then* $\omega(a, b)=\infty$.

Proof. Assume that a, b are congruent modulo η_{v-1}. They are congruent modulo η_m by Lemma 8 and (5.1); consequently, by the definition of m and Lemma 7, they are congruent modulo each of $\eta_{m+1}, \eta_{m+2}, \eta_{m+3}, \ldots$ (ad infinitum).

Proposition 9 ([4], Conjecture 1). *Let* A *be a finite Moore automaton such that the number* v *of its states satisfies* $v \geqq 2$. *Denote the complexity of* A *by* k. *If* k *is finite, then* $k \leqq v-2$.

Proof. By the finiteness of k, $\omega(a, b)$ is infinite (if and) only if $a=b$. Lemma 9 assures $\omega(a, b) \leqq v-2$ whenever $a \neq b$.

Corollary 3 of [3] shows that Proposition 9 cannot be sharpened.

MATHEMATICAL INSTITUTE OF THE
HUNGARIAN ACADEMY OF SCIENCES
REÁLTANODA U. 13—15.
BUDAPEST, HUNGARY
H—1364, P. O. BOX 127

References

[1] ÁDÁM, A., Gráfok és ciklusok (Graphs and cycles), *Mat. Lapok*, v. 22, 1971, pp. 269—282.
[2] ÁDÁM, A., On the question of description of the behaviour of finite automata, *Studia Sci. Math. Hungar.*, v. 13, 1978, pp. 105—124.
[3] ÁDÁM, A., On the complexity of codes and pre-codes assigned to finite Moore automata, *Acta Cybernet.* (Szeged), v. 5, 1981, 117—133.
[4] ÁDÁM, A., Research problem 29 (The connection of the state number and the complexity of finite Moore automata), *Period. Math. Hungar.*, v. 12, 1981, pp. 229—230.
[5] DVOŘÁK, V. & GERBRICH, J. & NOVOTNÝ, J., Homomorphisms of automata, I: Autonomous automata, *Scripta Fac. Sci. Nat. Univ. Purk. Brun.*, v. 12, 1982, pp. 115—128.
[6] GERBRICH, J. & NOVOTNÝ, J., Homomorphisms of automata, II: Sequential machines, *Scripta Fac. Sci. Nat. Univ. Purk. Brun.*, v. 12, 1982, pp. 129—136.
[7] GILL, A. & FLEXER, R., Periodic decomposition of sequential machines, *J. Assoc. Comput. Mach.*, v. 14, 1967, pp. 666—676.
[8] GINSBURG, S., *An introduction to mathematical machine theory*, Addison—Wesley, Reading, 1962.
[9] HWANG, K., Periodic realization of synchronous sequential machines, *IEEE Trans. Computers*, v. 22, 1973, pp. 929—927.
[10] HWANG, K., Cyclic decomposition of finite stochastic systems, *J. Comput. System Sci.*, v. 9, 1974, pp. 56—68.

[11] Kopeček, O., Homomorphisms of partial unary algebras, *Czechoslovak Math. J.*, v. 26, 1976, pp. 108—127.
[12] Machner, J. & Strassner, K., Kongruenzverbände autonomer Semiautomaten, *Elektronische Informationsverarbeitung und Kybernetik*, v. 13, 1977, pp. 137—146.
[13] Ore, O., *Theory of graphs*, Amer. Math. Soc., Providence, 1962.
[14] Yoeli, M. & Ginzburg, A., On homomorphic images of transition graphs, *J. Franklin Inst.*, v. 278, 1964, pp. 291—296.
[15] Егорова, Д. П., Структура конгруэнций унарной алгебры, *Упорядочен. Множества и Решётки* (Саратов), v. 5, 1978, pp. 11—44.
[16] Егорова, Д. П. & Скорняков, Л. А., О структуре конгруэнций унарной алгебры, *Упорядочен. Множества и Решётки* (Саратов), v. 4, 1977. pp. 28—40.
[17] (Скорняков, Л. А.) Skornjakov, L. A., Unars, *Universal Algebra (Colloq. Math. Soc. J. Bolyai, 29, Esztergom, 1977)*, North-Holland, Amsterdam, 1982, pp. 735—743.

Received Febr. 10, 1983.

On injective attributed characterization of 2-way deterministic finite state transducers

By M. BARTHA

Definitions and notation

A 2DFT starting from the left (right) is a 7-tuple $\mathbf{T}=(Q, X, L, R, Y, \delta, q_0)$, where

(i) Q is a finite, nonempty set of states;
(ii) X is a finite, nonempty input alphabet;
(iii) L (left endmarker) and R (right endmarker) are distinguished symbols not in X;
(iv) Y is a finite output alphabet;
(v) $\delta: Q\times(X\cup\{L, R\})\to Q\times Y\times\{left, right\}$ is a partial function;
(vi) $q_0\in Q$ is the initial state.

Informally $\mathbf{T}$ functions as follows. The input word is surrounded by the two endmarkers, and $\mathbf{T}$ starts from state q_0 with its tape head reading the left (right) endmarker. The moves of $\mathbf{T}$ are described by the transition function δ in the usual way (cf. [1]). The transduction terminates successfully when $\mathbf{T}$ moves right of R or left of L. It is obvious that the left or right start of $\mathbf{T}$ is only a technical question. $\mathbf{T}$ is called an 1DFT if δ allows it moving in only one direction.

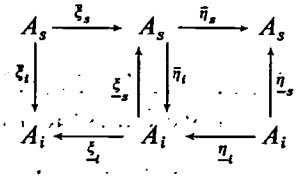
Let A be a finite, nonempty set such that $A=A_s\cup A_i$ and $A_s\cap A_i=\emptyset$. The elements of A_s and A_i are called synthesized attributes (s-attributes) and inherited attributes (i-attributes), respectively. Define the monoid $\mathbf{M}(A, Y)$ (Y is a finite alphabet) as follows. $M(A, Y)$ consists of all partial functions of A into $A\times Y^*$. Disjoining $\xi\in M(A, Y)$ into four parts we can represent it by the following diagram,

$$\xi:\quad \begin{array}{ccc} A_s & \xrightarrow{\bar{\xi}_s} & A_s\times Y^* \\ {\scriptstyle \bar{\xi}_i}\downarrow & & \uparrow{\scriptstyle \underline{\xi}_s} \\ A_i\times Y^* & \xleftarrow[\underline{\xi}_i]{} & A_i \end{array}$$

where $\xi=\bar{\xi}_s\cup\bar{\xi}_i\cup\underline{\xi}_s\cup\underline{\xi}_i$ and $\bar{\xi}_s$, $\bar{\xi}_i$, $\underline{\xi}_s$, $\underline{\xi}_i$ have pairwise disjoint domains. To make this kind of diagrams composable we rather consider ξ as a partial function $\xi': A\times Y^*\to A\times Y^*$, where $\xi'(a, w)$ $(a\in A, w\in Y^*)$ can be obtained from $\xi(a)$

by prefixing its second component with w. For simplicity we use the abusing notation $\xi'=\xi$, and do not indicate the factor Y^* in the diagrams. For $a\in A$, the first and the second component of $\xi(a)$ will be denoted by attr (ξ, a) and out (ξ, a), respectively. attr (ξ) will denote the partial function $\{(a, \text{attr}(\xi, a))|a\in A\}$. If $Y=\emptyset$, then we identify ξ with attr (ξ). ξ is called injective if such is attr (ξ). Now if $\xi, \eta\in M(A, Y)$, then $\xi\circ\eta=\zeta$ can be constructed as follows.

ζ:



$$\bar{\zeta}_s = \bigcup_{n\geqq 0} (\bar{\xi}_s \circ (\bar{\eta}_i \circ \underline{\xi}_s)^n \circ \bar{\eta}_s); \quad \bar{\zeta}_i = \bar{\xi}_i \cup \Big(\bigcup_{n\geqq 0} (\bar{\xi}_s \circ (\bar{\eta}_i \circ \underline{\xi}_s)^n \circ \bar{\eta}_i \circ \underline{\xi}_i)\Big).$$

$$\underline{\zeta}_i = \bigcup_{n\geqq 0} (\underline{\eta}_i \circ (\underline{\xi}_s \circ \bar{\eta}_i)^n \circ \underline{\xi}_i); \quad \underline{\zeta}_s = \underline{\eta}_s \cup \Big(\bigcup_{n\geqq 0} (\underline{\eta}_i \circ (\underline{\xi}_s \circ \bar{\eta}_i)^n \circ \underline{\xi}_s \circ \bar{\eta}_s)\Big).$$

ζ is well defined, since in each case those partial mappings, the union of which must be taken have pairwise disjoint domains. It is easy to verify that this composition is associative, preserves injectivity, and the unit element of $\mathbf{M}(A, Y)$ corresponds to the identity map of $A\times Y^*$. For $a\in A$, path $(\xi\circ\eta, a)$ will denote the sequence of attributes reached in the above composite diagram during the computation of $\xi\circ\eta(a)$.

Definition. A simple deterministic attributed string transducer (SDAST) starting from the left (right) is a 7-tuple $\mathbf{A}=(A, X, L, R, Y, h, a_0)$, where

(i) $A=A_s\cup A_i$ is the finite, nonempty set of attributes, $A_s\cap A_i=\emptyset$;
(ii) X, L, R and Y are as in the case of a 2DFT;
(iii) h is a mapping of $X(L, R)=X\cup\{L, R\}$ into $M(A, Y)$;
(iv) if $\mathbf{A}$ starts from the left, then $a_0\in A_s$, else $a_0\in A_i$.

Denote the extension of h to a homomorphism of $X(L, R)^*$ into $\mathbf{M}(A, Y)$ also by h. Then the transform of $w\in X^*$ by $\mathbf{A}$ is out$(h(LwR), a_0)$. $\mathbf{A}$ is called injective if $h(x)$ is injective for every $x\in X(L, R)$.

Lemma 1. 2DFT and SDAST are equivalent, i.e. they define the same class of mappings.

Proof. Let $\mathbf{T}=(Q, X, L, R, Y, \delta, q_0)$ be a 2DFT, and define the SDAST $\mathbf{A}=(2Q, X, L, R, Y, h, a_0)$ as follows. A_s and A_i are two (disjoint) isomorphic copies of Q. Let q_s and q_i denote the corresponding s-attribute and i-attribute of a state $q\in Q$, respectively. Then for $x\in X(L, R)$ and $q\in Q$, $h(x)(q_s)$ and $h(x)(q_i)$ are defined iff $\delta(q, x)$ is defined, and in this case

$$h(x)(q_s) = h(x)(q_i) = \left(\begin{Bmatrix} q'_s \\ q'_i \end{Bmatrix}, w\right) \quad \text{if} \quad \delta(q, x) = \left(q', w, \begin{Bmatrix} right \\ left \end{Bmatrix}\right).$$

$a_0=(q_0)_s$ if $\mathbf{T}$ starts from the left, otherwise $a_0=(q_0)_i$. It is easy to see that $\mathbf{T}$ and $\mathbf{A}$ are equivalent.

Let $\mathbf{A}=(A, X, L, R, Y, h, a_0)$ be an SDAST and define the 2DFT $\mathbf{T}=(A, X, L, R, Y, \delta, a_0)$ as follows. For $x\in X(L, R)$ and $a\in A$, $\delta(a, x)$ is defined iff $h(x)(a)$ is defined, and in this case

$$\delta(a, x)=\left(b,\ w,\ \begin{Bmatrix}right\\left\end{Bmatrix}\right) \quad \text{if} \quad h(x)(a)=(b, w) \quad \text{with} \quad b\in\begin{Bmatrix}A_s\\A_i\end{Bmatrix}.$$

The equivalence of $\mathbf{T}$ and $\mathbf{A}$ is again evident. Now we prove a lemma similar to Lemma 1 in [2].

Lemma 2. Every SDAST mapping is the composition of two 1DFT mappings and an injective SDAST mapping.

Proof. Let $\mathbf{A}=(A, X, L, R, Y, h, a_0)$ be an SDAST starting from the left, $w\in X^*$, and suppose that $LwR=w_1xw_2$ for some $x\in X(L, R)$, $w_i\in X(L, R)^*$ $(i=1,2)$. The triple $\alpha=(w_1, x, w_2)$ indicates an x-labelled node in LwR. Let $\xi=\mathrm{attr}(h(w_1))$, $\eta=\mathrm{attr}(h(xw_2))$, called the left and right dependency graphs of α, respectively, and define the subsets $A_s^{(u)}$ and $A_i^{(u)}$ of A as:

(i) if $h(LwR)(a_0)$ is undefined, then $A_s^{(u)}=A_i^{(u)}=\emptyset$;

(ii) else $A_s^{(u)}=\bigcup_{n\geqq 0}(\bar{\xi}_s\circ(\bar{\eta}_i\circ\bar{\xi}_s)^n(a_0))$,

$$A_i^{(u)}=\bigcup_{n\geqq 0}(\bar{\xi}_s\circ(\bar{\eta}_i\circ\bar{\xi}_s)^n\circ\bar{\eta}_i(a_0)).$$

$A_s^{(u)}(A_i^{(u)})$ is the set of useful s-attributes (i-attributes) at node α, i.e. only these attributes of α take part in the transduction of w. Our goal is to mark each node of LwR with a set $A_u\subseteq A$ which consists of the useful s-attributes of the node and the useful i-attributes of its right neighbour. (Take $A_i^{(u)}=\emptyset$ at the "right neighbour of (Lw, R, λ)".) This can be achieved by the successive application of two 1DFT as follows. The first 1DFT $\mathbf{T}_1$ starts from the right and marks each node with a pair consisting of the right dependency graph of the node and that of its right neighbour. The set of possible right dependency graphs is finite, so it can be used as the set of states for $\mathbf{T}_1$. The second 1DFT $\mathbf{T}_2$ starts from the left, and at each node first computes the left dependency graph of the node and that of its right neighbour, then from the mark put by $\mathbf{T}_1$ it is able to compute A_u and write it out as a new mark.

Let $X'\subseteq X(L, R)\times P(A)$ denote the alphabet of those marked symbols that can be achieved by the above marking process, and let $\mathbf{A}'=(A, X', L, R, Y, h', a_0)$ be the following SDAST (starting from the left).

(i) $h'(L)$ and $h'(R)$ are equal to the unit element of $\mathbf{M}(A, Y)$;

(ii) if $(x, A_u)\in X'$, then $h'((x, A_u))$ is the restriction of $h(x)$ to A_u.

$\mathbf{A}'$ is injective, because any duplication would imply a circular dependence among the useful attributes, which is impossible. (Note that if $(x, A_u)\in X'$, then there exist $w_1, w_2\in X(L, R)^*$ such that $w_1xw_2=LwR$ for some $w\in X^*$, and the set of useful attributes at the node (w_1, x, w_2) and its right neighbour is A_u.) It is also clear that the composite application of $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{A}'$ defines the same mapping as $\mathbf{A}$. The case of a right start can be treated symmetrically.

Simulation of 1DFT by injective SDAST

Let $\mathbf{T}=(Q, X, L, R, Y, \delta, \bar{q})$ be an 1DFT starting e.g. from the left, $Q=\{q_1, \ldots, q_n\}$. It can be supposed without loss of generality that δ is completely defined on $Q\times X$. We shall use the following attributes to simulate $\mathbf{T}$.

$$A_s^{(n)} = \{\mathbf{s}(i,j) | 1 \leqq i \neq j \leqq n\} \cup \{\mathbf{s}(i) | i \in [n]\}$$

as synthesized attributes, and

$$A_i^{(n)} = \{\mathbf{i}(i,j) | 1 \leqq i \neq j \leqq n\}$$

as inherited ones.

For $A_n=A_s^{(n)}\cup A_i^{(n)}$ let $H\subseteq M(A_n, \emptyset)$ be defined as follows. $\xi\in H$ iff it satisfies the following three conditions.

(i) for every $1\leqq j<k\leqq n$ $\{\xi(\mathbf{i}(j,k)), \xi(\mathbf{i}(k,j))\}=\{\mathbf{s}(j,k), \mathbf{s}(k,j)\}$;

(ii) there exists an $i\in[n]$ such that

a) $\xi(\mathbf{s}(1))=\mathbf{s}(i)$, and

b) $\xi(\mathbf{i}(i,j))=\mathbf{s}(\min(i,j), \max(i,j))$ for every $j\neq i$;

(iii) for every $1\leqq j\neq k\leqq n$ and $i\neq 1$, $\xi(\mathbf{s}(j,k))$ and $\xi(\mathbf{s}(i))$ are undefined.

It is easy to check that the elements of H are injective. We can define an equivalence relation on H as follows. $\xi\equiv\eta$ iff $\xi(\mathbf{s}(1))=\eta(\mathbf{s}(1))$. Let η_i ($i\in[n]$) be an arbitrary representant of the equivalence class characterized by $\eta_i(\mathbf{s}(1))=\mathbf{s}(i)$.

Lemma 3. For any mapping $f: Q\to Q$ there exists an injective $\xi_f\in M(A_n, \emptyset)$ such that

$$f(q_i)=q_j (i,j\in[n]) \quad \text{implies} \quad \eta_i\circ\xi_f\equiv\eta_j. \tag{1}$$

Proof. We follow an induction on n to construct ξ_f. The case $n=1$ is trivial. Let $n=p+1$ for some $p\geqq 1$, and suppose first that f is injective. Then take

$$\xi_f(\mathbf{s}(i)) = \mathbf{s}(j) \quad \text{if} \quad f(q_i)=q_j,$$

$$\xi_f(\mathbf{s}(i,j)) = \mathbf{s}(i',j') \quad \text{if} \quad (f(q_i), f(q_j))=(q_{i'}, q_{j'}),$$

$$\xi_f(\mathbf{i}(i,j)) = \mathbf{i}(i',j') \quad \text{if} \quad (q_i, q_j)=(f(q_{i'}), f(q_{j'})).$$

It is clear that (1) is satisfied this way. If f is not injective, then interchange the subscripts of the states so that $f^{-1}(q_{p+1})=\emptyset$ should hold. Let $g=f|Q\setminus\{q_{p+1}\}$, and construct $\xi_g\in M(A_p, \emptyset)$ to satisfy (1). This goes together with a reordering of $Q\setminus\{q_{p+1}\}$ that we fix from now on. Let $f(q_{p+1})=q_m$ and $g^{-1}(q_m)=\{q_{m_1}, \ldots, q_{m_k}\}$, where $m_i<m_j$ if $1\leqq i<j\leqq k$. We construct ξ_f in two steps.

Step 1. (i) for each $j\in[k]$

a) $\xi_f(\mathbf{s}(m_j))=\mathbf{i}(m_j, p+1)$, $\xi_f(\mathbf{s}(m_j, p+1))=\xi_g(\mathbf{s}(m_j))$,

b) $\xi_f(\mathbf{s}(p+1, m_j))$ is undefined;

(ii) $\xi_f(\mathbf{s}(p+1))=$ *if* $k=0$ *then* $\mathbf{s}(m)$ *else* $\mathbf{i}(p+1, m_k)$;

(iii) for each $j\in[k-1]$, $\xi_f(\mathbf{s}(m_k, m_j))=\mathbf{i}(p+1, m_j)$;

(iv) for any other $a\in\{\mathbf{s}(i) | i\in[p]\}\cup\{\mathbf{s}(i,j) | 1\leqq i\neq j\leqq p$ and $f(q_i)=f(q_j)\}$, $\xi_f(a)=\xi_g(a)$.

It is easy to see that (iii) is in fact not a real modification of ξ_g, because $\xi_g(\mathbf{s}(m_k, m_j))$ is undefined. (i)/b assures the same situation for ξ_f. It is also clear

that the segment of ξ_f defined so far is injective. After describing the first step of the construction we can prove that if $f(q_i)=q_j$, then

$$\eta_i \circ \xi_f(\mathbf{s}(1)) = \mathbf{s}(j). \tag{2}$$

If $j \neq m$, then we only have to observe that $\text{path}(\eta_i \circ \xi_f, \mathbf{s}(1)) = \text{path}(\eta_i \circ \xi_g, \mathbf{s}(1))$. The abusing notation η_i can be used on both sides of this equation provided $\eta_i \in M(A_p, \emptyset)$ on the right hand side is the restriction of $\eta_i \in M(A_{p+1}, \emptyset)$ on the left hand side. Let $j=m$, $q_i \in g^{-1}(q_m)$, $\text{path}(\eta_i \circ \xi_g, \mathbf{s}(1)) = \mathbf{s}(i) \times \alpha$, where α is an appropriate sequence of attributes ending with $\mathbf{s}(m)$. Then, using (i)/a, (2) follows from the equality

$$\text{path}(\eta_i \circ \xi_f, \mathbf{s}(1)) = (\mathbf{s}(i), \mathbf{i}(i, p+1), \mathbf{s}(i, p+1)) \times \alpha.$$

Finally, if $i=p+1$, then for the first sight it seems possible that the last attribute of $\text{path}(\eta_i \circ \xi_f, \mathbf{s}(1))$ is $\mathbf{s}(p+1, r)$, where $q_r \in g^{-1}(q_m)$. (By (i)/b ξ_f is undefined on these attributes.) However, this would imply that the tail of this path should be $(\mathbf{s}(r), \mathbf{i}(r, p+1), \mathbf{s}(p+1, r))$, which is impossible. Thus, the last attribute of the path must be $\mathbf{s}(m)$, which is the only way out of the circle it has entered (i.e. of the set $\{\mathbf{s}(r), \mathbf{s}(r, s), \mathbf{i}(r, s) | r \neq s, \{q_r, q_s\} \subseteq f^{-1}(q_m)\}$).

Step 2. (i) for each $i \in [p]$

$$(\xi_f(\mathbf{i}(i, p+1)), \xi_f(\mathbf{i}(p+1, i))) = (\mathbf{s}(i, p+1), \mathbf{s}(p+1, i));$$

(ii) if $f^{-1}(q_i)=\emptyset$ for some $i \in [p+1]$, then

$$(\xi_f(\mathbf{i}(m, i)), \xi_f(\mathbf{i}(i, m))) = (\mathbf{s}(\min(m, i), \max(m, i)), \mathbf{s}(\max(m, i), \min(m, i)));$$

(iii) if $i \in [p]$, $i \neq m$ and $f^{-1}(q_i) = \{q_{i_1}, \ldots, q_{i_l}\}$ for some $l \geqq 1$, then

a) $\xi_f(\mathbf{i}(m, i)) = \mathbf{i}(i_1, p+1)$,

b) $\xi_f(\mathbf{s}(i_1, p+1)) = \xi_g(\mathbf{i}(m, i))$,

c) $\xi_f(\mathbf{s}(i_k, p+1)) = \mathbf{i}(p+1, i_{k-1})$ if $1 < k \leqq l$,

d) $\xi_f(\mathbf{s}(p+1, i_k)) = \mathbf{i}(i_{k+1}, p+1)$ if $1 \leqq k < l$,

e) $\xi_f(\mathbf{s}(p+1, i_l)) = \mathbf{s}(\min(m, i), \max(m, i))$,

f) $\xi_f(\xi_g^{-1}(\mathbf{s}(\min(m, i), \max(m, i)))) = \mathbf{i}(p+1, i_l)$;

(iv) for any other $a \in \{\mathbf{i}(i, j) | 1 \leqq i \neq j \leqq p\} \cup \{\mathbf{s}(i, j) | 1 \leqq i \neq j \leqq p$ and $f(q_i) \neq f(q_j)\}$, $\xi_f(a) = \xi_g(a)$.

Again, let $i, j \in [p+1]$, $f(q_i)=q_j$. We prove that

a) for every $1 \leqq r \neq s \leqq p+1$ (3)

$$\eta_i \circ \xi_f(\{\mathbf{i}(r, s), \mathbf{i}(s, r)\}) = \{\mathbf{s}(r, s), \mathbf{s}(s, r)\}, \quad \text{and}$$

b) for every $s \neq j$

$$\eta_i \circ \xi_f(\mathbf{i}(j, s)) = \mathbf{s}(\min(j, s), \max(j, s))$$

(3)/a follows from the fact that all the attributes but the last one of $\text{path}(\eta_i \circ \xi_f, \mathbf{i}(r, s))$ are in the set $\{\mathbf{s}(i, j), \mathbf{i}(i, j) | \{f(q_i), f(q_j)\} = \{q_r, q_s\}\}$ and there are only two ways out of this circle which lead to $\mathbf{s}(r, s)$ and $\mathbf{s}(s, r)$. To prove (3)/b we distinguish three cases.

1) $i=p+1$.
   a) $f^{-1}(q_s)=\emptyset$: consider (ii),
   b) $f^{-1}(q_s)=\{q_{s_1}, \ldots, q_{s_l}\}$ for some $l\geqq 1$:

$$\text{path}\,(\eta_i\circ\xi_f, \mathbf{i}(j,s)) = (\mathbf{i}(s_1, p+1), \mathbf{s}(p+1, s_1), \ldots, \mathbf{i}(s_l, p+1), \mathbf{s}(p+1, s_l), \mathbf{s}(\min(m,s), \max(m,s)));$$

2) $i\neq p+1$ and $s\neq m$.
   a) $s=p+1$: consider (i),
   b) $s\neq p+1$: $\text{path}\,(\eta_i\circ\xi_f, \mathbf{i}(j,s))=\text{path}\,(\eta_i\circ\xi_g, \mathbf{i}(j,s))$;
3) $i\neq p+1$ and $s=m$.
Let $f^{-1}(q_j)=\{q_{i_1}, \ldots, q_{i_l}\}$ $(l\geqq 1)$, $i=i_k$ for some $1\leqq k\leqq l$ and

$$\text{path}\,(\eta_i\circ\xi_g, \mathbf{i}(m,j)) = \alpha.$$

Then

$$\text{path}\,(\eta_i\circ\xi_f, \mathbf{i}(m,j)) = \beta\times\alpha,$$

where $\beta=(\mathbf{i}(i, p+1), \ldots, \mathbf{i}(i_r, p+1), \mathbf{s}(i_r, p+1), \ldots, \mathbf{s}(i_1, p+1))$ for some $r\leqq k$. Since the last attribute of α is $\mathbf{s}(\max(m,j), \min(m,j))$, (2)/a implies that $\xi_f(\mathbf{i}(j,m))=\mathbf{s}(\min(m,j), \max(m,j))$. Finally, (2) and (3) imply (1).

It must be noticed, however, that (1) holds only under one particular ordering of Q. Let us fix an arbitrary order, i.e. suppose that $Q=[n]$. Then by steps 1 and 2 we in fact construct $\xi_{f'}$, where $f'=\varrho^{-1}\circ f\circ\varrho$ for some bijection ϱ. Since $f=\varrho\circ f'\circ\varrho^{-1}$, we can take $\xi_f=\xi_\varrho\circ\xi_{f'}\circ\xi_{\varrho^{-1}}$. (Recall that η_i is an arbitrary representant, and the construction of ξ_ϱ and $\xi_{\varrho^{-1}}$ can be carried out directly.)

Now define $h: X(L,R)\to M(A_n, Y)$ as follows. For $x\in X$ consider the mapping $f: [n]\to[n]$ for which $f(i)=j$ if $\delta(i,x)=(j,w)$. Let

(i) $\text{attr}\,(h(x))=\xi_f$;
(ii) for each $i\in[n]$
   $\text{out}\,(h(x), \mathbf{s}(i))=w$ if $\delta(i,x)=(j,w)$;
(iii) for each $1\leqq i\neq j\leqq n$
   $\text{out}\,(h(x), \mathbf{s}(i,j))=\text{out}\,(h(x), \mathbf{i}(i,j))=\lambda$.

Extend h to a homomorphism of X^* into $\mathbf{M}(A_n, Y)$. An easy induction shows that for any $u\in X^*$ $\delta(i,u)=(j,w)$ implies that

a) $\text{attr}\,(\eta_i\circ h(u))\equiv\eta_j$;
b) $\text{out}\,(\eta_i\circ h(u), \mathbf{s}(1))=w$.

Thus, to make $\mathbf{T}$ and the injective SDAST $(A_n, X, L, R, Y, h, a_0)$ equivalent we only have to set:

(i) $a_0=\mathbf{s}(1)$;
(ii) if $\bar{q}=i$ and $\delta(i,L)=(j,w)$, then $h(L)=\eta_j$ with the modification $\text{out}\,(h(L), \mathbf{s}(1))=w$;
(iii) $h(R)(a)$ is defined iff $a=\mathbf{s}(i)$ $(i\in[n])$ and $\delta(i,R)(=(j,w))$ is defined. In this case $h(R)(\mathbf{s}(i))=(\mathbf{s}(i), w)$.

In [3] we proved that injective SDAST mappings are closed under composition. Thus, using Lemmas 1 and 2 we get the following result.

Theorem. SDAST, injective SDAST and 2DFT define the same class of mappings.

Corollary. ([1], [2]). 2DFT mappings are closed under composition. Other results of [1] and [2] concerning 2DFT with regular lookahead (which are called quasideterministic in [2]) and the reverse run of 2DFT can also be derived from this theorem.

Abstract

The result indicated in the title is achieved as a corollary of the following four statements.

1. 2-way deterministic finite state transducers (2DFT) and simple deterministic attributed string transducers (SDAST) are equivalent.
2. Every SDAST mapping is the composition of two 1DFT mappings and an injective SDAST mapping.
3. 1DFT mappings can be defined by injective SDAST.
4. Injective SDAST mappings are closed under composition.

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

References

[1] AHO, A. V. and J. D. ULLMAN, A characterization of two-way deterministic classes of languages, JCSS 4, 1970, pp. 523—538.
[2] CHYTIL, M. P. and V. JÁKL, Serial composition of 2-way finite-state transducers and simple programs on strings, Proceedings of the fourth Colloquium on Automata, Languages and Programming, Turku, 1977, pp. 135—147.
[3] BARTHA, M., Linear deterministic attributed transformations, *Acta Cybernet.*, v. 6, 1983, pp. 125—147.

Received June 23, 1983.

On some extensions of russian parallel context free grammars

By JÜRGEN DASSOW

1. In the last years some authors have studied the effect of mechanisms regulating the derivation process to the generative capacity. The matrix grammars, programmed grammars, random context grammars, and periodically time-variant grammars belong to the most investigated mechanisms. A. Salomaa, O. Mayer, and M. V. Lomkovskaja, and others have given the results in the case of context free grammars (see [11], [5]); concerning L systems investigations have been done by S. H. von Solms, G. Rozenberg, and P. J. A. Reusch (see [12], [8], [6]); the author has considered extension of indian parallel context free grammars ([1]).

In this note we will complete these results and study the generative capacity of extensions of russian parallel grammars introduced by M. K. Levitina [3]. Already Levitina regarded the extension by the matrix mechanism and proved that the associated family of languages coincides with the family of programmed context free grammars. We will show that also the extensions by the mechanisms of programmed grammars, random context grammars, and periodically time variant grammars generate the same family of languages.

2. For sake of completeness, we will recall some definitions shortly. For detailed information see [11], [3]. A *russian parallel context free* grammar is a construct $G=(V_T, V_N, P, S)$ where

i) V_T and V_N are disjoint nonempty finite sets, $V=V_N \cup V_T$,
ii) P is a finite set of pairs (p, i) where $i\in\{1, 2\}$ and p is a production $A\to w$ with $A\in V_N$, $w\in V^+$,
iii) $S\in V_N$.

The derivation $x\Rightarrow y$, $x, y\in V^+$, is defined by

i) $x=x_1Ax_2$, $y=x_1wx_2$, $x_1, x_2\in V^*$,
ii) $(A\to w, 1)\in P$

or by

i) $x=x_1Ax_2Ax_3 \dots x_{n-1}Ax_n$, $y=x_1wx_2wx_3 \dots x_{n-1}wx_n$, $x_1, x_2, \dots, x_n\in(V\setminus\{A\})^*$,
ii) $(A\to w, 2)\in P$.

$\overset{*}{\Rightarrow}$ denotes the reflexive and transitive closure of $\Rightarrow$. The language $L(G)$ generated

by G is defined as

$$L(G) = \{w: S \overset{*}{\Rightarrow} w,\ w \in V_T^*\}.$$

Now we will give some mechanisms regulating the derivation process.

Programmed grammars: Each rule has the form $(l: (A \to w, i), F, S)$ where l is the label of the rule, F and S (the failure field and the success field) are sets of labels. If A occurs in x, we rewrite x (as in a russian parallel grammar), and in the next derivation step we have to apply a rule with a label in S. If A does not occur in x, we apply a rule with a label in F.

Random context grammars: Each rule has the form $((A \to w, i), U, T)$ where U and T are subsets of V_N. The production $(A \to w, i)$ is applicable to x if and only if any symbol of U occurs in x and no symbol of T occurs in x.[1]

Periodically time variant grammars: We associate a subset $\varphi(i)$ of P with an integer $i \geqq 1$ such that, for each k, $\varphi(n+k+j) = \varphi(n+k)$ for some n and j. The rule applied in the i-th step of the derivation has to be choosen from the set $\varphi(i)$.

For these three type of grammars, the generated language is defined as above.

Matrix grammars: A matrix $m_i = [r_{i1}, r_{i2}, \ldots, r_{i,i_k}]$ is an ordered sequence of rules $r_{i,j} \in P$. The application of a matrix m_i to a word x is defined as the application of the rules r_{ij} in the given order. The generated language consists of all words over V_T which can be derived from S by applications of matrices.

We use the following notations:
$\mathscr{F}(PRP)$ — family of programmed russian parallel languages,
$\mathscr{F}(RCRP)$ — family of random context russian parallel languages,
$\mathscr{F}(TVRP)$ — family of russian parallel periodically time-variant languages,
$\mathscr{F}(MRP)$ — family of russian parallel matrix languages.

If $i \neq 1$ $(i=2)$ for all rules $(A \to w, i)$ of P, we get the context free grammars (indian parallel context free grammars, introduced by K. Krithivasan and R. Siromoney) and its associated extensions. We will use the letters CF or IP instead of RP to denote the corresponding family of languages. It is known that

$$\mathscr{F}(PCF) = \mathscr{F}(RCCF) = \mathscr{F}(TVCF) = \mathscr{F}(MCF),$$

$$\mathscr{F}(PIP) = \mathscr{F}(RCIP) = \mathscr{F}(TVIP) = \mathscr{F}(MIP),$$

and

$$\mathscr{F}(XIP) \subset F(XCF) \quad \text{for} \quad X \in \{P, RC, TV, M\}.$$

3. We will prove analogous relations for russian parallel versions, too.

Theorem. $\mathscr{F}(PRP) = \mathscr{F}(RCRP) = \mathscr{F}(TVRP) = \mathscr{F}(MRP) = \mathscr{F}(PCF)$.

Proof i) By definition, $\mathscr{F}(XCF) \subseteq \mathscr{F}(XRP)$ for $X \in \{P, RC, TV, M\}$. Therefore we have to prove $\mathscr{F}(XRP) \subseteq \mathscr{F}(XCF)$ only.

ii) $\mathscr{F}(PRP) \subseteq \mathscr{F}(PCF)$.

[1] This definition is due to Lomkovskaja and differs slightly from van der Walt's definition. The difference has no effect to the generative capacity. By the parallel rewriting (if $i=2$), the above definition is more useful.

Let $L=L(G)$ for the programmed russian parallel grammar $G=(V_T, V_N, P, S)$. We will construct a programmed context free grammar G' which simulates the application of rules $(A\to w, 2)$ by a set of usual context free productions (in the construction we will write only $B\to v$ instead of $(B\to v, 1)$). We put

$$V' = \{A_l:(l:(A\to w, 2), F, S)\in P\},$$

$$P_1 = \{(l:A\to w, F, S):(l:(A\to w, 1), F, S)\in P\},$$

$$P_2 = \{(l:A\to A_l, F, \{l'\}):(l:(A\to w, 2), F, S)\in P\},$$

$$P_3 = (l':A\to A_l, \{l''\}, \{l'\}):(l:(A\to w, 2), F, S)\in P,$$

$$P_4 = \{(l'':A_l\to w, S, \{l''\}):(l:(A\to w, 2), F, S)\in P\},$$

and $G'=(V_T, V_N\cup V', P_1\cup P_2\cup P_3\cup P_4, S)$. Obviously, G' is a programmed context free grammar with $L(G')=L$.

iii) $\mathscr{F}(RCRP)\subseteqq\mathscr{F}(RCCF)$.

Let $L\in\mathscr{F}(RCRP)$ and $L=L(G)$ for some random context russian parallel grammar $G=(V_T, V_N, P, S)$. We introduce new alphabets V_1 and V_2 by $V_i=\{A_i: A\in V_N\}$, $i=1, 2$, and define the homomorphism h on V by $h(A)=A_2$ for $A\in V_N$ and $h(a)=a$ for $a\in V_T$. Further we put

$$P_1 = \{(A\to w, U, T\cup V_1\cup V_2):((A\to w, 1), U, T)\in P\},$$

$$P_2 = \{(A\to A_1, U, T\cup((V_1\cup V_2)\setminus\{A_1\})):((A\to w, 2), U, T)\in P\},$$

$$P_3 = \{(A_1\to h(w), (U\setminus\{A\})\cup\{A_1\}, (T\cup\{A\}\cup V_1)\setminus\{A_1\}):$$
$$:((A\to w, 2), U, T)\in P\},$$

$$P_4 = \{(A_2\to A, \{A_2\}, V_1): A_2\in V_2\},$$

and

$$G' = (V_T, V_N\cup V_1\cup V_2, P_1\cup P_2\cup P_3\cup P_4, S).$$

If the production $(A\to w, 1)$ is applicable to x in G, then the corresponding production of P_1 is applicable in G', and we derive the same word in both grammars. Now let $(A\to w, 2)$ be applicable to x in G and derive the word y. Then $A\to A_1$ is applicable to x in G', and then we have to apply $A\to A_1$ on all occurrences of A in x. Now we can apply only $A_1\to h(w)$, and we have to do this substitution at all occurrences of A_1. Then we have to use the applicable rules of P_4 and we get also the word y. Therefore $L(G)\subseteqq L(G')$. The other inclusion can be proved by analogous arguments. Thus we have constructed a random context (context free) grammar G' with $L=L(G')$.

iv) $\mathscr{F}(TVRP)\subseteqq\mathscr{F}(TVCF)$.

Let $L\in\mathscr{F}(TVRP)$ and $L=L(G)$ for some periodically time-variant russian parallel grammar $G=(V_T, V_N, P, S)$. Let $\varphi(i)$ be the subsets of P such that $\varphi(m)=\varphi(m+j)$ for a certain j and all $m\geqq n$. We will construct a programmed russian parallel grammar G' such that $L=L(G')$, which proves $\mathscr{F}(TVRP)\subseteqq\subseteqq\mathscr{F}(PRP)$. Thus $\mathscr{F}(TVRP)\subseteqq\mathscr{F}(TVCF)$ by ii) and the result concerning the context free case.

Let $V_N=\{A_1, \ldots, A_s\}$. We introduce new alphabets $V_i=\{A_{ki}: 1\leqq k\leqq s\}$ and the homomorphisms h_i on V by

$$h_i(x)=\begin{cases} A_{ki} & \text{if } x=A_k, \\ a & \text{if } x=a\in V_T \end{cases}$$

for $i=1, 2, \ldots, n+j-1$. Further we put

$$\varphi'(i)=\{(A_{ki}\to h_{i+1}(w), r): 1\leqq k\leqq s, (A_k\to w, r)\in\varphi(i)\}$$

for $i=1, 2, \ldots, n+j-2$,

$$\varphi'(n+j-1)=\{(A_{k,n+j-1}\to h_n(w), r): 1\leqq k\leqq s, (A_k\to w, r)\in\varphi(n+j-1)\}.$$

We consider the programmed russian parallel grammar $G'=(V_T \bigcup_{i=1}^{n+j-1} V_i, P', S_1)$ where the elements of P are given in figure 1.

Obviously, $L=L(G')$.

v) $\mathscr{F}(MRP)\subseteqq\mathscr{F}(MCF)$.

This fact follows from Levitina's result $\mathscr{F}(MRP)=\mathscr{F}(PCF)$. (Using the method of iv) we can prove it.)

Corollary 1. For $X\in\{P, RC, TV, M\}$, $\mathscr{F}(XIP)\subsetneqq\mathscr{F}(XRP)$.

Because some of the properties of $\mathscr{F}(PCF)$ are known, for instance $\mathscr{F}(PCF)$ forms an AFL, we get also information on the extensions of russian parallel languages.

4. A language L is called of index k if there exists a grammar G with $L(G)=L$ such that any word $w\in L$ has a derivation with the property that each sentential form of this derivation contains at most k occurrences of letters of V_N. L is of finite index iff there exists an integer k such that L is of index k.

By $\mathscr{F}(X)_{FIN}$ we denote the family of languages of $\mathscr{F}(X)$ which are of finite index.

By the results of [9], [7], [10], and the fact that the construction in [1] and in this note preserve the finiteness of the index of a language, we get a second corollary.

Corollary 2. $\mathscr{F}(PCF)_{FIN}=\mathscr{F}(RCCF)_{FIN}=\mathscr{F}(TVCF)_{FIN}=\mathscr{F}(MCF)_{FIN}=$
$=\mathscr{F}(PIP)_{FIN}=\mathscr{F}(RCIP)_{FIN}=\mathscr{F}(TVIP)_{FIN}=\mathscr{F}(MIP)_{FIN}=$
$=\mathscr{F}(PRP)_{FIN}=\mathscr{F}(RCRP)_{FIN}=\mathscr{F}(TVRP)_{FIN}=\mathscr{F}(MRP)_{FIN}$.

In [9], properties of this language family are studied. For instance, it forms an AFL again.

5. Finally, we remark that the context free languages and the russian parallel context free languages are incomparable with the extensions of indian parallel languages. This follows by the following facts:

— $\{a^nb^nc^n: n\geqq 1\}$ is in $\mathscr{F}(MIP)$, it is not a russian parallel context free language ([3]),

— the extensions of indian parallel context free languages coincide with the

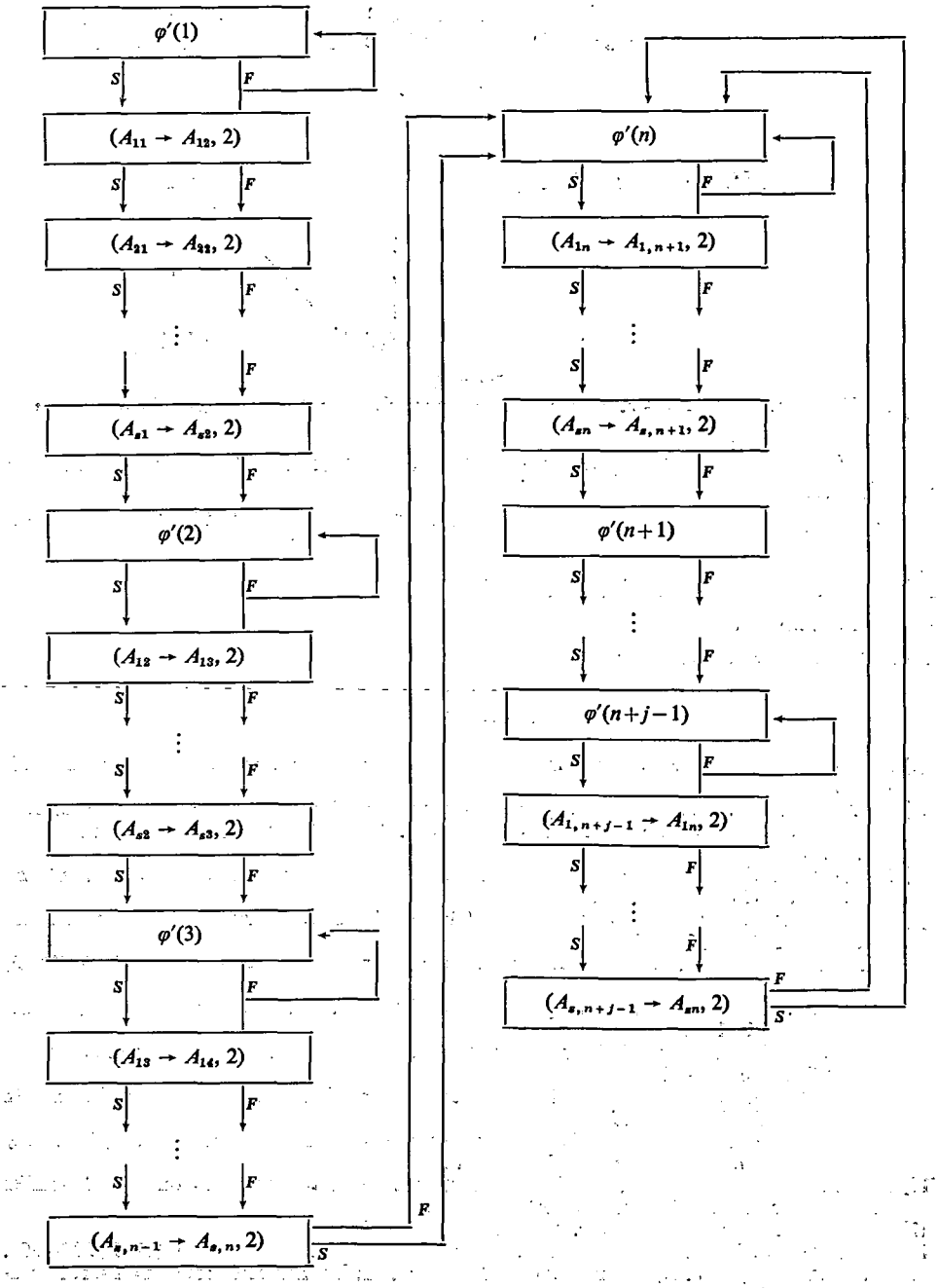


Figure 1

An arrow labelled by S leads to the success field; an arrow labelled by F to the failure field.

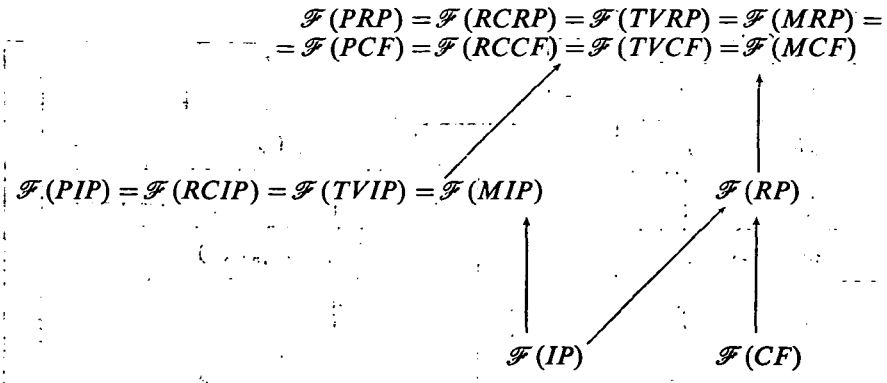


Figure 2

$X \to Y$ denotes $X \subseteq Y$, $X \neq Y$. Language families which are not connected are incomparable.

EDTOL languages ([1]), and there are context free languages which are not *EDTOL* languages.

Thus figure 2 gives the complete relation between the regarded language families.

TECHNOLOGICAL UNIVERSITY OTTO VON GUERICKE
DEPARTMENT OF MATHEMATICS AND PHYSICS
301 MAGDEBURG
BOLESLAW-BIERUT-PL. 5
PSF 124
GERMAN DEMOCRATIC REPUBLIC

References

[1] Dassow, J., On some extensions of indian parallel context free grammars. Acta Cybernetica v. 4. (1980) 303—310.
[2] Ehrenfeucht, A. & Rozenberg, G., On some context free languages that are not deterministic ETOL languages. Dept. Comp. Sc., University of Colorado at Boulder, Technical report No. CU-CS-048-74.
[3] Levitina, M. K., On some grammars with rules of global substitution. Naucno-tehn. inform. Ser. 2, No. 3, (1972), (Russian).
[4] Lomkovskaja, M. V., On some properties of k-conditional grammars. Naucno-tehn. inform. Ser. 2, No. 1 (1972), 16—21 (Russian).
[5] Mayer, O., Some restrictive devices for context free grammars. Inform. Control 20, (1972), 69—92.
[6] Reusch, P. J. A., Regulierte TOL-Systeme I, Gesellsch. f. Math. u. Datenverarbeitung, Bonn, Bericht No. 81 (1974).
[7] Rozenberg, G., More on ETOL systems versus random context grammars. Inform. Processing Letters 5 (1976), 102—106.
[8] Rozenberg, G. & Von Solms, S. H., Some aspects of random context grammars. Informatik Fachberichte 5 (1976), 63—75.
[9] Rozenberg, G. & Vermeir, D., On ETOL systems of finite index. Dept. of Math., University of Antwerp U.I.A., Techn. report No. 75—13.
[10] Rozenberg, G. & Vermeir, D., On the effect of finite index restriction on several families of grammars. Inform. Control 39 (1978) 284—301.
[11] Salomaa, A., Formal Languages. Academic Press, 1973.
[12] Von Solms, S. H., Some notes on ETOL languages. Int. J. Comp. Math. 5, (1975), 285—296.
[13] Van der Walt, A. P. J., Random context languages. In: Information Processing 1971, North-Holland, 1972, 66—68.

Received May 8, 1980.

A new method for the analysis and synthesis of finite Mealy-automata

By Cs. PUSKÁS

In this paper we deal with the problem of analysis and synthesis of finite Mealy-automata. As it is known, this problem has already been solved, namely, it is proved, that if X and Y are non-empty finite sets and $\alpha: X^* \to Y^*$ is an automaton mapping, then there is a finite Mealy-automaton inducing α if and only if all classes of the partition C_α of X^+ corresponding to α are regular languages (see [3]). For a given automaton mapping, a Mealy-automaton can be constructed inducing it and vice versa, but the known algorithms use, as an intermediate step, the notion of the acceptance of languages in automata without outputs and the synthesis algorithms give no reduced automaton, generally. In this paper we give a new proof of the previous theorem, which provides us more advantageus algorithms for both the analysis and the synthesis of finite Mealy-automata. In the latter case, our method supplies immediately the minimal Mealy-automaton inducing a given finite automaton mapping.

Preliminaries

Let X be a finite non-empty set. We shall denote the algebra of all languages over X by $\mathscr{L}(X)$ and the set of all matrices over $\mathscr{L}(X)$ by $M(X)$. A matrix $\mathbf{N} \in M(X)$ is said to be of type $m \times n$ if it has m rows and n columns. The language in the i-th row and in the j-th column of $\mathbf{N}$ will be denoted by $(\mathbf{N})_{ij}$. Based on the regular operations (addition, multiplication and iteration, denoted by $+$, $\cdot$ and $\{\}$, respectively) in $\mathscr{L}(X)$, we introduce the following operations on $M(X)$. If $L \in \mathscr{L}(X)$ and $\mathbf{N} \in M(X)$, then $L \cdot \mathbf{N}$ and $\mathbf{N} \cdot L$ are language matrices, defined by

$$(L \cdot \mathbf{N})_{ij} = L \cdot (\mathbf{N})_{ij} \quad \text{and} \quad (\mathbf{N} \cdot L)_{ij} = (\mathbf{N})_{ij} \cdot L,$$

respectively. Let $\mathbf{N}$ and $\mathbf{P}$ be two language matrices of the same type. Then the sum $\mathbf{N}+\mathbf{P}$ is the language matrix, given by

$$(\mathbf{N}+\mathbf{P})_{ij} = (\mathbf{N})_{ij} + (\mathbf{P})_{ij}.$$

If $\mathbf{N}$ is a language matrix of type $m \times n$ and $\mathbf{P}$ is another one of type $n \times p$, then

we define the product $\mathbf{N} \cdot \mathbf{P}$ in the usual way of matrix products, i.e.,

$$(\mathbf{N} \cdot \mathbf{P})_{ij} = \sum_{k=1}^{n} (\mathbf{N})_{ik} \cdot (\mathbf{L})_{kj}.$$

Using the definition of the product we can form the powers of quadratic matrices as follows: let

$$\mathbf{N}^k = \mathbf{N}^{k-1} \cdot \mathbf{N} \quad (k = 1, 2, \ldots),$$

where $\mathbf{N}^0 = \mathbf{E}$ means the unit language matrix, that is,

$$(\mathbf{E})_{ij} = \begin{cases} e & \text{if} \quad i = j, \\ \emptyset & \text{if} \quad i \neq j. \end{cases}$$

Finally, the iteration $\{\mathbf{N}\}$ of a quadratic matrix $\mathbf{N}$ is defined by

$$\{\mathbf{N}\} = \sum_{k=0}^{\infty} \mathbf{N}^k.$$

We note that we use the term language vector instead of language matrix if it has only one row or only one column. The set of all row language vectors over $\mathscr{L}(X)$ will be denoted by $V^r(X)$ and the set of all column language vectors over $\mathscr{L}(X)$ will be denoted by $V^c(X)$.

Let $\mathbf{N} \in M(X)$ be a quadratic matrix of type $n \times n$. Take a directed graph with n nodes, which are labelled by natural numbers $1, \ldots, n$ and there is an arrow from the node i to the node j if and only if $e \in (\mathbf{N})_{ij}$. This graph is called the *characteristic graph* (see [3]) of the matrix $\mathbf{N}$. If the characteristic graph of $\mathbf{N}$ has cycles and the node i belongs to a cycle, then the number i is said to be a *cyclic number* with respect to $\mathbf{N}$.

Now we consider matrix equations of form

$$\mathbf{N} \cdot \mathbf{Q} + \mathbf{P} = \mathbf{Q}, \tag{1}$$

where $\mathbf{N}$ and $\mathbf{P}$ are given language matrices and $\mathbf{N}$ is of type $n \times n$.

We shall use the following results which are generalizations of some results due to V. G. Bodnarčuk [2] (see also [3, 4, 5, 7]):

Statement 1 [6]. *If the characteristic graph of* $\mathbf{N}$ *has no cycle, then*

$$\mathbf{Q} = \{\mathbf{N}\} \cdot \mathbf{P}$$

is the unique solution of the equation (1). *In the opposite case, every solution of* (1) *has the form*

$$\mathbf{Q} = \{\mathbf{N}\} \cdot (\mathbf{P} + \mathbf{R}),$$

where $\mathbf{R}$ *is an arbitrary language matrix with the same type as* $\mathbf{P}$, *such that if* i $(1 \leqq i \leqq n)$ *is not a cyclic number with respect to* $\mathbf{N}$, *then* $(\mathbf{R})_{ij} = \emptyset$ *for all* j.

Statement 2 [6]. *If the equation* (1) *has a unique solution, then it can be determined by subsequent elimination of unknown rows of the matrix* $\mathbf{Q}$.

Statement 3 [6]. *If every element* $(\mathbf{N})_{ij}$ *and* $(\mathbf{P})_{ij}$ *of the matrices* $\mathbf{N}$ *and* $\mathbf{P}$, *respectively, is regular and the characteristic graph of* $\mathbf{N}$ *has no cycle, then every element* $(\mathbf{Q})_{ij}$ *of the solution matrix* $\mathbf{Q}$ *of the matrix equation* (1) *is regular.*

Connections between language vectors and automaton mappings

It is known, that every automaton mapping $\alpha: X^* \to Y^*$, where X and Y are non-empty finite sets, determines a partition C_α of X^+, which consists of classes

$$L_y = \langle p \in X^+ | \overline{\alpha(p)} = y \rangle \quad (y \in Y).$$

Here $\bar{r}$ denotes the last letter of the non-empty word r. Conversely, every partition C of X^+ defines a unique automaton mapping $\alpha: X^* \to Y^*$ appart from the notation of elements of Y. This fact makes possible for us to establish a one-to-one correspondence between automaton mappings and certain language vectors.

In the following we use the term *l*-vectors instead of row language vectors and they will be denoted by $\mathbf{a}, \mathbf{b}, \mathbf{c}, \ldots$.

An *l*-vector $\mathbf{a} \in V^r(X)$ is said to be *complete* if the sum of its components a_i is the free semigroup X^+ and the intersection of any two components a_i and a_j $(i \neq j)$ of $\mathbf{a}$ is the empty language.

It is obvious that if $X = \langle x_1, \ldots, x_l \rangle$, $Y = \langle y_1, \ldots, y_m \rangle$ and $\alpha: X^* \to Y^*$ is an automaton mapping then we can correspond to α a complete *l*-vector $\mathbf{a}$ of m components, such that

$$a_i = \langle p \in X^+ | \overline{\alpha(p)} = y_i \rangle \quad (i = 1, \ldots, m).$$

Conversely, if $\mathbf{a} \in V^r(X)$ is a complete *l*-vector of m components then it determines an automaton mapping $\alpha: X^* \to Y^*$, such that $\alpha(e) = e$ and for $p = x_{i_1} x_{i_2} \ldots x_{i_k}$, $\alpha(p) = y_{j_1} y_{j_2} \ldots y_{j_k}$ if and only if $x_{i_1} \in a_{j_1}$, $x_{i_1} x_{i_2} \in a_{j_2}$, $\ldots$, $x_{i_1} x_{i_2} \ldots x_{i_k} \in a_{j_k}$.

An *l*-vector $\mathbf{a} \in V^r(X)$ is called *regular* if every component of $\mathbf{a}$ is a regular language.

A system $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ of complete *l*-vectors from $V^r(X)$ is said to be *closed* if there exist functions

$$f: \langle 1, \ldots, n \rangle \times \langle 1, \ldots, l \rangle \to \langle 1, \ldots, n \rangle$$

and

$$\mathbf{b}: \langle 1, \ldots, n \rangle \to V^r(X),$$

such that

$$b(i)_k = \sum_{x_j \in a_{ik}} x_j$$

and

$$\mathbf{a}_i = \sum_{x_j \in X} x_j \mathbf{a}_{f(i,j)} + \mathbf{b}(i) \tag{2}$$

for all i $(=1, \ldots, n)$ holds.

We would like to direct attention to the fact, that a closed complete *l*-vector system can be considered as the rows of a solution language matrix of a matrix equation (1). Indeed, if we set

$$(\mathbf{N})_{ij} = \sum_{f(i,k)=j} x_k \quad (i, j = 1, \ldots, n)$$

and we put $\mathbf{a}_i$ and $\mathbf{b}(i)$ into the i-th row of $\mathbf{Q}$ and $\mathbf{P}$, respectively, then (2) gains the form (1). Therefore, by Statement 3, we have got immediately the following

Lemma 4. *If* $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ *is a closed complete l-vector system then* $\mathbf{a}_1, \ldots, \mathbf{a}_n$ *are regular.*

Theorem 5. *Let* $\mathfrak{A}=(A, X, Y, \delta, \lambda)$ *be a finite Mealy-automaton with state set* $A=\langle a_1, \ldots, a_n\rangle$, *input set* $X=\langle x_1, \ldots, x_l\rangle$, *output set* $Y=\langle y_1, \ldots, y_m\rangle$, *transition function* $\delta: A\times X\to A$ *and output function* $\lambda: A\times X\to Y$. *Let* f *and* $\mathbf{b}$ *be the functions*

$f: \langle 1, \ldots, n\rangle\times\langle 1, \ldots, l\rangle\to\langle 1, \ldots, n\rangle$, *defined by* $\delta(a_i, x_j)=a_{f(i,j)}$ *and*

$$\mathbf{b}: \langle 1, \ldots, n\rangle \to V^r(X), \quad \text{given by} \quad b(i)_k = \sum_{\lambda(a_i, x_j)=y_k} x_j.$$

Then the l-vectors $\mathbf{a}_i\in V^r(X)$ $(i=1, \ldots, n)$, *where*

$$a_{ik} = \langle p\in X^+ | \overline{\lambda(a_i, p)} = y_k\rangle \quad (i=1, \ldots, n;\ k=1, \ldots, m)$$

form a closed complete l-vector system, that is, satisfy the equalities (2).

Proof. It is obvious that $\mathbf{a}_1, \ldots, \mathbf{a}_n$ are complete l-vectors. Thus we have to show that $\mathbf{a}_1, \ldots, \mathbf{a}_n$ satisfy the equalities (2). Let i $(1\leqq i\leqq n)$ and k $(1\leqq k\leqq m)$ be arbitrary index pair. We prove that

$$a_{ik} = \sum_{x_j\in X} x_j a_{f(i,j)k} + b(i)_k.$$

Let p be an arbitrary element of a_{ik}. We distinguish two cases.

Case 1. If $|p|=1$, i.e., $p=x_j$ for some j $(1\leqq j\leqq l)$ then $x_j\in a_{ik}$ implies $\lambda(a_i, x_j)=y_k$. Hence, by definition of $\mathbf{b}$, we have that $x_j\in b(i)_k$ and therefore $p\in \sum_{x_j\in X} x_j a_{f(i,j)k} + b(i)_k$.

Case 2. If $|p|\geqq 2$ then $p=x_j q$ $(1\leqq j\leqq l)$ and

$$y_k = \overline{\lambda(a_i, p)} = \overline{\lambda(a_i, x_j q)} = \overline{\lambda(\delta(a_i, x_j), q)} = \overline{\lambda(a_{f(i,j)}, q)},$$

that is, $q\in a_{f(i,j)k}$ and $p=x_j q\in x_j a_{f(i,j)k}\subseteq \sum_{x_j\in X} x_j a_{f(i,j)k} + b(i)_k$. Conversely, let p be an arbitrary element of $\sum_{x_j\in X} x_j a_{f(i,j)k} + b(i)_k$. If $p\in b(i)_k$, then $|p|=1$ and $\lambda(a_i, p)=y_k$ and therefore $p\in a_{ik}$. If $p\in x_j a_{f(i,j)k}$ for some j $(1\leqq j\leqq l)$, then $p=x_j q$ with $q\in a_{f(i,j)k}$. This implies that

$$y_k = \overline{\lambda(a_{f(i,j)}, p)} = \overline{\lambda(\delta(a_i, x_j), q} = \overline{\lambda(a_i, x_j q)} = \overline{\lambda(a_i, p)},$$

i.e., $p\in a_{ik}$. Thus we have shown that

$$a_{ik} = \sum_{x_j\in X} x_j a_{f(i,j)k} + b(i)_k$$

for all i $(1\leqq i\leqq n)$ and k $(1\leqq k\leqq m)$ holds. □

By Lemma 4 and Theorem 5 we immediately get

Corollary 6. *If* $\mathfrak{A}=(A, X, Y, \delta, \lambda)$ *is a finite Mealy-automaton then for all state* $a\in A$ *and output* $y\in Y$ *the language*

$$a_y = \langle p\in X^+ | \overline{\lambda(a, p)} = y\rangle$$

is regular.

Theorem 5 and Statement 2 provide us an algorithm to the analysis of finite Mealy-automata. To illustrate this, let us consider

Example 1. Let $\mathfrak{A}$ be the Mealy-automaton, given by the transition-output table:

$\mathfrak{A}$	a_1	a_2	a_3
x	(a_2, u)	(a_3, v)	(a_2, u)
y	(a_3, v)	(a_2, w)	(a_3, w)

Taking the ordering $u \prec v \prec w$ of the output letters, by Theorem 5 we have the following l-vector equations:

$$\mathbf{a}_1 = x\mathbf{a}_2 + y\mathbf{a}_3 + [x, y, \emptyset],$$

$$\mathbf{a}_2 = x\mathbf{a}_3 + y\mathbf{a}_2 + [\emptyset, x, y],$$

$$\mathbf{a}_3 = x\mathbf{a}_2 + y\mathbf{a}_3 + [x, \emptyset, y].$$

From the third equation we obtain that

$$\mathbf{a}_3 = \{y\}(x\mathbf{a}_2 + [x, \emptyset, y]).$$

Substituting this into the expressions of $\mathbf{a}_1$ and $\mathbf{a}_2$, we have, that

$$\mathbf{a}_1 = x\mathbf{a}_2 + y\{y\}(x\mathbf{a}_2 + [x, \emptyset, y]) + [x, y, \emptyset] =$$

$$= \{y\}x\mathbf{a}_2 + [\{y\}x, y, y^2\{y\}],$$

$$\mathbf{a}_2 = x\{y\}(x\mathbf{a}_2 + [x, \emptyset, y]) + y\mathbf{a}_2 + [\emptyset, x, y] =$$

$$= (y + x\{y\}x)\mathbf{a}_2 + [x\{y\}x, x, y + x\{y\}y].$$

Now we can already determine the l-vector $\mathbf{a}_2$:

$$\mathbf{a}_2 = \{y + x\{y\}x\}[x\{y\}x, x, y + x\{y\}y] =$$

$$= [\{y + x\{y\}x\}x\{y\}x, \{y + x\{y\}x\}x, \{y + x\{y\}x\}(y + x\{y\}y)]$$

Then

$$\mathbf{a}_1 = [\{y\}x\{y + x\{y\}x\}x\{y\}x, \{y\}x\{y + x\{y\}x\}x, \{y\}x\{y + x\{y\}x\}(y + x\{y\}y)] +$$

$$+ [\{y\}x, y, y^2\{y\}] =$$

$$= [\{y\}x(e + \{y + x\{y\}x\}x\{y\}x), y + \{y\}x\{y + x\{y\}x\}x, y^2\{y\} +$$

$$+ \{y\}x\{y + x\{y\}x\}(y + x\{y\}y)]$$

and

$$\mathbf{a}_3 = \{y\}x\mathbf{a}_2 + [\{y\}x, \emptyset, \{y\}y] =$$

$$= [\{y\}x(e + \{y + x\{y\}x\}x\{y\}x), \{y\}x\{y + x\{y\}x\}x, y\{y\} +$$

$$+ \{y\}x\{y + x\{y\}x\}(y + x\{y\}y)].$$

Now we define a new operation on the set of l-vectors. It is well-known that if L is a language from $\mathscr{L}(X)$ and $p \in X^*$ then the left-side derivation of L with

respect to p is the language ${}_lD_p(L)=\langle q\in X^*\mid pq\in L\rangle$. We modify this concept as follows: by the *left-side e-free derivation* of L with respect to p we mean the language ${}_lD_p^-(L)=\langle q\in X^+\mid pq\in L\rangle$. It is obvious that

$${}_lD_p^-(L)=\begin{cases}{}_lD_p(L) & \text{if } p\notin L,\\ {}_lD_p(L)-\langle e\rangle & \text{if } p\in L.\end{cases}$$

We extend this operation to l-vectors, that is, if $\mathbf{a}=[a_1, \ldots, a_m]$ then we define the left-side e-free derivation of $\mathbf{a}$ with respect to p, by

$${}_lD_p^-(\mathbf{a})=[{}_lD_p^-(a_1), \ldots, {}_lD_p^-(a_m)].$$

Lemma 7. *If* $\mathbf{a}=[a_1, \ldots, a_m]$ *is a complete l-vector then for all* $p\in X^*$, ${}_lD_p^-(\mathbf{a})$ *is a complete l-vector as well.*

Proof. It is easily seen that $a_i\cap a_j=\emptyset$ implies that ${}_lD_p^-(a_i)\cap{}_lD_p^-(a_j)=\emptyset$. On the other hand, if q is an arbitrary element of X^+ then $pq\in X^+$. Consequently, there exist a unique component a_i of $\mathbf{a}$, such that $pq\in a_i$ because of the completeness of $\mathbf{a}$. Hence we obtain that $q\in{}_lD_p^-(a_i)$. Since $e\notin{}_lD_p^-(a_i)$ for all i $(1\leqq i\leqq m)$ holds, it follows that $X^+=\sum_{i=1}^{m}{}_lD_p^-(a_i)$. □

Lemma 8. *If* $\mathbf{a}=[a_1, \ldots, a_m]$ *is an arbitrary l-vector in* $V^r(X)$ *then*

$$\mathbf{a}=\sum_{x_j\in X}x_j{}_lD_{x_j}^-(\mathbf{a})+\mathbf{b},$$

where $\mathbf{b}=[b_1, \ldots, b_m]$ *is an l-vector for which* $b_i=\sum_{x_j\in a_i}x_j$ $(i=1, \ldots, m)$.

Proof. Let a_i $(1\leqq i\leqq m)$ be an arbitrary component of $\mathbf{a}$. By the definition of $\mathbf{b}$ it is trivial that any word of length one from a_i is in b_i and there is no other element of b_i. On the other hand, the word $p\in a_i$, for which $|p|\geqq 2$, is in $x_j{}_lD_{x_j}^-(a_i)$ if and only if the first letter of p is x_j. □

A closed complete l-vector system $\langle\mathbf{a}_1, \ldots, \mathbf{a}_m\rangle$ is said to be *reduced* if $\mathbf{a}_1, \ldots, \mathbf{a}_m$ are pairwise different l-vectors.

Lemma 9. *If* $\mathbf{a}$ *is a complete regular l-vector then there exists a unique reduced closed complete l-vector system containing* $\mathbf{a}$ *and it can be determined algorithmically.*

Proof. If $\mathbf{a}\in V^r(X)$ with $X=\langle x_1, \ldots, x_l\rangle$ then we extend the ordering of X, which is given by the indices of the elements in X onto X^* as follows: for arbitrary pair of words p and q let $p\prec q$ if either $|p|<|q|$ or $|p|=|q|$ and in the latter case p precedes q by the lexicographical ordering. Then we form the left-side e-free derivations of $\mathbf{a}$. Since $\mathbf{a}$ is a regular l-vector, it has only finite different left-side e-free derivations and they are regular as well. Therefore, there exist a system of words $p_1, \ldots, p_n$ in X^*, such that the following conditions hold:

(i) if $i\neq j$ $(1\leqq i, j\leqq n)$ then ${}_lD_{p_i}^-(\mathbf{a})\neq{}_lD_{p_j}^-(\mathbf{a})$,

(ii) for all $q\in X^*$ there exists a unique i $(1\leqq i\leqq n)$, such that ${}_lD_q^-(\mathbf{a})={}_lD_{p_i}^-(\mathbf{a})$,

(iii) if q is an arbitrary word in X^* for which ${}_lD_q^-(\mathbf{a})={}_lD_{p_i}^-(\mathbf{a})$ $(1\leqq i\leqq n)$ then $p_i\prec q$.

Let us assume that the elements of the system $\langle p_1, \ldots, p_n\rangle$ are indexed according to the ordering of X^*, that is, $p_1\prec p_2\prec\ldots\prec p_n$. Then $p_1=e$. Let $\mathbf{a}_i=$

$={}_lD^-_{p_i}(\mathbf{a})$ for all $i(=1, \ldots, n)$. The system $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ consist of pairwise different complete l-vectors. We show that this system is closed as well. To prove this, we have to note that for all $q \in X^*$ and $\mathbf{a}_i \in \langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ there exists p_j $(1 \leqq j \leqq n)$, such that ${}_lD^-_q(\mathbf{a}_i) = {}_lD^-_{p_j}(\mathbf{a}_1)$ because $\mathbf{a}_1 = {}_lD^-_{p_i}(\mathbf{a}) = {}_lD^-_e(\mathbf{a}) = \mathbf{a}$ and ${}_lD^-_q(\mathbf{a}_i) = {}_lD^-_q({}_lD^-_{p_i}(\mathbf{a}_1)) = {}_lD^-_{p_iq}(\mathbf{a}_1)$ and the system $\langle p_1, \ldots, p_n \rangle$ satisfy the condition (ii). We have to determine the functions $f: \langle 1, \ldots, n \rangle \times \langle 1, \ldots, l \rangle \to \langle 1, \ldots, n \rangle$ and $\mathbf{b}: \langle 1, \ldots, n \rangle \to V^r(X)$ yet. Let for all i $(1 \leqq i \leqq n)$ and j $(1 \leqq j \leqq l)$,

$$f(i, j) = k \Leftrightarrow {}_lD^-_{x_j}(\mathbf{a}_i) = {}_lD^-_{p_k}(\mathbf{a}_1) \quad (1 \leqq k \leqq n)$$

and

$$\mathbf{b}(i) = [b(i)_1, \ldots, b(i)_m], \quad \text{where} \quad b(i)_s = \sum_{x_j \in a_{is}} x_j \quad (s = 1, \ldots, m).$$

Finally, the fact that $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ is the unique closed complete l-vector system, which contains the l-vector $\mathbf{a}(=\mathbf{a}_1)$ follows from Lemma 8. □

To illustrate the algorithm described above consider the

Example 2. Let $X = \langle x, y \rangle$ with the ordering $x \prec y$ and take

$$\mathbf{a} = [x\{x\}, y\{y\}, (x\{x\}y + y\{y\}x)\{x+y\}].$$

Let $\mathbf{a}_1 = \mathbf{a}$. Then

$$\mathbf{a}_1 = x[x\{x\}, \emptyset, \{x\}y\{x+y\}] + y[\emptyset, y\{y\}, \{y\}x\{x+y\}] + [x, y, \emptyset].$$

Let $\mathbf{a}_2 = {}_lD^-_x(\mathbf{a}_1) = [x\{x\}, \emptyset, \{x\}y\{x+y\}]$ and $\mathbf{a}_3 = {}_lD^-_y(\mathbf{a}_1) = [\emptyset, y\{y\}, \{y\}x\{x+y\}]$. Then

$$\mathbf{a}_2 = x[x\{x\}, \emptyset, \{x\}y\{x+y\}] + y[\emptyset, \emptyset, (x+y)\{x+y\}] + [x, \emptyset, y].$$

Let $\mathbf{a}_4 = {}_lD^-_y(\mathbf{a}_2) = {}_lD^-_{xy}(\mathbf{a}_1) = [\emptyset, \emptyset, (x+y)\{x+y\}]$.

$$\mathbf{a}_3 = x[\emptyset, \emptyset, (x+y)\{x+y\}] + y[\emptyset, y\{y\}, \{y\}x\{x+y\}] + [\emptyset, y, x]$$

and

$$\mathbf{a}_4 = x[\emptyset, \emptyset, (x+y)\{x+y\}] + y[\emptyset, \emptyset, (x+y)\{x+y\}] + [\emptyset, \emptyset, x+y].$$

It can be seen that ${}_lD^-_{xx}(\mathbf{a}_1) = {}_lD^-_x(\mathbf{a}_1)$, ${}_lD^-_{yy}(\mathbf{a}_1) = {}_lD^-_y(\mathbf{a}_1)$ and ${}_lD^-_{yx}(\mathbf{a}_1) = {}_lD^-_{xyx}(\mathbf{a}_1) = {}_lD^-_{xyy}(\mathbf{a}_1) = {}_lD^-_{xy}(\mathbf{a}_1)$, that is, $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3$ and $\mathbf{a}_4$ are the all different left-side e-free derivations of $\mathbf{a}$. Finally, the functions $f: \langle 1, 2, 3, 4 \rangle \times \langle 1, 2 \rangle \to \langle 1, 2, 3, 4 \rangle$ and $\mathbf{b}: \langle 1, 2, 3, 4 \rangle \to V^r(X)$ are derived from the previous computations: $f(1, 1) = 2$, $f(1, 2) = 3$, $f(2, 1) = 2$, $f(2, 2) = 4$, $f(3, 1) = 4$, $f(3, 2) = 3$ and $f(4, 1) = f(4, 2) = 4$, furthermore $\mathbf{b}(1) = [x, y, \emptyset]$, $\mathbf{b}(2) = [x, \emptyset, y]$, $\mathbf{b}(3) = [\emptyset, y, x]$ and $\mathbf{b}(4) = [\emptyset, \emptyset, x+y]$.

Theorem 10. *Let $X = \langle x_1, \ldots, x_l \rangle$, $Y = \langle y_1, \ldots, y_m \rangle$ and let $\alpha: X^* \to Y^*$ be an automaton mapping. Let $\mathbf{a}_1$ be the complete l-vector corresponding to α. If $\mathbf{a}_1$ is a regular l-vector and $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ is the reduced closed complete l-vector system containing $\mathbf{a}_1$ then α can be induced by the reduced initially connected Mealy-automaton $\mathfrak{A} = (\langle a_1, \ldots, a_n \rangle, a_1, X, Y, \delta, \lambda)$, where $\delta(a_i, x_j) = a_{f(i,j)}$ and $\lambda(a_i, x_j) = y_k$ if and only if $x_j \in b(i)_k$ $(i = 1, \ldots, n; j = 1, \ldots, l; 1 \leqq k \leqq m)$ and the functions f and $\mathbf{b}$ are determined by the system $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$.*

Proof. It is obvious that $\mathfrak{A}$ is well-defined. Since every l-vector $\mathbf{a}_i$ of the system $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ is a left-side e-free derivation of $\mathbf{a}_1$ and $\mathbf{a}_i = {}_iD_p^-(\mathbf{a}_1)$ implies that $a_1 p = a_i$, it follows that $\mathfrak{A}$ is an initially connected Mealy-automaton. We have to prove that $\mathfrak{A}$ induces the mapping α. To verify this, it is sufficient to show that

$$a_{1s} = \langle p \in X^+ | \overline{\lambda(a_1, p)} = y_s \rangle$$

for all $s(=1, \ldots, m)$ holds. Instead of this equality, we prove more, namely that

$$a_{is} = \langle p \in X^+ | \overline{\lambda(a_i, p)} = y_s \rangle \tag{3}$$

for all $i(=1, \ldots, n)$ and $s(=1, \ldots, m)$ holds. If $|p|=1$, that is, $p=x_j$ for some $x_j \in X$ then by the definitions of the functions $\mathbf{b}$ and λ we obtain that

$$x_j \in a_{is} \Leftrightarrow x_j \in b(i)_s \Leftrightarrow \lambda(a_i, x_j) = y_s.$$

Let us assume that (3) have already been proved for all $p \in X^+$ of length less than or equal to r, for all $i\ (=1, \ldots, n)$ and $s\ (=1, \ldots, m)$. Now let $p \in X^+$, such that, $|p|=r+1$. Then $p=x_j q$ for some $x_j \in X$ and $|q|=r$. Thus taking into account that the system $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ is closed and the previous hypothesis, we obtain that

$$p \in a_{is} = \sum_{x_t \in X} x_t a_{f(i,t)s} + b(i)_s \Leftrightarrow q \in a_{f(i,j)s} \Leftrightarrow$$

$$\Leftrightarrow \overline{\lambda(a_{f(i,j)}, q)} = y_s \Leftrightarrow \overline{\lambda(\delta(a_i, x_j), q)} = y_s \Leftrightarrow \overline{\lambda(a_i, p)} = y_s.$$

Therefore, (3) is true. But this means that $\mathbf{a}_i$ is just the l-vector corresponding to the automaton mapping induced by the state a_i of $\mathfrak{A}$ for all $i(=1, \ldots, n)$. Thus, the fact that the system $\langle \mathbf{a}_1, \ldots, \mathbf{a}_n \rangle$ is reduced implies that $\mathfrak{A}$ is reduced as well. □

To show how we can apply this result for the synthesis of finite Mealy-automata consider

Example 3. Let $X=\langle x, y \rangle$, $Y=\langle u, v, w \rangle$ and let $\alpha: X^* \to Y^*$ be the automaton mapping, given by

$$\alpha(e) = e,$$

$$\alpha(x^k) = u^k \quad (k \geqq 1),$$

$$\alpha(y^k) = v^k \quad (k \geqq 1),$$

$$\alpha(x^k y p) = u^k w^{m+1} \quad (k \geqq 1, m = |p|),$$

$$\alpha(y^k x p) = v^k w^{m+1} \quad (k \geqq 1, m = |p|).$$

Then the complete l-vector corresponding to α is just the regular l-vector $\mathbf{a}_1 = [x\{x\}, y\{y\}, (x\{x\}y + y\{y\}x)\{x+y\}]$ from Example 2. Thus the mapping α can be induced by the automaton $\mathfrak{A}=(\langle a_1, a_2, a_3, a_4 \rangle, a_1, X, Y, \delta, \lambda)$, where δ and λ is given by the transition-output table:

$\mathfrak{A}$	a_1	a_2	a_3	a_4
x	(a_2, u)	(a_2, u)	(a_4, w)	(a_4, w)
y	(a_3, v)	(a_4, w)	(a_3, v)	(a_4, w).

Summarizing the results of Theorems 5 and 10, we have got

Corollary 11. *If X and Y are finite non-empty sets and $\alpha: X^* \to Y^*$ is an automaton mapping then it can be induced by a finite Mealy-automaton if and only if the complete l-vector corresponding to α is regular.*

INST. OF MATH. AND COMP. SCI.
K. MARX UNIV. OF ECONOMICS
DIMITROV TÉR 8
1093 BUDAPEST, HUNGARY

References

[1] ARDEN, D. N., Delayed logic and finite state machines, *Theory of Comp. Machine Design. Univ. of Michigan*, Ann Arbor 1960, 1—35.
[2] BODNARČUK, V. G., Системы уравнений в алгебре событий, *Журн. вичисл. матем. и матем. физ.*, 3 (1963), 1077—1088.
[3] GÉCSEG, F. and I. PEÁK, *Algebraic Theory of Automata*, Akadémiai Kiadó, Budapest, 1972.
[4] PEÁK, I., *Bevezetés az automaták elméletébe, II* (Introduction to the Theory of Automata, Vol. II, in Hungarian), Tankönyvkiadó, Budapest, 1978.
[5] PEÁK, I., *Algebra* (in Hungarian), Tankönyvkiadó, Budapest, 1983.
[6] PUSKÁS, Cs., Matrix equations in the algebra of languages with applications to automata, *Papers on Automata Theory, IV*, K. Marx Univ. of Economics, Dept. of Math., Budapest, 1982, No. DM 82-1, 77—89.
[7] SALOMAA, A., *Theory of Automata*, Pergamon Press, New York—London, 1969.
[8] SPIVAK, M. A., К методу анализа абстрактных автоматов с помощью уравнений в алгебре событий, *Кибернетика*, 1965, No. 1, p. 28.

Received May 9, 1983.

Komplexität von Erzeugen in Algebren

H. JÜRGENSEN

1. Das Problem

$\mathfrak{A}=(A, F)$ sei eine endliche Algebra; A ist die endliche Trägermenge von $\mathfrak{A}$, und F ist die endliche Menge von Operationszeichen. Die Abbildung $\nu: F \to \mathbf{N}_0 = \mathbf{N} \cup \{0\}$ ordnet jedem Operationszeichen $f \in F$ seine Stelligkeit ν_f zu. Jedes $f \in F$ definiert eine Operation auf A, die ebenfalls mit f bezeichnet sei, d.h., eine Abbildung $f: A^{\nu_f} \to A$. Für $M \subseteq A$ sei

$$F(M) := \{a \mid a \in A \wedge \exists f \in F \exists a_1, \ldots, a_{\nu_f} \in M: a = f(a_1, \ldots, a_{\nu_f})\},$$

und für $G \subseteq F$ und $N \subseteq A$ sei

$$G(M, N) = \left\{ a \,\middle|\, \begin{array}{l} a \in A \wedge \exists f \in G \exists i, 1 \leqq i \leqq \nu_f \exists a_i \in M \\ \exists a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_{\nu_f} \in N: a = f(a_1, \ldots, a_{\nu_f}) \end{array} \right\}.$$

Ferner seien $\mathfrak{M}, \mathfrak{M}'$ die kleinsten Unteralgebren von $\mathfrak{A}$, die M enhalten und gegen F bzw. gegen $G(\cdot, A)$ abgeschlossen sind. $\mathfrak{M}$ ist also die von M erzeugte Unteralgebra; $\mathfrak{M}'$ könnte man als das von M erzeugte G-Ideal bezeichnen.

In der vorliegenden Arbeit untersuchen wir den Aufwand zur Bestimmung von $\mathfrak{M}$ oder $\mathfrak{M}'$ aus $\mathfrak{A}$, M und G. Als Maschinenmodell für die Realisierung der Algorithmen verwenden wir die RAM (=random access machine); diese Wahl erlaubt es, die Kosten einigermaßen realistisch auch für „relativ" kleine Algebren abzuschätzen, weil keine wesentlichen Kosten für die Adreßrechnungen anfallen.

Um das Problem genau zu stellen, müssen wir noch festlegen, wie $\mathfrak{A}$, M und G dargestellt werden: Die Elemente von A können abstrakt, d.h., etwa als natürliche Zahlen $1, 2, \ldots, n := |A|$ gegeben sein oder konkret, etwa als Transformationen einer endlichen Menge. In vielen Fällen ist die Kenntnis von n und $\mathfrak{A}$ für den Erzeugungsalgorithmus irrelevant, in anderen wird man $\mathfrak{A}$ mit $\mathfrak{M}$ identifizieren können. Wir setzen die Existenz eines Programms zur Berechnung einer Bijektion χ von A auf $[1:n] := \{1, \ldots, n\}$ voraus, dessen Kosten mit konstant h_A angesetzt werden. Die Operation $f \in F$ kann sich auf die χ-Bilder der Argumente oder auf die Argumente selbst beziehen. Für f setzen wir konstant die Kosten c_f an. Als typische Realisierung von χ könnte man an ein hash-Verfahren denken mit im wesentlichen linearen Kosten relativ zur Größe der Darstellung der Elemente. Als Realisierung

für f käme einerseits ein Tabellenzugriff in Frage oder andererseits z.B. ein tatsächliches Rechnen mit den konkreten Elementen, etwa den Transformationen.

Wir wollen Platz- und Zeitaufwand verschiedener Verfahren diskutieren; zu diesem Zweck vereinbaren wir, daß die Kostenfunktion Φ eines Problems jeweils die zwei Varianten Φ^p und Φ^z für die Platz- und die Zeitkomplexität hat.

Verwandte Fragen werden in [3, 4, 6] behandelt. Dabei diskutiert [6] allerdings keine Komplexitätsfragen. Die Aufsätze [3, 4], soweit sie thematisch einschlägig sind, arbeiten mit einem anderen Algorithmusbegriff, der es erlaubt die „Buchführungskosten" zu ignorieren. In [7] wird ein Algorithmus zum Erzeugen von Normalteilern in Gruppen angegeben, dessen Aufwand mit $O(n^2)$ abgeschätzt werden kann, wenn n die Gruppenordnung ist; allerdings werden auch bei dieser Abschätzung alle Buchführungskosten vernachlässigt. Weitere einschlägige Arbeiten findet man in der Bibliographie des Übersichtsartikels [5]. Im allgemeinen werden in diese keine Komplexitätsaussagen gemacht. In [2] schließlich wird gezeigt, daß einige natürliche Probleme für Permutationsgruppen in polynomialer Zeit lösbar sind.

2. Die naiven Algorithmen

Die Trägermenge $\langle M\rangle$ von $\mathfrak{M}$ erhält man mit Hilfe der folgenden trivialen Bemerkung:

2.1. Bemerkung. Sei $M_0 := M$ und $M_{i+1} := M_i \cup F(M_i)$ für $i = 0, 1, \ldots$. Es sei k minimal mit $M_{k+1} = M_k$. Dann ist

$$\langle M\rangle = \bigcup_{i=0}^{k} M_i.$$

Dabei gilt $k \leqq |\langle M\rangle| - |M| \leqq |A| - |M|$.

Sei $m := |M|$, $m_i := |M_i|$, $\tilde{m} := |\langle M\rangle|$, $n := |A|$. Zur Bestimmung von M_{i+1} aus M_i werden $\sum_{f\in F} m_i^{v_f}$ Polynome berechnet. Jedes Ergebnis ist in eine Menge M' mit $M_i \subseteq M' \subseteq M_{i+1}$ einzusortieren. Bei Realisierung der M_i durch Listen kommt man ohne Berücksichtigung der Kosten für die Abbildung χ und die Operationen f mit einem Zeitaufwand

$$O\left(m_{i+1} \sum_{f\in F} m_i^{v_f}\right)$$

für die Bestimmung von M_{i+1} aus M_i aus. Insgesamt ergibt $m_i = O(\tilde{m}) = O(n)$ und $\tilde{m} - m + 1 = O(\tilde{m}) = O(n)$ die (sehr grobe) obere Schranke

$$(\tilde{m} - m + 1)\, O\Big(\sum_{f\in F} \tilde{m}^{v_f+1}\Big) = O\Big(\sum_{f\in F} \tilde{m}^{v_f+2}\Big) = O(\tilde{m}^{\mu+2}) = O(n^{\mu+2})$$

für den Zeitaufwand mit

$$\mu := \max(v_f \mid f \in F).$$

Der Platzaufwand hat ohne Berücksichtigung des zur Definition von $\mathfrak{A}$ erforderlichen Platzes die Größenordnung $O(\tilde{m})$; es wird nur Platz zur Abspeicherung der M_i benötigt, und M_{i+1} kann jeweils als Verlängerung von M_i realisiert werden.

Für die Trägermenge $[M]$ von $\mathfrak{M}'$, gegeben durch $M \subseteq A$ und $G \subseteq F$, hat man die folgende konstruktive Definition:

2.2. Bemerkung. Sei $M'_0 := M$, $M'_{2i+1} := M'_{2i} \cup F(M'_{2i})$, $M'_{2i+2} := M'_{2i+1} \cup G(M'_{2i+1}, A)$ für $i = 0, 1, \ldots$. Es sei k minimal mit $M'_{2k+2} = M'_{2k}$. Dann ist

$$[M] = \bigcup_{i=0}^{2k} M'_i.$$

Dabei gilt $k \leqq |[M]| - |M| \leqq n - m$.

Sei $m'_i := |M'_i|$, $\tilde{m} := |[M]|$. Unter denselben Bedingungen wie oben erhält man als Zeitschranke für die Bestimmung von $[M]$:

$$\sum_{i=0}^{m'-m} \Big(m'_{2i+1} \cdot \sum_{f \in F} m'^{\nu_f}_{2i} + m'_{2i+2} \cdot \sum_{\substack{f \in F \\ \nu_f > 1}} m'_{2i+1} \cdot n^{\nu_f - 1} \cdot \nu_f \Big)$$

$$= (\tilde{m}' - m + 1) O(\tilde{m}'^{\mu+1} + \mu' \tilde{m}'^2 n^{\mu'-1})$$

$$= O(\tilde{m}'^{\mu+2} + \tilde{m}'^3 n^{\mu'-1})$$

$$= O(n^{\mu+2} + n^{\mu'+2}) = O(n^{\mu+2})$$

mit

$$\mu' := \max(\nu_f | f \in G),$$

wobei $\mu' > 1$ vorausgesetzt sei (sonst ist $[M] = \langle M \rangle$). Der Platzaufwand kann wieder durch $O(\tilde{m}')$ abgeschätzt werden.

Für den Speziallfall von Halbgruppen oder Ringen ergeben diese zugegebenermaßen sehr naiven Abschätzungen die folgenden Aussagen über den Zeitaufwand:

Halbgruppen: Erzeugen einer Unterhalbgruppe der Ordnung $\tilde{m}$: $O(\tilde{m}^4)$.
Erzeugen eines Ideals der Ordnung $\tilde{m}'$: $O(\tilde{m}'^3 n)$.
Erzeugen eines einseitigen Ideals der Ordnung $\tilde{m}'$: $O(\tilde{m}'^3 n)$.

Ringe: Erzeugen eines Teilrings der Ordnung $\tilde{m}$: $O(\tilde{m}^4)$.
Erzeugen eines Ideals der Ordnung $\tilde{m}'$: $O(\tilde{m}'^3 n)$.

3. Der Vorteil sorgfältiger Buchführung

Wir wollen jetzt den durch Bemerkung 2.1 gegebenen Algorithmus sorgfältiger studieren; dabei übernehmen wir die oben eingeführte Bezeichnung und definieren zusätzlich

$$Q_{i+1} := F(M_i) \setminus M_i \quad \text{für} \quad i = 0, 1, \ldots.$$

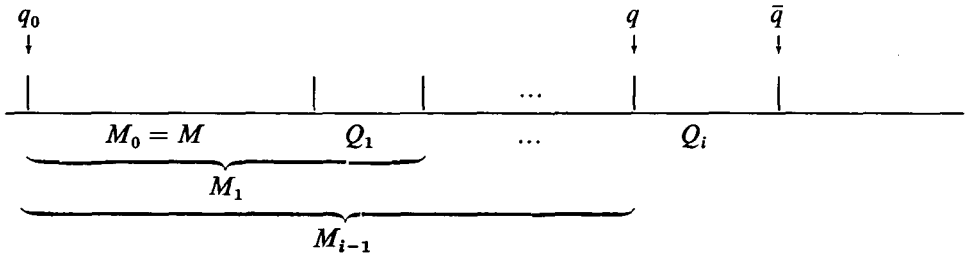
Dann ist offenbar

$$M_{i+1} = M_i \cup F(Q_i, M_i) \quad \text{für} \quad i = 1, 2, \ldots.$$

Zur Bestimmung von $F(Q_i, M_i)$ müssen genau

$$\sum_{f \in F} m_i^{\nu_f} - m_{i-1}^{\nu_f}$$

verschiedene Polynomwerte berechnet werden. Wir wollen wieder voraussetzen, daß die M_i in Listenform gespeichert sind:



Man benötigt vier Zeiger:

q_0 verweist auf den Anfang der Liste, $\bar{q}_0$ hinter das Ende,

q verweist auf den Anfang von Q_i, $\bar{q}$ hinter das Ende.

Zu beginn der Berechnung von M_{i+1} ist $\bar{q}_0 = \bar{q}$. Mit jedem neuen Element wird $\bar{q}_0$ „nach hinten" verschoben. Der Algorithmus endet, wenn nach der Berechnung von M_{i+1} gilt: $\bar{q} = \bar{q}_0$. Andernfalls setzt man $q := \bar{q}$, $\bar{q} := \bar{q}_0$ und fährt mit der Berechnung von M_{i+2} fort.

Sei nun $f \in F$ und $v_f > 0$. Zur Bestimmung der Argumente $(a_1, \ldots, a_{v_f}) \in M_i^{v_f} \setminus M_{i-1}^{v_f}$ benutze man z.B. v_f Laufvariable $p_1, \ldots, p_{v_f}$ mit

$$q \leqq p_1 < \bar{q} \quad \text{und} \quad q_0 \leqq p_j < \bar{q} \quad \text{für} \quad j > 1.$$

Zu jedem derartigen v_f-tupel $(p_1, \ldots, p_{v_f})$ betrachte man ferner sämtliche Permutationen der Form

$$(p_j, p_2, p_3, \cdots, p_{j-1}, p_1, p_{j+1}, \cdots, p_{v_f}),$$

für die $p_j < q$ ist. Bei sorgfältiger Buchführung über die p_j mit $p_j < q$ (z.B. durch gekettete Speicherung dieser p_j) kann man diese Permutationen insgesamt mit dem Aufwand $O(t+1)$ bestimmen, wobei t die Anzahl dieser p_j ist. Der gesamte Zeitaufwand für alle v_f-tupel ist daher proportional der Anzahl der betrachteten v_f-tupel, und diese ist

$$q_i \cdot \sum_{j=0}^{v_f-1} m_{i-1}^{v_f-j-1} \cdot q_i^j \cdot \binom{v_f-1}{j} \cdot (v_f - j) \leqq \frac{(v_f+1)^2}{4 v_f} (m_i^{v_f} - m_{i-1}^{v_f}) =$$

$$= O\big(v_f (m_i^{v_f} - m_{i-1}^{v_f})\big).$$

Der Platzaufwand hat die Größenordnung $O(v_f)$ für die Laufvariablen — diesen Aufwand hatten wir in den „naiven" Abschätzungen vernachlässigt.

Für jedes v_f-tupel $(a_1, \ldots, a_{v_f})$ sind die folgenden Operationen auszuführen:

(1) Berechnung von $a := f(a_1, \ldots, a_{v_f})$.

(2) Prüfen, ob a im bisher aufgebauten Teil von M_{i+1} liegt, und, wenn nein, a in Q_{i+1} einfügen.

Für (1) treten jedesmal die Kosten c_f auf. Die Aufgabe (2) läßt sich durch Suchen in einer zu m_{i+1} proportionalen Zeit erledigen. Eine Verbesserug ist nur bei Abänderung der Speicherungsmethode für die M_i zu erzielen. Diese Möglichkeit soll später weiter verfolgt werden.

In der bisherigen Version haben wir also die Zeitschranke

$$\Phi^z := \sum_{f\in F} m_0^{\nu_f}\cdot c_f\cdot m_1 + \sum_{i=1}^{\tilde{m}-m+1}\sum_{\substack{f\in F\\ \nu_f>0}} \frac{(\nu_f+1)^2}{4\nu_f}\cdot (m_i^{\nu_f}-m_{i-1}^{\nu_f})\cdot c_f\cdot m_{i+1}$$

und die Platzschranke

$$\Phi^p := O(\tilde{m}+\mu).$$

Φ^z wird maximal für $m_1=m_2=\ldots=\tilde{m}$, d.h., $m_1-m_0=\tilde{m}-m$. Dies entspricht der Situation, daß man zunächst alle Elemente von $\langle M\rangle\setminus M$ erhält und danach nur noch kostspielige Überprüfungen gemäß (2) mit negativem Ergebnis macht. Dies ergibt

$$\Phi^z \leqq \sum_{f\in F} m^{\nu_f}\cdot\tilde{m}\cdot c_f + \sum_{\substack{f\in F\\ \nu_f>0}} \frac{(\nu_f+1)^2}{4\nu_f}\cdot(\tilde{m}^{\nu_f}-m^{\nu_f})\cdot\tilde{m}\cdot c_f$$

$$\leqq \sum_{\substack{f\in F\\ \nu_f>0}} \frac{(\nu_f+1)^2}{4\nu_f}\cdot\tilde{m}^{\nu_f+1}\cdot c_f + \sum_{\substack{f\in F\\ \nu_f=0}}\tilde{m}\cdot c_f =$$

$$= \begin{cases} O(\mu\cdot\tilde{m}^{\mu+1}\cdot c), & \text{falls } \mu>0,\\ O(\tilde{m}\cdot c), & \text{falls } \mu=0,\end{cases}$$

mit $c:=\max(c_f|f\in F)$. Für festes μ und c hat man also $\Phi^z=O(\tilde{m}^{\mu+1})=O(n^{\mu+1})$. Diese Schranke erhält man — etwa für $\mu=2$ — auch aus dem trivialen Erreichbarkeitsalgorithmus für gerichtete Graphen (vgl. [1], S. 207).

Wie angekündigt, wollen wir jetzt die Speicherungsmethode ändern. Ein einfacher Übergang auf hash-Tabellen oder boolesche Vektoren, der die Vereinigungsoperation (2) erheblich beschleunigen würde, ist nicht zu empfehlen, weil mit dieser Beschleunigung eine Verlangsamung der Bestimmung der ν_f-tupel eingehandelt würde. Man beachte, daß der Zugriff auf die M_i gleichzeitig sequentiell und indiziert möglich sein sollte. Zwei Lösungswege mit unterschiedlichen Auswirkungen hinsichtlich der Kosten liegen nahe: Man kann neben dem Listenspeicher für die M_i vorsehen

(a) ein Feld der Größe $O(n)$ zur Markierung der in M_i vorhandenen Elemente von A (etwa als hash-Tabelle) oder

(b) eine zusätzliche Abspeicherung der M_i in Form balancierter Bäume.

In beiden Fällen wird zu jedem Element ein zusätzlicher Rechenaufwand zur Indexberechnung bei (a) oder für Vergleiche bei (b) nötig, dessen Zeitbedarf durch h_A abgeschätzt werden kann. Im Falle (a) sind je Element $O(1)$ Zugriffe ausreichend; im Falle (b) muß man für Suchen, Einfügen und Restrukturieren $O(\log m_i)$ Schritte ansetzen. Für (a) erhält man somit die Zeitschranke

$$\Psi^z := \sum_{f\in F} m_0^{\nu_f}\cdot c_f\cdot h_A + \sum_{i=1}^{\tilde{m}-m+1}\sum_{\substack{f\in F\\ \nu_f>0}} \frac{(\nu_f+1)^2}{4\nu_f}\cdot(m_i^{\nu_f}-m_{i-1}^{\nu_f})\cdot c_f\cdot h_A + O(n) =$$

$$= O(\tilde{m}^{\mu}\cdot c\cdot h_A + n)$$

und die Platzschranke

$$\Psi^p := O(n+\tilde{m}+\mu).$$

Der zusätzliche Zeitaufwand $O(n)$ ist erforderlich, um die Anfangsbesetzung der hash-Tabelle zu organisieren. Dies spart man bei (b). Dort hat man die Kosten

$$\Omega^z := \sum_{f\in F} m_0^{\nu_f}\cdot c_f\cdot h_A\cdot\log m_1+\sum_{i=1}^{\tilde{m}-m+1}\sum_{\substack{f\in F\\ \nu_f>0}}\frac{(\nu_f+1)^2}{4\nu_f}\cdot(m_i^{\nu_f}-m_{i-1}^{\nu_f})\cdot c_f\cdot h_A\cdot\log m_{i+1}=$$

$$= O(\tilde{m}^\mu\cdot c\cdot h_A\cdot\log\tilde{m})$$

und

$$\Omega^p := O(\tilde{m}+\mu).$$

Zusammenfassend erhalten wir:

3.1. Satz. Die Erzeugung von $\mathfrak{M}$ ist möglich mit dem Aufwand

$$\Psi^z = O(\tilde{m}^\mu+n),\quad \Psi^p = O(n)$$

beziehungsweise mit dem Aufwand

$$\Omega^z = O(\tilde{m}^\mu\cdot\log\tilde{m}),\quad \Omega^p = O(\tilde{m}).$$

Für die Erzeugung von $\mathfrak{M}'$ aus $M\subseteq A$ und $G\subseteq F$ kann man ähnlich vorgehen. Sei

$$Q'_{2i+1} := F(M'_{2i})\setminus M'_{2i},$$

$$Q'_{2i+2} := G(M'_{2i+1}, A)\setminus M'_{2i+1}.$$

Dann ist also

$$M'_{2i+1} = M'_{2i}\cup F(Q'_{2i}\cup Q'_{2i-1}, M'_{2i})$$

und

$$M'_{2i+2} = M'_{2i+1}\cup G(Q'_{2i+1}\cup Q'_{2i}, A).$$

Wie oben schätzt man den Zeitaufwand und den Platzaufwand ab und erhält:

3.2. Satz. Die Erzeugung von $\mathfrak{M}'$ ist möglich mit dem Aufwand

$$\Psi'^z = O(\tilde{m}'^\mu+\tilde{m}'n^{\mu'-1}+n),\quad \Psi'^p = O(n)$$

beziehungsweise mit dem Aufwand

$$\Omega'^z = O(\tilde{m}'^\mu\log\tilde{m}'+\tilde{m}'n^{\mu'-1}\log\tilde{m}'),\quad \Omega'^p = O(\tilde{m}').$$

Es ist klar, daß die Schranken Ψ^z und Ψ'^z der Größenordnung nach optimal sind, weil ohne genauere Kenntnis von $\mathfrak{A}$ jeweils sämtliche Operationen f auf sämtliche mögliche ν_f-tupel angewendet werden müssen.

An dieser Stelle ist anzumerken, daß die obigen Aussagen stillschweigend voraussetzen, daß $|F|=O(1)$ ist für die einmal gewählte Klasse von Algebren. Es ist offensichtlich, welche Verallgemeinerungen für den Fall, daß $|F|\neq O(1)$ ist, durchzuführen sind. Wir beschränken uns darauf, für diese Situation ein repräsentatives Beispiel anzugeben: $\mathfrak{A}$ sei eine Gruppe; um auch die Normalteilereigenschaften ausdrücken zu können, wählen wir

$$F = \{\cdot, {}^{-1}\}\cup\{\gamma_a | a\in A\}$$

mit

$$v_\cdot = 2, \quad v_{-1} = 1, \quad v_{\gamma_a} = 1$$

und

$$\gamma_a(b) = a^{-1} \cdot b \cdot a.$$

Die Unteralgebren von $\mathfrak{A}$ sind gerade die Normalteiler, und geeignete Modifikation von Satz 3.1 ergibt die Aufwandsschranken

$$\Psi^z = O(n\tilde{m}+\tilde{m}^2+n), \quad \Psi^p = O(n)$$

beziehungsweise

$$\Omega^z = O(n\tilde{m}\log\tilde{m}+\tilde{m}^2\log\tilde{m}), \quad \Omega^p = O(\tilde{m}).$$

Die entsprechenden Schranken ohne Berücksichtigung der Buchführungskosten findet man in [7].

4. Kosten der Uniformisierung

Der Ablauf der im vorigen Abschnitt vorgeführten Algorithmen ist in hohem Maße von den Eingabedaten M und gegebenenfalls G abhängig. In diesem Abschnitt untersuchen wir an den Spezialfällen endlicher Halbgruppen und endlicher Ringe die Kosten eines von den Eingabedaten unabhängigen Erzeugungsverfahrens.

Für die angesprochenen Spezialfälle ergeben sich aus 3.1 und 3.2 die folgenden Schranken:

Halbgruppen: $\Psi^z = O(\tilde{m}^2+n) = O(n^2)$, $\Psi^p = O(n)$;

$$\Psi'^z = O(\tilde{m}'n) = O(n^2), \quad \Psi'^p = O(n) \quad \text{mit} \quad G = F = \{\cdot\}.$$

Ringe: Wie Halbgruppen, mit $G = \{\cdot\}$.

Wir werden zeigen, wie sich die Erzeugungsaufgaben in Halbgruppen und Ringen auf die Multiplikation von $n \times n$-Matrizen über dem booleschen Halbring **B** zurückführen lassen.

Sei also jetzt $\mathfrak{A}$ eine endliche Halbgruppe. Zu $a \in A$ sei

$$\Lambda_a = (\lambda^a_{b,c})$$

die durch die innere Linkstranslation von a definierte $n \times n$-Matrix über **B**, d.h.,

$$\lambda^a_{b,c} = \begin{cases} 1, & \text{falls} \quad ab = c, \\ 0, & \text{falls} \quad ab \neq c. \end{cases}$$

Für $M \subseteq A$ sei

$$\Lambda_M := \sum_{a \in M} \Lambda_a.$$

Sei weiter Λ^0_M die $n \times n$-Einheitsmatrix über **B** und

$$\Lambda^*_M := \sum_{j=0}^{\infty} \Lambda^j_M.$$

Man beweist leicht, daß Λ^*_M an der Stelle (b, c) genau dann eine 1 hat, wenn $b = c$ ist oder wenn $c = a_1 a_2 \ldots a_j b$ für geeignete $j \in \mathbf{N}$ und $a_1, \ldots, a_j \in M$ ist.

Sei nun π_M der n-komponentige Zeilenvektor über $\mathbf{B}$, dessen b-te Komponente genau dann 1 ist, wenn $b \in M$ ist. Damit gilt:

4.1. Lemma. $c \in \langle M \rangle$ genau dann, wenn die c-te Komponente von $\pi_M \Lambda_M^*$ gleich 1 ist.

Die zur Berechnung von Λ_M^* erforderliche Zeit hat bekanntlich dieselbe Größenordnung wie die zur Multiplikation zweier $n \times n$-Matrizen über $\mathbf{B}$ erforderliche Zeit $\mathrm{Mult}^z(n, \mathbf{B})$. Der Zeitaufwand für die Herstellung von Λ_M ist durch $O(n^2)$ beschränkt. Damit folgt:

4.2. Satz. Die Unterhalbgruppe $\mathfrak{M}$ von $\mathfrak{A}$ kann mit einem durch $O(\mathrm{Mult}^z(n, \mathbf{B}))$ beschränkten Zeitaufwand (uniform) erzeugt werden.

Natürlich hat auch der Platzaufwand für die Erzeugung von $\mathfrak{M}$ die Größenordnung des für die Berechnung von Λ_M^*, d.h., für die Berechnung der transitiven Hülle einer binären Relation erforderlichen Platzaufwandes.

Für die Idealerzeugung geht man ein wenig anders vor:

4.3. Satz. Die Berechnung des von M erzeugten Linksideals (Rechtsideals, Ideals) der Halbgruppe $\mathfrak{A}$ kann (uniform) in der Zeit $O(n^2)$ durchgeführt werden.

Zum Beweis hat man nur zu beachten, daß c genau dann im von M erzeugten Linksideal liegt, wenn die c-te Komponente von $\pi_M \Lambda_A^*$ gleich 1 ist. Zur Berechnung von Λ_A^* ist jedoch keine Multiplikation, sondern nur die Addition

$$\Lambda_A^* = \Lambda_A + \Lambda_A^0$$

erforderlich. Mit der dualen Aussage erhält man auch die Behauptung für Rechtsideale und Ideale.

Es ist ohne Schwierigkeiten möglich, diese Überlegungen auf Ringe $\mathfrak{A}$ zu übertragen. Neben der Assoziativität für die beiden Operationen $+$ und $\cdot$ nutzt man dabei nur noch die Distributivitätsgesetze

$$a \cdot (b+c) = a \cdot b + a \cdot c, \quad (a+b) \cdot c = a \cdot c + b \cdot c$$

aus. Den durch $M \subseteq A$ bezüglich $+$ und $\cdot$ bewirkten Linkstranslationen entsprechen die Matrizen $\Lambda_{+,M}$ und $\Lambda_{\cdot,M}$. Man beweist leicht, daß $c \in \langle M \rangle$ genau dann gilt, wenn die c-te Komponente von $\pi_{M_1} \Lambda_{+,M_1}^*$ gleich 1 ist, wobei M_1 durch $\pi_{M_1} = \pi_M \Lambda_{\cdot,M}^*$ definiert ist. Damit erhält man aus Satz 4.2:

4.4. Korollar. Der Teilring $\mathfrak{M}$ von $\mathfrak{A}$ kann mit einem durch $O(\mathrm{Mult}^z(n, \mathbf{B}))$ beschränkten Zeitaufwand (uniform) erzeugt werden.

Für die Erzeugung von Idealen tritt, weil im allgemeinen auf die Erzeugung der durch M erzeugten Unterhalbgruppe bezüglich $+$ nicht verzichtet werden kann, ebenfalls der Aufwand $O(\mathrm{Mult}^z(n, \mathbf{B}))$ auf.

Ein uniformes Erzeugungsverfahren für beliebige universelle Algebren wird schon vom Ansatz her erheblich komplizierter als für die Spezialfälle von Halbgruppen und Ringen. Man beachte insbesondere, daß die so bequeme Darstellung durch Matrizen über $\mathbf{B}$ im allgemeinen wenig Vorteile bringen dürfte; in den Beispielen von Halbgruppen und Ringen ist durch die Matrixdarstellung etwas zu gewinnen, weil der Assoziativität in der Algebra die Assoziativität der Matrixmultiplikation korrespondiert.

Abstract

Time and space complexity of algorithms for generating algebras is studied; the bounds derived are essentially optimal.

DEPARTMENT OF COMPUTER SCIENCE
THE UNIVERSITY OF WESTERN ONTARIO
LONDON, ONTARIO
CANADA N6A 5B7

Literatur

[1] A. V. AHO, J. E. HOPCROFT, J. D. ULLMAN: The Design and Analysis of Computer Algorithms. Addison-Wesley Publ. Co., Reading, 1975.
[2] M. FURST, J. HOPCROFT, E. LUKS: Polynomial-Time Algorithms for Permutation Groups Proc. 21st FOCS Symp., 1980, 36—41.
[3] A. GORALČÍKOVÁ, P. GORALČÍK, V. KOUBEK: Testing Properties of Finite Algebras. Proc. ICALP 1980. Lecture Notes in Computer Science *85*, Springer-Verlag, Berlin, 1980, 273—281.
[4] P. GORALČÍK, A. GORALČÍKOVÁ, V. KOUBEK, V. RÖDL: Fast Recognition of Rings and Lattices. Proc. FCT 1981. Springer Lecture Notes in Computer Science *117*. Springer-Verlag, Berlin, 1981, 137—145.
[5] H. JÜRGENSEN: Computers in Semigroups. Semigroup Forum *15* (1977/78), 1—20.
[6] T. KREID: The Determination of Subalgebras of a Given Algebra. Demonstratio Mathemat. *8* (1975), 269—279.
[7] R. E. Tarjan: Determining whether a Groupoid is a Group. Information Proc. Letters *1* (1972), 120—124.

Received May 9, 1983.

On the lattice of clones acting bicentrally

By László Szabó

1. Introduction

For a set F of operations on a set A the centralizer F^* of F is the set of operations on A commuting with every member of F. If $F=F^{**}$ then we say that F acts bicentrally. The sets of operations on A acting bicentrally forms a complete lattice $\mathscr{L}_A$ with respect to $\subseteq$.

The sets of operations acting bicentrally were characterized in [5] and [11]. For $|A|=3$ the lattice $\mathscr{L}_A$ is completely described in [2] and [3]. The aim of this paper is to investigate the lattice $\mathscr{L}_A$. Among others we show that for any set A there exists a single operation f such that $\{f\}^{**}$ is the set of all operations of A (Theorem 5). Furthermore, it is proved that if $B\subseteq A$ then $\mathscr{L}_B$ can be embedded into $\mathscr{L}_A$ (Corollary 7).

2. Preliminaries

Let A be an at least two element set which will be fixed in the sequel. The set of n-ary operations on A will be denoted by $O_A^{(n)}$ $(n\geq 1)$. Furthermore, we set $O_A=\bigcup_{n=1}^{\infty} O_A^{(n)}$. A set $F\subseteq O_A$ is said to be a *clone* if it contains all projections and is closed with respect to superpositions of operations. Denote by $[F]$ the clone generated by F. Let f and g be operations of arites n and m, respectively. If M is an $m\times n$ matrix of elements of A, we can apply f to each row of M to obtain a column vector consisting of m elements, which will be denoted by $f(M)$. Similarly, we can apply g to each column of M to obtain a row vector of n elements, which will be denoted by $(M)g$. We say that f and g *commute* if for every $m\times n$ matrix M over A, we have $(f(M))g=f((M)g)$.

By the *centralizer* of a set $F\subseteq O_A$ we mean the set $F^*\subseteq O_A$ consisting of all operations on A that commute with every member of F. It can be shown by a simple computation that $F^*=[F]^*=[F^*]$ for every $F\subseteq O_A$. The mapping $F\to F^*$ defines a Galois-connection between the subsets of O_A. Indeed, $F_1\subseteq F_2$ implies $F_1^*\supseteq F_2^*$ and $F\subseteq(F^*)^*=F^{**}$ for every $F_1, F_2, F\subseteq O_A$. From this

it follows that $F^*=F^{***}$ for every $F\subseteq O_A$. Thus the mapping $F\to F^{**}$ is a closure operator on the subsets of O_A. The set F^{**} is called the bicentralizer of F. If $F=F^{**}$ then we say that F acts bicentrally. The sets of operations on A acting bicentrally form a complete lattice with respect to $\subseteq$. Denote by $\mathscr{L}_A$ this lattice. In $\mathscr{L}_A$ we have $\bigwedge_{i\in I} F_i=\bigcap_{i\in I} F_i$, $\bigvee_{i\in I} F_i=\left(\bigcup_{i\in I} F_i\right)^{**}$ and $\left(\bigvee_{i\in I} F_i\right)^*=\bigwedge_{i\in I} F_i^*$, $\left(\bigwedge_{i\in I} F_i\right)^*=\bigvee_{i\in I} F_i^*$. It follows that the mapping $F\to F^*(F\in\mathscr{L}_A)$ is a dual automorphism of $\mathscr{L}_A$.

The set of all projections, and the set of all injective unary operations on A will be denoted by P_A and S_A, respectively. An operation $f\in F$ is said to be homogeneous if $f\in S_A^*$. The symbol H_A denotes the set of all homogeneous operations, i.e., $H_A=S_A^*$.

We say that an operation $f\in O_A$ is parametrically expressible or generated by a set $F\subseteq O_A$ if the predicate $f(x_1, \ldots, x_n)=y$ is equivalent to a predicate of the form

$$(\exists t_1)\ldots(\exists t_l)((A_1=B_1)\wedge\ldots\wedge(A_m=B_m))$$

where A_i and B_i contain only operation symbols from F, variables $x_1, \ldots, x_n$, $y, t_1, \ldots, t_l$, commas and round brackets.

For $3\leq n\leq|A|$ denote by l_n the n-ary near-projection, i.e. the n-ary operation defined as follows:

$$l_n(x_1, \ldots, x_n)=\begin{cases}x_1 & \text{if } x_i\neq x_j,\quad 1\leq i<j\leq n,\\ x_n & \text{otherwise.}\end{cases}$$

We need the ternary dual discriminator-function d which is defined in the following way:

$$d(x, y, z)=\begin{cases}x & \text{if } y\neq z,\\ z & \text{if } y=z.\end{cases}$$

If $f\in O_A$ and $B\subseteq A$ then f_B denotes the restriction of f to B.

3. Results

First we give two examples. For every subset $X\subseteq A$ let C_X be the set of all unary constant operations with value belonging to X. Furthermore, let I_X be the set of all operations $f\in O_A$ for which $f(x, \ldots, x)=x$ for every $x\in X$.

Example 1. For every subset $X\subseteq A$ we have $C_X^*=I_X$ and $I_X^*=[C_X]$. In particular, $P_A^*=O_A$ and $O_A^*=P_A$.

Proof. $C_X^*=I_X$ and $I_X^*\supseteq[C_X]$ are obvious. Now let $f\in I_X^*$ be an n-ary operation and suppose that $f\notin[C_X]$. Then f is neither a projection nor a constant operation with value belonging to X. Therefore there are elements $a_{i1}, \ldots, a_{in}\in A$, $i=1, \ldots, n+2$, such that $a_i=f(a_{i1}, \ldots, a_{in})\neq a_{ii}$, $i=1, \ldots, n$, and $(a_{n+1}, a_{n+2})=(f(a_{n+1,1}, \ldots, a_{n+1,n}), f(a_{n+2,1}, \ldots, a_{n+2,n}))\notin\{(x, x)\mid x\in X\}$. Let $M=(a_{ij})_{(n+2)\times n}$. Since $(a_1, \ldots, a_{n+2})\notin\{(x, \ldots, x)\mid x\in X\}$, and $(a_1, \ldots, a_{n+2})$ is distinct from each column of M, there exists an $(n+2)$-ary operation $g\in I_X$ such that $(f(M))g=g(a_1, \ldots, a_{n+2})\neq f((M)g)$, showing that f and g do not commute and $f\notin I_X^*$. This contradiction shows that $I_X^*\subseteq[C_X]$. Hence $I_X^*=[C_X]$.

Finally if $X=\emptyset$ then we have $I_X=O_A$ and $[C_X]=[\emptyset]=P_A$. □

Example 2. If $|A|\geqq 3$ then $(S_A\cup C_A)^*=H_A$ and $H_A^*=[S_A\cup C_A]$.

Proof. It is well known that $H_A\subseteq I_A$ if $|A|\geqq 3$ (see e.g. [1]). Therefore $(S_A\cup C_A)^*=S_A^*\cap C_A^*=H_A\cap I_A=H_A$. In [10] it is proved that $[S_A\cup C_A]$ acts bicentrally. Thus $H_A^*=((S_A\cup C_A)^*)^*=[S_A\cup C_A]^{**}=[S_A\cup C_A]$. □

For $|A|=2$, E. Post [8] described the lattice of clones over A. Using this result the lattice $\mathscr{L}_A$ can be determined by routine. Figure 1 is the diagram of $\mathscr{L}_A$ in case $|A|=2$. (We use the notation of [9]).

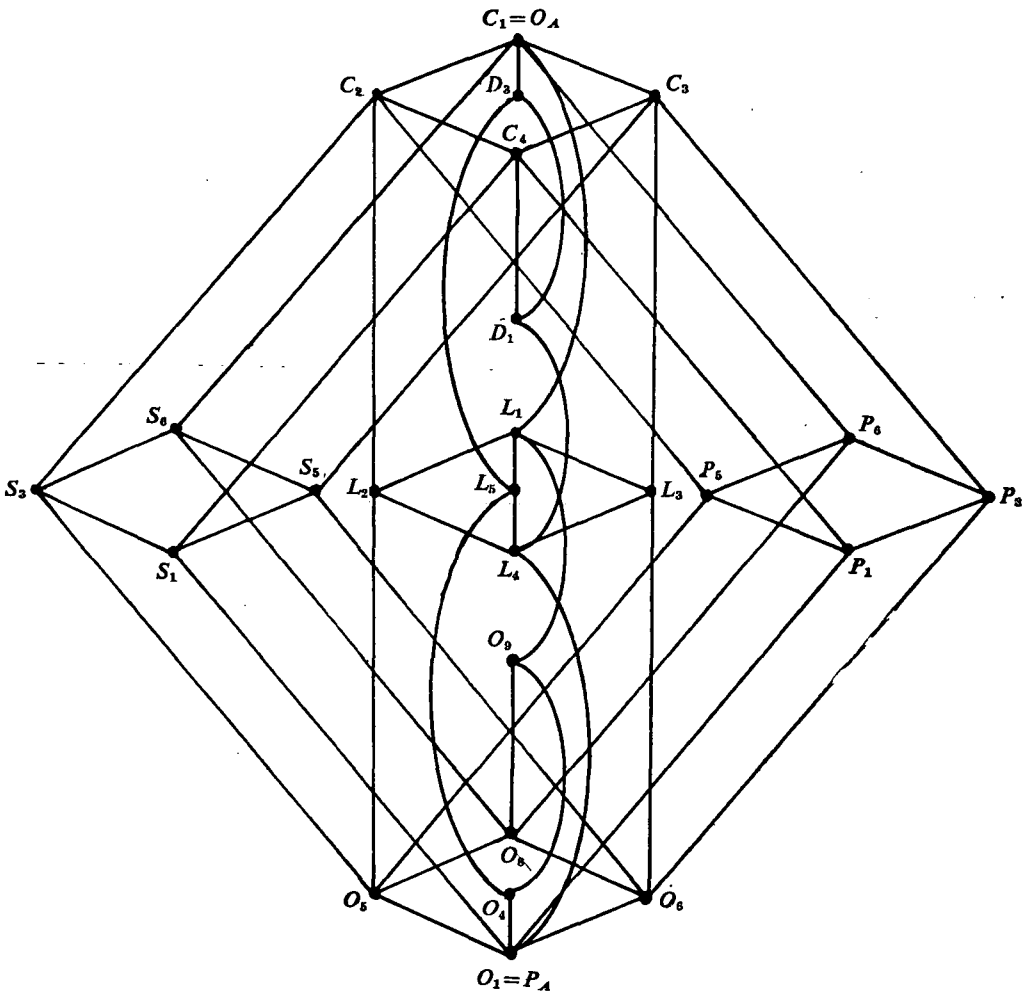


Fig. 1.

Considering the diagram we can observe the following facts: if $|A|=2$ then $\mathscr{L}_A$ has 25 elements, six atoms $(O_4, O_5, O_6, S_1, P_1, L_4)$, and six dual atoms $(D_3, C_2, C_3, S_6, P_6, L_1)$. Remark that the dual automorphism $F \to F^*$ coincides with the reflection of the diagram with respect to the axis $S_3 - P_3$.

For $|A|=3$, $\mathscr{L}_A$ is a finite lattice of power 2986 and it has 44 atoms and dual atoms (see [2], [3] and [4]).

In general we have the following.

Theorem 3. If A is a finite set, then the closure operator $F \to F^{**}$ is algebraic, and $\mathscr{L}_A$ is an atomic and dually atomic algebraic lattice. If A is infinite, then the closure operator $F \to F^{**}$ is not algebraic.

Proof. First let A be a finite set. A. V. Kuznecov showed in [5] that $F=F^{**}$ if and only if F contains every operation parametrically generated by F. From this it follows that the closure operator $F \to F^{**}$ is algebraic. Thus $\mathscr{L}_A$ is an algebraic lattice. It is well-known that there are finite sets $F \subseteq O_A$ such that $F^{**}=O_A$ (see e.g. [4]). Therefore $\mathscr{L}_A$ is dually atomic. Since $\mathscr{L}_A$ is dually isomorphic to itself, it is atomic, too.

A. F. Danil'cenko proved in [4] that if $|A| \geqq 3$ then every dual atom of $\mathscr{L}_A$ is of the form $\{f\}^*$ where $f \in O_A$ is an at most $|A|$-ary operation. From this it follows that $\mathscr{L}_A$ has finitely many dual atoms and atoms (the numbers of atoms and dual atoms are equal).

Now let A be an infinite set and let $x_1, x_2, \ldots \in A$ be pairwise distinct elements. Put $X_i=\{x_i, x_{i+1}, \ldots\}$, $i=1, 2, \ldots$. Then, by Example 1, $I_{X_i} \in \mathscr{L}_A$, $i=1, 2, \ldots$ and clearly $I_{X_1} \supseteq I_{X_2} \supseteq \ldots$. Furthermore $\bigcup_{i=1}^{\infty} I_{X_i} \neq O_A$ and $(\bigcup_{i=1}^{\infty} I_{X_i})^{**}=(\bigcap_{i=1}^{\infty} I_{X_i}^*)^*=$ $=(\bigcap_{i=1}^{\infty} [C_{X_i}])^* P_A^*=O_A$. It follows that the closure operator $F \to F^{**}$ is not algebraic. □

Theorem 4. If $|A| \geqq 5$, then H_A is an atom and $[S_A \cup C_A]$ is a dual atom in $\mathscr{L}_A$.

Proof. First we show that if d is the ternary dual discriminator and l_n $(3 \leqq n \leqq |A|)$ is a near-projection then $\{d\}^*=\{l_n\}^*=[S_A \cup C_A]$. The inclusions $\{d\}^* \supseteq [S_A \cup C_A]$ and $\{l_n\}^* \supseteq [S_A \cup C_A]$ are obvious. Let $f \in O_A \setminus [S_A \cup C_A]$ be an m-ary operation. If f depends on one variable only then we can assume without loss of generality that f is a unary operation. Since f is non-injective and non-constant, there are pairwise distinct elements $a, b, c \in A$ such that $f(a) \neq f(b)=f(c)$. Furthermore choose elements $x_4, \ldots, x_n \in A$ such that $a, b, c, x_4, \ldots, x_n$ are pairwise distinct. Then $f(d(a, b, c))=f(a) \neq f(c)=d(f(a), f(b), f(c))$ and $f(l_n(a, b, x_4, \ldots$ $\ldots, x_n, c))=f(a) \neq f(c)=l_n(f(a), f(b), f(x_4), \ldots, f(x_n), f(c))$ showing that f does not commute with d and l_n, i.e. $f \notin \{d\}^*$ and $f \notin \{l_n\}^*$. Now suppose that f depends on at least two variables, among others on the first. Therefore there are elements $a_2, \ldots, a_n \in A$ such that the unary operation $g(x)=f(x, a_2, \ldots, a_n)$ is not a constant. If f takes on at most $n-1$ elements from A then g is not injective. Therefore $g \notin \{d\}^*$ and $g \notin \{l_n\}^*$. From this it follows that $f \notin \{d\}^*$ and $f \notin \{l_n\}^*$. Finally suppose that f takes on at least n $(\geqq 3)$ values. Since f depends on at least two variables, there are elements $a_1, \ldots, a_m, b_1, \ldots, b_m, a, b, c \in A$ such that a, b and c are pairwise distinct and $a=f(a_1, \ldots, a_m)$, $b=f(b_1, a_2, \ldots, a_m)$, $c=f(a_1, b_2, \ldots, b_m)$

(see e.g. [6]). Then $d(f(a_1, b_2, \ldots, b_m), f(b_1, a_2, \ldots, a_m), f(a_1, \ldots, a_m)) = d(c, b, a) = c \neq a = f(a_1, \ldots, a_m) = f(d(a_1, b_1, a_1), d(b_2, a_2, a_2), \ldots, d(b_m, a_m, a_m))$ showing that $f \notin \{d\}^*$. Finally, since f takes on at least n values, there are elements $x_{i1}, \ldots, x_{im} \in A$, $i = 4, \ldots, n$, such that $a, b, c, x_4, \ldots, x_n$ are pairwise distinct elements where $x_i = f(x_{i1}, \ldots, x_{im})$. Now consider the following $n \times m$ matrix M.

$$M = \begin{pmatrix} x_{41} \ldots & x_{4m} \\ \vdots & \vdots \\ x_{n1} \ldots & x_{nm} \\ a_1\, a_2 \ldots & a_m \\ b_1\, a_2 \ldots & a_m \\ a_1\, b_2 \ldots & b_m \end{pmatrix}.$$

Then $(f(M))l_n = l_n(x_4, \ldots, x_n, a, b, c) = x_4 \neq c = f(a_1, b_2, \ldots, b_m) = f((M)l_n)$ showing that f and l_n do not commute. This completes the proof of the equalities $\{d\}^* = [S_A \cup C_A]$ and $\{l_n\}^* = [S_A \cup C_A]$.

Now we are ready to prove the theorem. Since $H_A^* = [S_A \cup C_A]$, it is enough to show that H_A is an atom in $\mathscr{L}_A$, i.e. for any nontrivial operation $f \in H_A$ we have $\{f\}^{**} = H_A$ or equivalently $\{f\}^* = [S_A \cup C_A]$. In [1] and [7] it is shown that if $|A| \geqq 5$ then every non-trivial clone of homogeneous operations contains the dual discriminator or a near-projection. Therefore, if $f \in H_A$ is a non-trivial operation and $d \in [\{f\}]$ then $[S_A \cup C_A] \subseteq \{f\}^* = [\{f\}]^* \subseteq \{d\}^* = [S_A \cup C_A]$. If $l_n \in [\{f\}]$ for some $n \geqq 3$, then $[S_A \cup C_A] \subseteq \{f\}^* = [\{f\}]^* \subseteq \{l_n\}^* = [S_A \cup C_A]$. Hence $\{f\}^* = [S_A \cup C_A]$, which completes the proof. □

Theorem 5. There exists a function $f \in O_A$ such that $\{f\}^{**} = O_A$.

Proof. If A is a finite set then let $f \in O_A$ be a Sheffer function, i.e. an operation f for which $[\{f\}] = O_A$. Then $[\{f\}^{**} = [\{f\}]^{**} = O_A^{**} = O_A$.

Now let A be an infinite set. In [12] it is proved that there exists a binary rigid relation ϱ on A (ϱ is rigid if the identity operation is the only unary operation preserving ϱ). Choose a rigid relation ϱ and define a binary operation h as follows: $h(x, y) = x$ if $(x, y) \in \varrho$ and $h(x, y) = y$ if $(x, y) \notin \varrho$. We show that $\{h\}^* \cap S_A = \{id_A\}$. Indeed, let $t \in S_A$ and $t \neq id_A$. Then there is a pair $(x, y) \in \varrho$ such that $(t(x), t(y)) \notin \varrho$. Clearly $x \neq y$, since otherwise the unary constant operation $A \to \{x\}$ preserves ϱ. It follows that $t(h(x, y)) = t(x) \neq t(y) = h(t(x), t(y))$ and $t \notin \{h\}^*$.

Let $g \in O_A$ be a fixed point free permutation whose cycles are all infinite. Furthermore, let $a, b \in A$ with $a \neq b$.

Now we are ready to define an operation f such that $\{f\}^{**} = O_A$. Let

$$f(x, y, z, u) = \begin{cases} g(x) & \text{if } x = y = z = u, \\ d(y, z, u) & \text{if } y = g(x), \\ h(z, u) & \text{if } x = g(y), \\ a & \text{if } y = g(g(x)), \\ b & \text{if } x = g(g(y)), \\ x & \text{otherwise.} \end{cases}$$

Denote by c_a and c_b the unary constant operations with values a and b, respectively. Then $g, d, h, c_a, c_b \in [\{f\}]$ since $f(x, x, x, x) = g(x)$, $f(x, g(x), y, z) = d(x, y, z)$, $f(g(x), x, x, y) = h(x, y)$, $f(x, g(g(x)), x, x) = c_a(x)$ and $f(g(g(x)),$

$x, x, x)=c_b(x)$. If $t\in\{f\}^*$ then $t\in\{d, h, c_a, c_b\}^*$. Since $t\in\{d\}^*$, by Theorem 4, $t\in[S_A\cup C_A]$. We can suppose that t is unary. If $t\in S_A$ then $t\in\{h\}^*$ implies $t=id_A$. If $t\in C_A$, i.e. t is a constant operation with value x_0 then we have that $a=c_a(t(a))=$ $=t(c_a(a))=x_0=t(c_b(a))=c_b(t(a))=b$ which is a contradiction. Thus we have $\{f\}^*=P_A$ and $\{f\}^{**}=P_A^*=O_A$. □

Let $B\subseteqq A(B\neq\emptyset)$ and let s be a mapping from A onto B such that $s(b)=b$ for every $b\in B$. For any operation $f\in O_B^{(n)}$, $n\geqq 1$, let us define an operation $f^S\in O_A$ as follows: $f^S(a_1, \ldots, a_n)=f(s(a_1), \ldots, s(a_n))$ for any $a_1, \ldots, a_n\in A$. For any $F\subseteqq O_B$ let $F^S=P_A\cup\{f^S\mid f\in F\}$.

Theorem 6. Let $F\subseteqq O_B$ such that $id_B\in F$. Then $(F^S)^{**}=(F^{**})^S$. In particular, if $F=F^{**}$ then $F^S=(F^S)^{**}$.

Proof. We shall prove the theorem through some statements:

(1) $s\in F^S$ and $s\in(F^S)^*$.

Since $id_B\in F$, we have $s=id_B^S\in F^S$. Let $g\in F$. If $g\in P_A$ then, clearly, s commutes with g. If $g=f^S$ for some $f\in F$, then for any $a_1, \ldots, a_n\in A$ we have $s(g(a_1, \ldots, a_n))=s(f^S(a_1, \ldots, a_n))=s(f(s(a_1), \ldots, s(a_n))=f(s(s(a_1)), \ldots, s(s(a_n)))=$ $=g(s(a_1), \ldots, s(a_n))$. Hence s commutes with g and $s\in(F^S)^*$.

(2) If $g\in(F^S)^*$ then g preserves B.

Indeed, if g is n-ary and $b_1, \ldots, b_n\in B$ then $g(b_1, \ldots, b_n)=g(s(b_1), \ldots, s(b_n))=$ $=s(g(b_1, \ldots, b_n))\in B$.

(3) $g\in(F^S)^*$ if and only if $g_B\in F^*$ and g commutes with s.

First suppose that $g\in(F^S)^*$. Then g commutes with s, since $s\in F^S$. If $f\in F$, then g commutes with f^S. By (2), we have $g_B\in O_B$, and clearly the restriction of f^S to B coincides with f. These facts imply that g_B commutes with f. Hence $g_B\in F^*$. Now suppose that $g_B\in F^*$, g commutes with s, and $f^S\in F^S(f\in F)$. Let g and f be m-ary and n-ary, respectively, and choose arbitrary elements $a_{i1}, \ldots, a_{im}\in A$, $i=1, \ldots, n$. Then

$$f^S(g(a_{11}, \ldots, a_{1m}), \ldots, g(a_{n1}, \ldots, a_{nm}))=f(s(g(a_{11}, \ldots, a_{1m})), \ldots, s(g(a_{n1}, \ldots, a_{nm})))=$$
$$=f(g_B(s(a_{11}), \ldots, s(a_{1m})), \ldots, g_B(s(a_{n1}), \ldots, s(a_{nm})))=$$
$$=g_B(f(s(a_{11}), \ldots, s(a_{n1}))), \ldots, f(s(a_{1m}), \ldots, s(a_{nm}))=$$
$$=g(f^S(a_{11}, \ldots, a_{n1}), \ldots, f^S(a_{1m}, \ldots, a_{nm})).$$

Hence g commutes with f^S and $g\in(F^S)^*$.

(4) If $f\in F^*$ then $f^S\in(F^S)^*$.

Clearly, the restriction f_B^S to B coincides with f, and f^S commutes with s. Therefore, by (3), we have $f^S\in(F^S)^*$.

(5) If $g\in(F^S)^{**}$ then $g\in P_A$ or g maps into B.

Suppose $g\in(F^S)^{**}\setminus P_A$ is an n-ary operation which takes on a value from $A\setminus B$. Since g is not a projection, for every $i\in\{1, \ldots, n\}$ there are $a_{i1}, \ldots, a_{in}\in A$ such that $a_i=g(a_{i1}, \ldots, a_{in})\neq a_{il}$. Furthermore let $a_{n+1,1}, \ldots, a_{n+1,n}\in A$ such that $g(a_{n+1,1}, \ldots, a_{n+1,n})=a_{n+1}\notin B$. Let us define an $(n+1)$-ary operation $h\in O_A$ as follows:

$$h(x_1, \ldots, x_{n+1})=\begin{cases}s(a_{n+1}) & \text{if } (x_1, \ldots, x_{n+1})=(a_1, \ldots, a_{n+1}),\\ x_{n+1} & \text{otherwise.}\end{cases}$$

Then h commutes with s, and h_B, being a projection, belongs to F^*. Therefore, by (3), $h \in (F^S)^*$. Now $g(h(a_{11}, \ldots, a_{n+1,1}), \ldots, h(a_{1n}, \ldots, a_{n+1,n})) = g(a_{n+1,1}, \ldots, a_{n+1,n}) = a_{n+1} \neq s(a_{n+1}) = h(a_1, \ldots, a_{n+1}) = h(g(a_{11}, \ldots, a_{1n}), \ldots, g(a_{n+1,1}, \ldots, a_{n+1,n}))$. It follows that g does not commute with h, which is a contradiction.

(6) If $g \in (F^S)^{**}$ then g preserves B.

This follows from (5)

(7) If $g \in (F^S)^{**}$ then $g_B \in F^{**}$.

Let $g \in (F^S)^{**}$ and let f be an arbitrary operation in F^*. Then, by (4), we have that g commutes with f^S. Taking into consideration (6), this implies that g_B ($\in O_B$) commutes with f (the restriction of f^S to B). It follows that $g_B \in F^{**}$.

Now we are ready to prove the theorem. First let $g \in (F^S)^{**}$. If $g \in P_A$ then clearly $g \in (F^{**})^S$. Suppose that $g \notin P_A$ and let $g_B = f$. Taking into consideration (5), (1) and (7), we have that g maps into B, g commutes with s, and $f \in F^{**}$. Thus if g is n-ary then for any $a_1, \ldots, a_n \in A$ we have $g(a_1, \ldots, a_n) = s(g(a_1, \ldots, a_n)) = g(s(a_1), \ldots, s(a_n)) = f(s(a_1), \ldots, s(a_n))$ showing that $g = f^S$ and $g \in (F^{**})^S$. Finally let $g \in (F^{**})^S$. If $g \in P_A$ then $g \in (F^S)^{**}$. If $g \notin P_A$ then there is an $f \in F^{**}$ such that $g = f^S$. Take an arbitrary operation h from $(F^S)^*$. Then, by (3), h commutes with s and $h_B \in F^*$. It follows that h_B commutes with f ($h_B \in F^* = (F^{**})^*$). Let g and h be m-ary and n-ary, respectively, and choose arbitrary elements $a_{i1}, \ldots, a_{im} \in A$, $i = 1, \ldots, n$. Now

$$h(g(a_{11}, \ldots, a_{1m}), \ldots, g(a_{n1}, \ldots, a_{nm})) = h_B(f(s(a_{11}), \ldots, s(a_{1m})), f(s(a_{n1}), \ldots, s(a_{nm}))) =$$

$$= f(h_B(s(a_{11}), \ldots, s(a_{n1})), \ldots, h_B(s(a_{1m}), \ldots, s(a_{nm}))) =$$

$$= f(h(s(a_{11}), \ldots, s(a_{n1})), \ldots, h(s(a_{1m}), \ldots, s(a_{nm}))) =$$

$$= f(s(h(a_{11}, \ldots, a_{n1})), \ldots, s(h(a_{1m}, \ldots, a_{nm}))) = g(h(a_{11}, \ldots, a_{n1}), \ldots, h(a_{1m}, \ldots, a_{nm})).$$

It follows that g commutes with h and $g \in (F^S)^{**}$. □

Corollary 7. The mapping $F \to F^S$ from $\mathscr{L}_B$ into $\mathscr{L}_A$ is an isomorphism.

Proof. From Theorem 6 it follows that if $F \in \mathscr{L}_B$ then $F^S \in \mathscr{L}_A$. Observe that $(F_1 \cap F_2)^S = F_1^S \cap F_2^S$ and $(F_1 \cup F_2)^S = F_1^S \cup F_2^S$ for any $F_1, F_2 \in \mathscr{L}_B$. Therefore taking into consideration Theorem 6, for any $F_1, F_2 \in \mathscr{L}_B$ we have that $(F_1 \wedge F_2)^S = (F_1 \cap F_2)^S = F_1^S \cap F_2^S = F_1^S \wedge F_2^S$ and $(F_1 \vee F_2)^S = ((F_1 \cup F_2)^{**})^S = ((F_1 \cup F_2)^S)^{**} = (F_1^S \cup F_2^S)^{**} = F_1^S \vee F_2^S$. Finally, it is obvious that the mapping $F \to F^S$ is injective. □

Corollary 8. If $s \neq id_A$ then $[\{s\}]$ is an atom in $\mathscr{L}_A$.

Proof. Let $P_B \subseteq O_B$ be the set of projections on B. Then $P_B^S = [\{s\}]$ and therefore, by Theorem 6, $[\{s\}] \in \mathscr{L}_A$. It is trivial that $[\{s\}]$ is an atom in $\mathscr{L}_A$. □

BOLYAI INSTITUTE
ARADI VÉRTANÚK TERE 1
6720 SZEGED, HUNGARY

References

[1] B. Csákány and T. Gavalcová, Finite homogeneous algebras. I, Acta Sci. Math., 42 (1980), 57—63.
[2] A. F. Danil'čenko, On the parametrical expressibility of functions of three-valued logic (in Russian), Algebra i logika, 16/4 (1977), 379—494.
[3] A. F. Danil'čenko, Parametrically closed classes of functions of three-valued logic (in Russian), Izv. Akad. Nauk Moldav. SSR, 1978, no. 2, 13—20.
[4] A. F. Danil'čenko, On parametrical expressibility of the functions of k-valued logic, in: Finite Algebra and Multiple-valued Logic (Proc. Conf. Szeged, 1979). Colloq. Math. Soc. J. Bolyai, vol. 28, North-Holland (Amsterdam, 1981), 147—159.
[5] A. V. Kuznecov, On detecting non-deducibility and non-expressibility (in Russian), in: Logical deduction, Nauka, Moscow (1979), 5—33.
[6] A. I. Mal'cev, A strengthening of the theorems of Słupecki and Jablonskiĭ (in Russian), Algebra i Logika 6, no. 3 (1967), 61—75.
[7] S. S. Marčenkov, On homogeneous algebras (in Russian), Dokl. Acad. Nauk SSSR, 256 (1981), 787—790.
[8] E. Post, The two-valued iterative systems of mathematical logic, Ann. Math. Studies 5, Princeton (1941).
[9] R. Pöschel und L. A. Kalužnin, Functionen- und Relationenalgebren, Ein Kapitel der diskreten Mathematik, Math. Monographien B. 15, Berlin (1979).
[10] L. Szabó, Endomorphism monoids and clones, Acta Math. Acad. Sci. Hungar., 26 (1975), 279—280.
[11] L. Szabó, Concrete representation of related structures of universal algebras. I, Acta Sci. Math., 40 (1978).
[12] P. Vopenka, A. Pultr and Z. Hedrlin, A rigid relation exists on any set, Comment. Math. Univ. Carolinae, 6 (1965), 149—155.

Received Agust 30, 1983.

Analysis of data flow for SIMD systems

REINHARD KLETTE

0. Introduction

A general approach to characterizing the inherent complexity of computational problems is given by the quantitative analysis of the extent of the data flow that has to be performed during the solution of these problems. On the other hand, any parallel processing system possesses a restricted ability for fast data transfer determined essentially by the interconnection pattern of the processing elements. In the present paper, these general observations, as previously mentioned by Gentleman (1978), Siegel (1979), Abelson (1980), or Klette (1980), will be transformed into precise definitions of local, global and total data transfer within SIMD systems, and the corresponding definitions of local, global and total data dependencies for computational problems as well. The basic relation between these corresponding notions — the computational time must at least be sufficient for realizing the necessary extent of data transfer — will be represented in a so-called data transfer lemma that outlines the starting point of our formalized method of obtaining lower time bounds by data flow analysis. This approach will be illustrated by application to a variety of different parallel processing architectures where the unifying feature will be that we shall use SIMD models that employ an interconnection network and use no shared memory. Our parallel processing systems will be abstract models of computation where the level of abstraction may be compared with that of a random access machine (RAM); cp. Aho et al. [2] for this model of serial computation. For computational problems such as those mentioned in the present paper the author was inspired by the digital image processing area, where reference is made to Rosenfeld et al. [9] and Klette [5]. But, of course, this does not represent a serious restriction; e.g., matrix multiplication or pattern matching are computational problems of general importance.

The general SIMD model as used in this paper is characterized by a finite or infinite set of processing elements (PEs), an interconnection network, and a central processing unit (CPU). For a rough scheme of an SIMD system which the reader may have in mind throughout this paper, see Fig. 1.

CPU. The CPU has a (central) random access memory which consists of a finite or infinite sequence of registers $r_0, r_1, r_2, \ldots$ with a distinguished accumu-

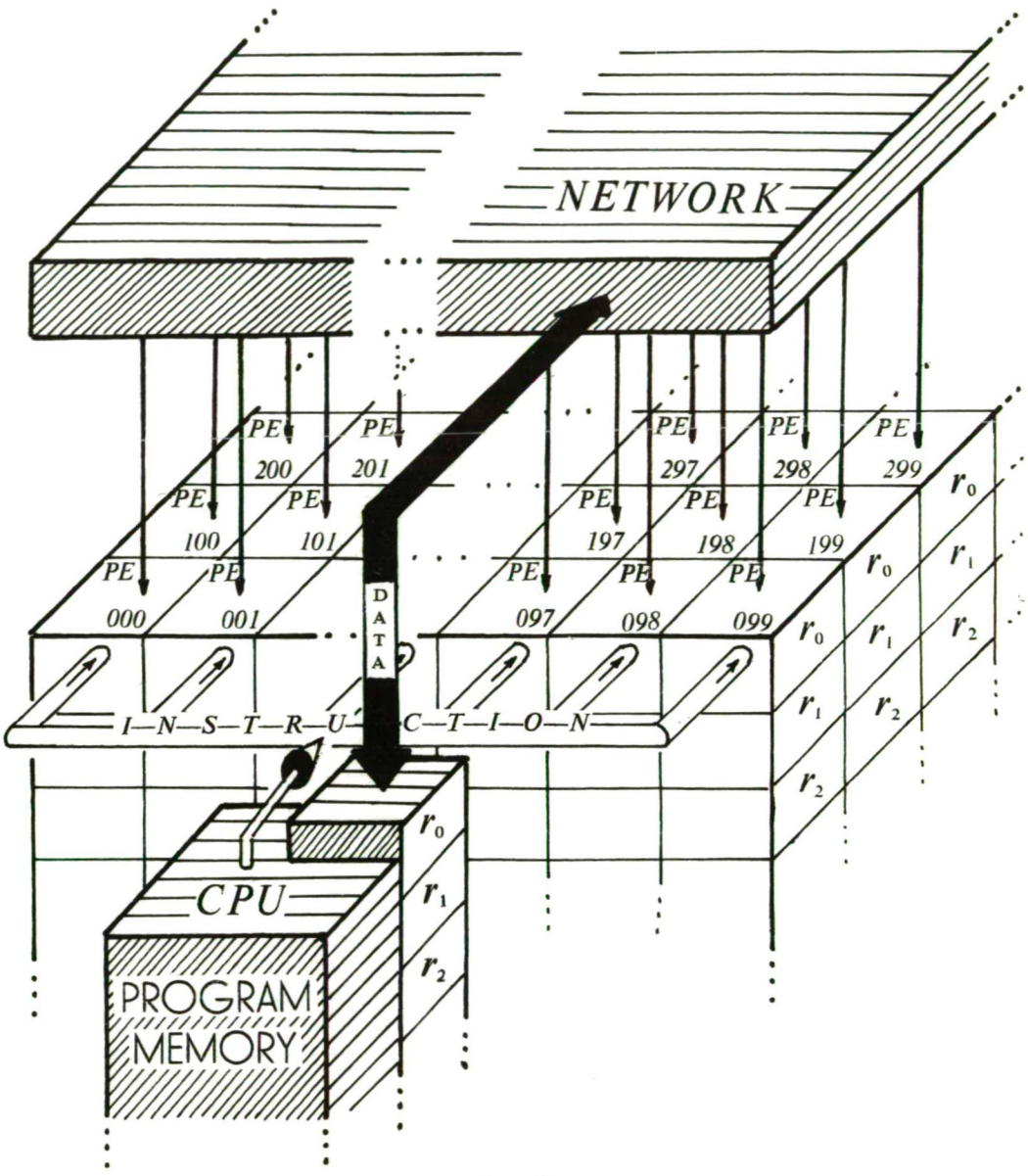


Figure 1.
Scheme of an SIMD system

lator r_0. Let D_{CPU} be the depth of this random access memory, i.e., the number of CPU registers, for $1 \leqq D_{CPU} \leqq \infty$. Furthermore, let W_{CPU} be the word length of these registers (number of bit positions), which is assumed to be constant for all CPU registers, for $1 \leqq W_{CPU} \leqq \infty$. The CPU spreads a single instruction stream to the synchronized working PEs. The programs of the system are stored in *a*, potentially size-unlimited, special program memory of the CPU. Part of any instruction addressed to the PEs is an enable/disable mask to select a subset of the PEs that are to perform the given instruction; the remaining PEs will be idle. The CPU may read the accumulator contents of any one PE of a specified subset of all PEs, and is able to transfer its accumulator contents to some of the PE accumulators. Any data transfer between CPU and PEs is restricted to serial mode.

PEs. Each PE has some (local) random access memory which consists of a finite or infinite sequence of registers $r_0, r_1, r_2, \ldots$ with a distinguished register r_0 called the accumulator. Let D_{PE} be the depth of these random access memories, i.e., this depth is assumed to be constant for all PEs of a given system, for $1 \leqq D_{PE} \leqq \infty$. Furthermore, let W_{PE} be the unique word length of the PE registers, for $1 \leqq W_{PE} \leqq \infty$. Each PE is capable of performing some basic operations which take place in its accumulator. Direct data access is restricted to its own registers, to the accumulators of the directly connected PEs in the sense of the given interconnection network, and, possibly, to the accumulator of the CPU. The PEs are indexed by integers or tuples of integers. Each PE knows its index. Let N_{PE}, $0 \leqq N_{PE} \leqq \infty$, be the number of PEs of a given system, and $\mathbf{ind} = \{j_1, j_2, \ldots, j_{N_{PE}}\}$ be the set of all PE indices of a given SIMD system.

Interconnection network. Each PE is located in a node of a given undirected graph representing the two-way interconnection scheme. Any PE may uniquely identify the different edges connected to its node by using a given coding scheme. Let N_{IN} be the branching degree of the network, i.e., the maximum degree of the nodes of the given graph, for $0 \leqq N_{IN} < \infty$.

For the selection of a specialized SIMD model the following system features may be concretely specified:

- off-line or on-line communication with the outside world,
- special values for N_{PE}, N_{IN}, D_{CPU}, D_{PE}, W_{CPU}, or W_{PE},
- the set **ind**,
- the interconnection network structure including the edge coding scheme,
- the CPU instruction set including the available set of enable/disable masks as well as the method of the data exchange between CPU and PEs, and
- the restrictions on the system in communication with the outside world, i.e., input and output management.

Note that as regards the technical realization of an SIMD computing facility, in principle, one implementation may offer different ways to run such a system, i.e., the working principles of several SIMD models as considered in the present paper may be unified within one implementation. Essentially, this is the problem of constructing a flexible interconnection network with reconfigurability, and/or of running a system using different modes.

The outline of this paper is as follows. In the first section we shall present some standardized system description features for specifications of SIMD models. In Section 2 we shall describe how the data flow of an SIMD system may be measured

by functions in a quantitative way. Then, in Section 3 the corresponding notions of data dependencies will be explained for computational problems. In Section 4 the data transfer lemma will be given as well as some applications of this lemma to different models of computation for lower time bound determination. Our concluding remarks are given at the end of the paper.

The standard SIMD models as described in Section 1 constitute the framework of a parallel simulation system (PARSIS) presently under implementation; cp. Legendi [7] for a similar project for simulation of cellular processors.

1. OFF-NETs and ON-NETs

In our experience in parallel program design the exclusion of given technical restrictions, e.g., on N_{PE}, N_{IN}, etc., in the first steps of problem solutions, enables us to find important methods of parallelization of solution processes as well as general features for system description. Of course, for concrete implementation quite a lot of time must be spent in taking given restrictions for N_{PE}, N_{IN}, etc. into consideration. The present paper is concerned with the first phase, the theoretical preparation for the second phase, which is the concrete implementation. In this sense, we shall deal with abstract SIMD models throughout this paper. More detailed discussion will be the subject of forthcoming papers, depending on the progress of the PARSIS project.

The common one-accumulator computer, e.g., the random access machine (RAM) in the sense of Aho et al. [2], may be considered as the simplest example of an abstract SIMD system — $N_{PE}=0$ and $D_{CPU}=W_{CPU}=\infty$. We shall use the RAM as the underlying model for serial data processing where, in distinction to [2], infinite precision, real number arithmetic is assumed, which is convenient for our theoretical considerations of computational problems such as the Fourier transform, or for operations on finite sets of points in the real plane, by avoiding discussions of round-off errors. In this sense, our standardized system description features start with the declaration of abstract registers.

Abstract registers. For an SIMD system with abstract registers we assume that any register may store one real number at a time, without any special encoding tricks. For our theoretical considerations in this paper, it is not important to specify how the reals are stored in these abstract registers by special bit representations.

Standard register enumeration. We assume a unique enumeration of all registers as follows. For registers r_m of the PE with index j or (j,k), called PE(j) or PE(j,k) in the sequel, we use the integer tuples (j,m) or (j,k,m), respectively, and for register r_m of the CPU just the integer m.

Uniform network structure. Either $N_{IN}=0$, or $N_{IN}=p\geqq 1$ and the network structure is characterized by p different functions $f_0, f_1, \ldots, f_{p-1}$ on the set **ind** of all PE indices in the following way. For $j,k\in$**ind**, PE(j) and PE(k) are *directly connected* iff there exists an i, $0\leqq i\leqq p-1$, such that $f_i(j)=k$. Because of our assumption that all connections are two-way it follows that

$$(\wedge j, k\in\textbf{ind})[(\vee i\in\{0,1,\ldots,p-1\})f_i(j)=k \equiv (\vee h\in\{0,1,\ldots,p-1\})\ f_h(k)=j].$$

In [10] the functions $f_0, f_1, \ldots, f_{p-1}$ were called *interconnection functions*. With the exception of a fixed set of PEs at the network border, we also claim that all

PEs are directly connected to exactly p different PEs. When $f_i(j)=k$, PE(k) is called the *ith neighbor* of PE(j). In this way, the edge coding scheme for uniform networks is defined. For each PE, the neighborhood consists of all (i.e., at most p) neighbor PEs. Examples of infinite networks as well as finite networks matching our uniformity demand are given in Table 1. In the sequel we shall use these networks as defined here.

Some remarks are necessary regarding Table 1. The left-right 2^i (LR2I) network and the left-right-up-down 2^i network (LRUD2I) network were used for vector machines in Pratt et al. [8] and Klette et al. [6], respectively, without the restriction by an integer m as stated in Table 1. Note that we have restricted ourselves to interconnection networks with finite branching degree. The special form of the set **ind** in the Quadtree network is determined by our standard PE address masking scheme as defined later on. The finite uniform networks mentioned in Table 1 were studied by Siegel [10] — the perfect shuffle (PS), the ILLIAC, the Cube, the plus-minus 2^i (PM2I), and the wrap-around plus-minus 2^i (WPM2I) network, with the modification that the PS network is an undirected graph to match our uniform network convention, i.e., for the PS network the inverse shuffle function was added in comparison to [10]. For $j \in \mathbf{ind} = \{0, 1, \ldots, 2^m-1\}$ let $a_{m-1} \ldots a_1 a_0$ denote the binary representation of j and $\bar{a}_i$ denote the complement of a_i. Then

$$\text{exch}\,(a_{m-1} \ldots a_1 a_0) = a_{m-1} \ldots a_1 \bar{a}_0,$$

$$\text{shuf}\,(a_{m-1} \ldots a_1 a_0) = a_{m-2} \ldots a_1 a_0 a_{m-1},$$

$$\text{shuf}^{-1}\,(a_{m-1} \ldots a_1 a_0) = a_0 a_{m-1} \ldots a_2 a_1,$$

$$\text{cube}_i\,(a_{m-1} \ldots a_{i+1} a_i a_{i-1} \ldots a_0) = \bar{a}_{m-1} \ldots a_{i+1} \bar{a}_i \bar{a}_{i-1} \ldots a_0,$$

$$\text{WPM}_{+i}(a_{m-1} \ldots a_i \ldots a_0) = b_{m-1} \ldots b_i \ldots b_0,$$

where $b_{i-1} \ldots b_0 b_{m-1} \ldots b_{i+1} b_i = (a_{i-1} \ldots a_0 a_{m-1} \ldots a_{i+1} a_i) + 1 \bmod 2^m$,

$$\text{WPM}_{-i}(a_{m-1} \ldots a_i \ldots a_0) = b_{m-1} \ldots b_i \ldots b_0,$$

where $b_{i-1} \ldots b_0 b_{m-1} \ldots b_{i+1} b_i = (a_{i-1} \ldots a_0 a_{m-1} \ldots a_{i+1} a_i) - 1 \bmod 2^m$, for $0 \leqq i < m$ and $m \geqq 1$.

Standard PE *masking scheme.* As standard masks we shall use the simple bit patterns for PE indices as used, for example, in [10]. In the case of integer indices, a standard PE address mask is given by an arbitrary, non-empty word on the alphabet $\{0, 1, x\}$ enclosed by brackets, where x represents the "dont't care" situation. The only PEs that will be active are those whose address (i.e., index) matches the mask from right to left, where the indices are given in binary representation; 0 matches 0, 1 matches 1, and either 0 or 1 matches x. For example, by mask $[x]$ all PE's are activated. For the representation of concrete standard masks within programs, etc. we take liberties such as [all PE's] instead of $[x]$, or [odd PE's] instead of $[1x]$ if the rightmost bit position is assumed to be the sign position. In the case of integer tuple indices, the standard PE address masks are arbitrary tuples of non-empty words on $\{0, 1, x\}$ enclosed by brackets. Note that for infinite networks as given in Table 1 any given PE address mask activates an infinite manifold of PE's. For example, the mask $[0xx]$ applied to the bintree network will

Table 1. Uniform networks

Network	ind	N_{IN}	Case	Edge coding scheme							
				0	1	2	3	4	5	6	7
LINEAR	integers	2	all	$j-1$	$j+1$	—	—	—	—	—	—
LR2I^m	integers	$2m$	all	$f_{2i}(j)=j+2^i$ and $f_{2i+1}(j)=j-2^i$ for $0\leqq i<m$ and $m\geqq 2$							
BINTREE	positive integers	3	$j\geqq 2$	$\lfloor j/2 \rfloor$	—	—	—	—	—	—	—
			all	—	$2j$	$2j+1$	—	—	—	—	—
TRIANGLE	positive integers	5	$j\geqq 2$	$\lfloor j/2 \rfloor$	—	—	—	—	—	—	—
			all	—	$2j$	$2j+1$	—	—	—	—	—
			$j\neq 2^i$	—	—	—	$j-1$	—	—	—	—
			$j\neq 2^i-1$	—	—	—	—	$j+1$	—	—	—
QUADTREE	$\bigcup_{i=0}^{\infty} \cdot \{4^i, \ldots, 2\cdot 4^i-1\}$	5	$j\geqq 4$	$\lceil j/4 \rceil$	—	—	—	—	—	—	—
			all	—	$4j$	$4j+1$	$4j+2$	$4j+3$	—	—	—
HEXAGONAL	tuples of integers	3	all	$(j,k-1)$	$(j,k+1)$	—	—	—	—	—	—
			$j+k$ even	—	—	$(j-1,k)$	—	—	—	—	—
			$j+k$ odd	—	—	$(j+1,k)$	—	—	—	—	—
SQUARE	tuples of integers	4	all	$(j,k-1)$	$(j,k+1)$	$(j-1,k)$	$(j+1,k)$	—	—	—	—
TRIAGONAL	tuples of integers	6	all	$(j,k-1)$	$(j,k+1)$	$(j-1,k)$	$(j+1,k)$	$(j-1,k-1)$	$(j+1,k+1)$	—	—
DIAGONAL	tuples of integers	8	all	$(j,k-1)$	$(j,k+1)$	$(j-1,k)$	$(j+1,k)$	$(j-1,k-1)$	$(j+1,k+1)$	$(j-1,k+1)$	$(j+1,k-1)$
LRUD2I^m	tuples of integers	$4m$	all	$f_{4i}(j,k)=(j+2^i,k)$, $f_{4i+1}(j,k)=(j-2^i,k)$, $f_{4i+2}(j,k)=(j,k+2^i)$, $f_{4i+3}(j,k)=(j,k-2^i)$, for $0\leqq i<m$ and $m\geqq 2$							
PS^m	$\{0,1,\ldots,2^m-1\}$	3	all	exch	shuf	shuf^{-1}	—	—	—	—	—
ILLIAC^m	$\{0,1,\ldots,2^m-1\}$	4	all	$+1 \bmod 2^m$	$-1 \bmod 2^m$	$+\frac{m}{2} \bmod 2^m$	$-\frac{m}{2} \bmod 2^m$	—	—	—	—
CUBE^m	$\{0,1,\ldots,2^{m-1}\}$	m	all	$f_i(j)=\text{cube}_i(j)$, for $0\leqq i<m$							
PM2I^m	$\{0,1,\ldots,2^{m-1}\}$	$2m$	all	$f_{2i}(j)=j+2^i \bmod 2^m$, $f_{2i+1}(j)=j-2^i \bmod 2^m$, for $0\leqq i<m$							
WPM2I^m	$\{0,1,\ldots,2^{m-1}\}$	$2m$	all	$f_{2i}(j)=\text{WPM}_{+i}(j)$, $f_{2i+1}(j)=\text{WPM}_{-i}(j)$, for $0\leqq i<m$							

activate the processing elements PE(2) and PE(3) on layer 1 of the bintree, disables layer 2, enables the first four PE's of layer 3, and so on, where the common binary representation of non-negative integers is assumed for the PE indices of the bintree network.

Abstract CPU *instruction set.* For any one of our theoretical SIMD systems, we shall assume that its CPU instruction set may be obtained by special interpretation and selection of the instructions of an abstract CPU instruction reservoir defined as follows. There are two different types of instructions, parallel instructions for activating some of the PEs, and serial instructions where the CPU itself is addressed for certain activity. Any *parallel instruction* consists of a PE address mask, an operation code (READ, WRITE, LOAD, STORE, OP, or OP_{l+1}, $l \geqq 1$), and an operation address α where we shall use the standard register enumeration for explaining the meaning of these operation addresses. For the *serial instructions,* we assume branching instructions JUMP b, JGTZ b, JZERO b, JLTZ b (where b symbolizes an instruction number in a CPU program and the contents of the CPU accumulator are tested), the HALT instruction, and instructions consisting of an operation code (READ, WRITE, LOAD, STORE, OP_1, or OP_2). See Table 2

Table 2. Abstract CPU instruction set without test and stop instructions

Instruction	Possible operation address α			
[mask] READ α	m;	$^*m$		
[mask] WRITE α	m;	$^*m$		
[mask] LOAD α	m;	$^*m$;	: i	
[mask] STORE α	m;	$^*m$;	: $i_1, i_2, \ldots, i_l$	
[mask] OP_1 α	m;	$^*m$;	: i	
[mask] OP_2 α	m;	$^*m$;	: i	
[mask] OP_{l+1}	: $i_1, i_2, \ldots, i_l$			
READ α	m;	$^*m$		
WRITE α	$=x$;	m;	$^*m$	
LOAD α	$=x$;	m;	$^*m$;	(j)
STORE α	m;	$^*m$;	(j)	
OP_1 α	$=x$;	m;	$^*m$;	(j)
OP_2 α	$=x$;	m;	$^*m$;	(j)

for the complete abstract CPU instruction set without jump and stop instructions. In the case of a parallel instruction, OP_1 denotes a unary operation determining the new accumulator contents of all activated PEs by a certain transformation of the contents of the register addressed by α as well as the old accumulator contents of the activated PEs; and OP_{l+1} denotes an $(l+1)$-ary operation in the same sense. For the activated PE(j) the operation address m indicates the contents of register (j, m), $^*m$ indicates the contents of register (j, n) if the nonnegative integer n is the contents of register (j, m) at that moment (i.e., indirect operand addressing, in any situation of incorrect programming; e.g., in the case that (j, m) does not have a nonnegative integer contents at that moment, an interrupt of the programmed system is assumed), and the operand : $i_1, i_2, \ldots, i_l$ for $l \geqq 1$ indicates the contents of the accumulators of those neighbors of the activated PEs that are encoded by

$i_1, i_2, \ldots, i_l$ according to the edge coding scheme of the interconnection network. LOAD and STORE have the obvious meanings that the accumulator contents of the activated PEs are replaced by the addressed value, or copied to the addressed registers, respectively. READ and WRITE denote the necessary operations for communication with the outside world where the source and the destination of the data in the "outside world" remain unspecified (certain places within a computing environment not belonging to the given SIMD system itself). In the case of a serial instruction, the unary operation OP_1 and the binary operation OP_2 produce new CPU accumulator contents by a certain transformation of the addressed values, where in the case of OP_2 the old CPU accumulator contents is used as the operand in the first position. READ, WRITE, LOAD, and STORE have the obvious fixed meanings. The operands $=x, m, {}^*m$, and (j) indicate the data unit x itself, the contents of CPU register m, the contents of CPU register n if register m contains the nonnegative number n at that moment, and the contents of register $(j, 0)$, respectively. Note that with this abstract CPU instruction set data transfer between the CPU and the PEs is possible via the accumulators in serial mode only. Furthermore, for a specialized SIMD model, it is convenient to identify the basic computational power of the PEs and the CPU with that of the RAM as represented by the RAM instruction set [2, Fig. 15], roughly speaking. In this way, an interesting point is provided by the description of how the PEs are able to perform local logical decisions in SIMD mode as we shall explain in Example 1 by equation (1) for a special SIMD model.

Off-line I/O *convention.* For the off-line communication of an SIMD system with the outside world we assume that a special set of input registers of the system is fixed such that all other registers of the system contain value zero at the beginning of any computation (moment $t=0$) as it is assumed for those input registers not actually needed for the placement of input data. Each of the input registers may contain at most one data unit of the input data. Thus, for concrete problem solutions, it is necessary to specify

- what data structure is assumed for the given input data, and
- how the data are placed in the given input register set.

Also, a set of output registers of the system must be fixed. In this sense, for concrete problem solutions it has to be clear

- what is the desired data structure for the output data, and
- how this data structure has to be stored, or computed in the predetermined output register set.

As off-line I/O convention we declare that for a certain L, $1 \leqq L \leqq D_{CPU}$, the CPU registers $0, 1, \ldots, L-1$ are fixed to be input and output registers, and for any PE(j), if there exists a certain $m \geqq 0$ such that register (j, m) is fixed to be an input register (output register) then register $(j, 0)$ is an input register (output register) as well. What is true for the register holds for the accumulator, too.

On-line I/O *convention.* For the on-line communication of an SIMD system with the outside world some registers are predetermined to act as input and/or output registers. As on-line I/O convention we adopt the same rules as in the off-line case. But, at the beginning of any on-line computation (moment $t=0$), all registers of the system are assumed to hold value zero. Input data or output data may enter or leave the system at a moment as specified by the CPU program according to READ or WRITE instructions. In any correct program these input (output)

instructions have to be addressed to a proper subset of all registers specified as input (output) registers. For the input (output) data it is assumed that there exists a memory facility in the outside world from where (to where) the input (output) data are obtained (given) by the system. Thus, for concrete problem solutions it is necessary to specify

- what data structures are assumed for the input and output data, and
- how these data are partitioned into waves of information such that one wave may enter (leave) the system per input (output) operation as performed according to the CPU program.

The size of these waves of information, i.e., the number of data units forming those waves, may alter during a computation process, and just one data unit, for example by LOAD $= x$, will be considered to be the simplest case of a wave of information.

Uniform cost criterion. For measuring the time complexity of computations, we assume that any (basic) instruction of the SIMD system needs one unit of time for performance on this system.

Definition 1. A model of computation SYS is called a *standard off-line network system* (SYS$\in$OFF-NET) iff SYS is defined by

- a CPU and a fixed set of indexed PEs, with concrete values for D_{CPU} and D_{PE},
- abstract registers if not otherwise specified, and the standard register enumeration,
- a uniform interconnection network with $0 \leqq N_{\text{IN}} < \infty$,
- the standard PE masking scheme,
- a special interpretation and selection of instructions of the abstract CPU instruction set where

(OFF. 1) no READ and WRITE instructions are contained in the instruction set of SYS,

(OFF. 2) for the CPU all RAM instructions [2, Fig. 1.5] except READ and WRITE are avilable,

(OFF. 3) for $N_{\text{IN}} = p \geqq 1$ at least one instruction of the type [all PE's] OP_{p+1}: $0, ., ..., p-1$ is available, and

(OFF. 4) for any output register $(j, 0)$, i.e., accumulator of PE(j), at least one instruction of the type $\text{OP}_2(j)$ is available, i.e., the CPU may have control of any outputting PE,

- the off-line I/O convention, and
- the uniform cost criterion.

For the defined class OFF-NET we may define subclasses — e.g., OFF-NET_p to be the set of all SYS$\in$OFF-NET having the branching degree $p = N_{\text{IN}}$, OFF-SQUARE to be the set of all SYS$\in$OFF-NET having a square network as defined in Table 1, OFF-BINTREE with the same reference of Table 1, OFF-PS $= \bigcup_{m=1}^{\infty} \text{OFF-PS}^m$, or just OFF-RAM.

Example 1. Let us consider the following special SIMD system EXAMP1$\in$ $\in$OFF-SQUARE. Let $D_{\text{CPU}} = D_{\text{PE}} = \infty$. Additionally to the CPU registers $0, 1, ..., L-1$ for a certain $L \geqq 1$, all the accumulators $(j, k, 0)$, $0 \leqq j < M$ and $0 \leqq k < N$ for some $M, N \geqq 1$, are fixed as input and output registers of EXAMP1. The system possesses the following instruction set:

[mask] ADD α, α for $m, {}^*m, :i_1, \ldots, i_l$ for $i_1, \ldots, i_l \in \{0, 1, 2, 3\}$,
[mask] $OP_l\ \alpha, \alpha$ for $m, {}^*m, :i$ for $i \in \{0, 1, 2, 3\}$, $l = 1, 2$,
[mask] LOAD α, α for $m, {}^*m, :i$ for $i \in \{0, 1, 2, 3\}$,
[mask] STORE α, α for $m, {}^*m, :i_1, \ldots, i_l$ for $i_1, \ldots, i_l \in \{0, 1, 2, 3\}$,
LOAD α, α for $=x, m, {}^*m, (j, k)$,
STORE α, α for $m, {}^*m, (j, k)$,
$OP_2\ \alpha, \alpha$ for $=x, m, {}^*m, (j, k)$,
JUMP b, JGTZ b, JZERO b, JLTZ b, and HALT.

Here, [mask] represents an arbitrary PE address mask, OP_1 is ABS (absolute value) or SIGN (signum function), OP_2 is ADD, SUB, MULT, or DIV, for the tuples (j, k) with $0 \leqq j < M$ and $0 \leqq k < N$.

To give a short illustration of the computing power of EXAMP1 let us consider the computation of the *parallel Roberts gradient* (cp. [9] for its importance to digital image processing), where the input image $A = (a_{jk})$ of size $M \times N$ is assumed to be stored in the PE input registers (a_{jk} in register $(j, k, 0)$) at the beginning of the computation. At the end of the computation, value $\max \{|a_{jk} - a_{j+1,k+1}|, |a_{j+1,k} - a_{j,k+1}|\}$ has to be present in register $(j, k, 0)$.

By performing the following sequence pf parallel instructions,

1. [all PEs] STORE 1
2. [all PEs] LOAD :2
3. [all PEs] STORE 2
4. [all PEs] LOAD :1
5. [all PEs] SUB 1
6. [all PEs] ABS 0
7. [all PEs] STORE 3
8. [all PEs] LOAD 1
9. [all PEs] LOAD :1
10. [all PEs] SUB 2
11. [all PEs] ABS 0
12. [all PEs] STORE 4

all registers $(j, k, 3)$ contain value $|a_{jk} - a_{j+1,k+1}|$, and all registers $(j, k, 4)$ contain value $|a_{j+1,k} - a_{j,k+1}|$, for $0 \leqq j < M$ and $0 \leqq k < N$. These values may be considered as two $M \times N$ matrices B and C. For $\max(B, C) = (\max \{b_{jk}, c_{jk}\})$ we have

$$\max(B, C) = B \times \operatorname{sign}(B - C) + C \times \operatorname{sign}(C - B) + B - B \times \operatorname{sign}|B - C|, \quad (1)$$

where $\times$ means the parallel MULT operation (cross product of two matrices), and sign the parallel SIGN operation. Using this formula, the parallel Roberts gradient may be computed on the defined special OFF-SQUARE system within time 29 or less, independent of the values of M and N, as the reader may check easily. Note that formula (1) describes a way in which the PEs are able to perform local logical decisions in SIMD mode.

Example 2. By some easily described modifications, the system EXAMP1 may be altered dramatically. Replace the square network by LRUD2I^m, for $m < \max \{\log_2 M, \log_2 N\}$, let $W_{PE} = 1$, and replace the parallel operations ADD, OP_1 and OP_2 by logical operations AND, NOT, and OR, respectively. What results is a special OFF-LRUD2I^m system EXAMP2 which essentially coincides with the PBS (paralleles Binärbildverarbeitungssystem). The computational power of the PBS was extensively studied in [4].

Definition 2. A model of computation SYS is called a *standard on-line network system* (SYS$\in$ON-NET) iff SYS is defined by

- a CPU and a fixed set of indexed PEs, with concrete values for D_{CPU} and D_{PE},
- abstract registers if not otherwise specified, and the standard register enumeration,
- a uniform interconnection network with $0 \leqq N_{IN} < \infty$,
- the standard PE masking scheme,
- a special interpretation and selection of instructions of the abstract CPU instruction set where, for $N_{IN} \geqq 2$, an integer tuple (p, q) may be denoted to be the *characteristic* of SYS in the following sense:

(ON. 1) $P = N_{IN}$ and $1 \leqq q < p$,
(ON. 2) a proper subset $\{i_1, i_2, \ldots, i_q\}$ of all directions $\{0, 1, \ldots, p-1\}$ is specified,
(ON. 3) at least one instruction of the type [all PE's] $OP_{q+1} : i_1, i_2, \ldots, i_q$ is avaible,
(ON. 4) for any of the instructions [mask] LOAD : j or [mask] $OP_{k(+1)}: j_1, j_2, \ldots, j_k$, $k \geqq 1$, it follows that $j, j_1, j_2, \ldots, j_k \in \{i_1, i_2, \ldots, i_q\}$,
(ON. 5) for any of the instructions [mask] STORE : $j_1, j_2, \ldots, j_k$, $k \geqq 1$, it follows that $j_1, j_2, \ldots, j_k \in \{0, 1, \ldots, p-1\} - \{i_1, i_2, \ldots, i_q\}$, i.e., the result sof consecutive parallel operations may be shifted through the system in directions $\{0, 1, \ldots, p-1\} - \{i_1, i_2, \ldots, i_q\}$ only, and, furthermore
(ON. 6) for the CPU all RAM instructions are avilable including READ and WRITE,
(ON.7) for any output register $(j, 0)$, at least one instruction of the type $OP_2(j)$ is available,

- the on-line I/O convention, and
- the uniform cost criterion.

For the defined class ON-NET we may define subclasses — e.g., $\text{ON-NET}_{p,q}$ to be the set of all ON-NET systems with characteristic (p, q), ON-LR2I^m to be the set of all SYS$\in$ON-NET having a left-right 2^i network as defined in Table 1, ON-ILLIAC^m with the same reference to Table 1, $\text{ON-PM2I} = \bigcup_{m=1}^{\infty} \text{ON-PM2I}^m$, or just ON-RAM.

Any infinite network class OFF-LINEAR or ON-DIAGONAL may be considered as an abstraction of a finite network system, or as the union of classes of finite network systems in the following way.

Definition 3. Let OFF-IN be the set of all OFF-NET systems which are defined by a special infinite network IN, e.g., IN=LINEAR or IN=LRUD2I^m. A model of computation SYS is called a *finite OFF-IN system* (SYS$\in$FIN-OFF-IN) iff there exists a system $SYS_0 \in$OFF-IN such that SYS may be obtained as a restriction of SYS_0 in the following sense:

Let $\mathbf{ind}_0$ and D^0_{PE} be the PE index set and the PE memory depth for SYS_0, respectively. A finite cut-off of the PE register set of SYS_0 is defined by a certain finite subset **ind** of $\mathbf{ind}_0$ and a (possibly infinite) memory depth $D_{PE} \leqq D^0_{PE}$. The work of SYS may be described as follows. All registers in a certain finite cut-off of SYS_0 are available in SYS but all registers not in this finite cut-off will be considered to be dummy registers, i.e., they are assumed to store value zero if addressed as an operand, and to "forget" any value handed over to them; this is the only difference between SYS_0 and SYS.

Analogously the set FIN-ON-IN may be defined.

Example 3. An example of a FIN-ON-BINTREE system may be specified as follows. Let $D_{CPU}=\infty$ and $D_{PE}=m\geqq 2$. The finite cut-off of the bintree network is given by $\mathbf{ind}=\{1, 2, ..., 2^m-1\}$. Additionally to the CPU accumulator which acts as an input and output register $(L=1)$, the registers $(2^{m-1}, 0)$, $(2^{m-1}+1, 0), ..., (2^m-1, 0)$, i.e., the accumulators of the 2^{m-1} leaf node PEs, are fixed as input registers, and register $(1, 0)$, i.e., the accumulator of the top node PE, is fixed as an output register. The system possesses the following instruction set:

[mask] ADD α, α for $m, {}^*m, :1, :2, :1, 2$,
[mask] $OP_l\ \alpha, \alpha$ for $m, {}^*m, :1, :2$ and $l=1, 2$,
[mask] LOAD α, α for $m, {}^*m, :1, :2$,
[mask] STORE α, α for $m, {}^*m, :0$,
[subset leaf nodes] READ 0,
[top node] WRITE 0,
LOAD α, α for $=x, m, {}^*m, (1)$,
STORE α, α for $m, {}^*m, (1)$,
$OP_l\ \alpha, \alpha$ for $=x, m, {}^*m, (1)$, and $l=1, 2$,
READ 0,
WRITE α, α for $=x, 0$,
JUMP b, JGTZ b, JZERO b, JLTZ b, HALT.

Here, [mask] represents an arbitrary PE address, OP_1 either ABS or SIGN, OP_2 one of the operation codes ADD, SUB, MULT, or DIV. Altogether, a FIN-ON-BINTREE system EXAMP3 is defined which may be obtained by a restriction of an infinite ON-BINTREE model where infinite sets of input and output PE registers are available in the infinite origin.

To give a short illustration of the computational power of the system EXAMP3 let us consider the computation of the *arithmetical average* $\frac{1}{N}\sum_{i=0}^{N-1} a_i$, $N=2^{n-1}$ and n odd, *for M consecutive waves of information* $(a_0, a_1, ..., a_{N-1})$ where a_i is fed to the accumulator of the PE $(2^{n-1}+i)$, for $i=0, 1, ..., N-1$. In order of the M consecutive waves of information the arithmetical average have to leave the system via register $(1, 0)$.

For initialization of the system, at first the instruction LOAD $=N$, STORE (1), [top node] STORE 1 will be performed in this order. For $M\geqq(n-1)/2$ the following sequence of instructions is executed $(n-1)/2$ times:

[leaf nodes] READ 0,
[all PEs] ADD : 1, 2,
[leaf nodes] LOAD 1,
[all PEs] ADD : 1, 2,

followed by the following sequence of instructions which is executed $M-[(n-1)/2]$ times:

[top node] DIV 1,
[top node] WRITE 0,
[leaf nodes] READ 0,
[all PEs] ADD : 1, 2,
[leaf nodes] LOAD 1,
[all PEs] ADD : 1, 2.

Finally, the following sequence of instructions is executed $(n-3)/2$ times:

[top node] DIV 1,
[top node] WRITE 0,
[all PEs] ADD : 1, 2,
[all PEs] ADD : 1, 2,

followed by the last two instructions [top node] DIV 1 and [top node] WRITE 0. Thus, altogether, the arithmetic averages of $M \cong (n-1)/2$ consecutive waves of information $(a_0, a_1, ..., a_{N-1})$ may be computed within $6M+n$ basic operations of EXAMP3, instead of $O(N \cdot M)$ basic operations in the serial case using a RAM as model for computation.

In conclusion, we point out that SIMD now denotes not a general concept (single-instruction, multiple data) but an exactly defined class of models for computation, namely the union of all system classes given by Definitions 1, 2, and 3.

2. Local, global, and total data flow measures

Let SYS$\in$SIMD; throughout this paper such a special parallel processing system will be used as a standard system for considerations of data transfer restrictions in computing systems. Any computational process performed on such a model SYS may be uniquely specified by a CPU program π and a concrete input situation I characterized by the placement of input values into the set of input registers if off-line mode is used, or by the partition of the input data into consecutive waves of information fed to some of the input registers of the system from the outside world if on-line mode is used.

As suggested by applications to visual perception, the set of input registers of the model SYS may be considered as the *retina* of the system, and any new wave of information to this set of input registers represents a snapshot of the outside world. In this sense, after t steps of a computational process characterized by a program π and an input situation I, for any register r of the system we may mark out a certain receptive field $\mathrm{rec}_\pi^I(r, t)$ containing all the names of those input registers which have had any influence on the contents of register r up to the moment t, where new waves of information to the retina of the system create new names of the input registers, formally represented by $r^{(0)}, r^{(1)}, r^{(2)}, ..., r^{(i)}, ...$ for register r.

Standard register names. At time $t=0$ of any computational process, each register r in our standard enumeration possesses the name $r^{(0)}$. At $t=0$ let the wave number $WN=0$ also. At time $t+1$ assume that a serial or parallel READ instruction, or an instruction LOAD$=x$, $\mathrm{OP}_1=x$, or $\mathrm{OP}_2=x$ has to be performed. Then, by this operation we obtain $WN \leftarrow WN+1$ and the new names $r^{(WN)}$ for all registers which were addressed by these instructions. For example, the number $(j, c(j, m))^{(WN)}$ in the case of an instruction [mask] READ $*m$ for all activated processing elements PE(j), where $c(j, m)$ denotes the actual contents of register (j, m), or the name $0^{(WN)}$ in the case of an instruction $\mathrm{OP}_2=x$.

Definition 4. Let SYS$\in$SIMD. Standard register names are assumed. For a program π of SYS, an input situation I of SYS, a register r of SYS, and an arbitrary moment $t \cong 0$, the *receptive field* $\mathrm{rec}_\pi^I(r, t)$ is recursively defined as follows:

moment $t=0$:

$$\mathrm{rec}^I_\pi(r,0)=\begin{cases}\{r^{(0)}\} & \text{if input register } r \text{ stores an input value according to } I, \text{ for off-line mode,}\\ & \text{empty set, otherwise}\end{cases}$$

moment $t+1$, $t\geqq 0$:
At moment $t+1$ a certain instruction has to be applied according to π and I, or the HALT instruction is assumed for this moment.

(i) Depending on this instruction, if it is one of those listed in Table 3, the changes of receptive fields are defined as given in this Table where we omit the indices π and I for simplification of the expressions. In the case of parallel instructions, the mentioned changes are valid for all activated PEs PE (j) where j matches [mask].

Table 3. Changes of receptive fields in step $t+1$

Instructions	Changes of receptive fields
[mask] OP_1 m	$\mathrm{rec}((j,0),t+1)=\mathrm{rec}((j,m),t)$
[mask] OP_1 $*m$	$\mathrm{rec}((j,0),t+1)=\mathrm{rec}((j,m),t)\cup\mathrm{rec}((j,c(j,m)),t)$
[mask] OP_1 $:i$	$\mathrm{rec}((j,0),t+1)=\mathrm{rec}((f_i(j),0),t)$
[mask] OP_2 m	$\mathrm{rec}((j,0),t+1)=\mathrm{rec}((j,0),t)\cup\mathrm{rec}((j,m),t)$
[mask] OP_2 $*m$	$\mathrm{rec}((j,0),t+1)=\mathrm{rec}((j,0),t)\cup$ $\cup\mathrm{rec}((j,m),t)\cup\mathrm{rec}((j,c(j,m)),t)$
[mask] $OP_{l+1}:i_1,i_2,\ldots,i_l$	$\mathrm{rec}((j,0),t+1)=\mathrm{rec}((j,0),t)\cup\mathrm{rec}((f_{i_1}(j),0),t)\cup$ $\cup\mathrm{rec}((f_{i_2}(j),0),t)\cup\ldots\cup\mathrm{rec}((f_{i_l}(j),0),t)$
[mask] STORE m	$\mathrm{rec}((j,m),t+1)=\mathrm{rec}((j,0),t)$
[mask] STORE $*m$	$\mathrm{rec}((j,c(j,m),t+1)=\mathrm{rec}((j,0),t)\cup\mathrm{rec}((j,m),t)$
[mask] STORE $:i_1,i_2,\ldots,i_l$	$\mathrm{rec}((f_{i_1}(j),0),t+1)=\mathrm{rec}((j,0),t)$, $\mathrm{rec}((f_{i_2}(j,0),t+1)=$ $=\mathrm{rec}((j,0),t)$, ..., $\mathrm{rec}((f_{i_l}(j),0),t+1)=\mathrm{rec}((j,0),t)$
[mask] READ m	$\mathrm{rec}(j,m),t+1)=\{(j,m)^{(WN)}\}$
[mask] READ $*m$	$\mathrm{rec}((j,c(j,m),t+1)=\mathrm{rec}((j,m),t)\cup\{(j,c(j,m))^{(WN)}\}$
$OP_1=x$	$\mathrm{rec}(0,t+1)=\{0^{(WN)}\}$
OP_1 m	$\mathrm{rec}(0,t+1)=\mathrm{rec}(m,t)$
OP_1 $*m$	$\mathrm{rec}(0,t+1)=\mathrm{rec}(m,t)\cup\mathrm{rec}(c(m),t)$
OP_1 (j)	$\mathrm{rec}(0,t+1)=\mathrm{rec}((j,0),t)$
$OP_2=x$	$\mathrm{rec}(0,t+1)=\mathrm{rec}(0,t)\cup\{0^{(WN)}\}$
OP_2 m	$\mathrm{rec}(0,t+1)=\mathrm{rec}(0,t)\cup\mathrm{rec}(\mathrm{m},t)$
OP_2 $*m$	$\mathrm{rec}(0,t+1)=\mathrm{rec}(0,t)\cup\mathrm{rec}(m,t)\cup\mathrm{rec}(c(m),t)$
OP_2 (j)	$\mathrm{rec}(0,t+1)=\mathrm{rec}(0,t)\cup\mathrm{rec}((j,0),t)$
STORE m	$\mathrm{rec}(m,t+1)=\mathrm{rec}(0,t)$
STORE $*m$	$\mathrm{rec}(c(m),t+1)=\mathrm{rec}(0,t)\cup\mathrm{rec}(m,t)$
STORE (j)	$\mathrm{rec}((j,0),t+1)=\mathrm{rec}(0,t))$
READ m	$\mathrm{rec}(m,t+1)=\{m^{(WN)}\}$
READ $*m$	$\mathrm{rec}(c(m),t+1)=\mathrm{rec}(m,t)\cup\{c(m)^{(WN)}\}$

(ii) For the parallel or serial LOAD instructions the changes of receptive fields are the same as for the corresponding OP_1 instructions.

(iii) In the case of a WRITE, JUMP, or HALT instruction no changes of receptive fields appear.

(iv) In the case of a JGTZ, JZERO, or JLTZ instruction no changes of receptive fields appear in step $t+1$, but the set $\mathrm{rec}(0, t)$ will be added at moment $t' \geqq t+2$ to any receptive field that alters at moment t' according to (i) or (ii), if at moment t' an instruction has to be performed covered by cases (i) and (ii). For example, the instruction [mask] OP_2 m, at moment $t' \geqq t+2$, will produce the changes $\mathrm{rec}((j, 0), t') = \mathrm{rec}((j, 0), t'-1) \cup \mathrm{rec}((j, m), t'-1) \cup \mathrm{rec}(0, t)$ for all activated PEs.

For illustration of this definition, consider the special OFF-SQUARE system as defined in Example 1. Let I be any concrete input situation for computing the parallel Roberts gradient and let π be the sequence of the 12 parallel instructions as given there. At moment $t=0$ we have $\mathrm{rec}((j, k, 0), 0) = \{(j, k, 0)^{(0)}\}$, for $0 \leqq j < M$ and $0 \leqq k < N$, and for any other register r of the system EXAMP 1, $\mathrm{rec}(r, 0)$ is the empty set. After performing the 12 instructions of π the reception fields of maximal cardinality 2 belong to the registers $(j, k, 0)$, $(j, k, 3)$ and $(j, k, 4)$, for $0 \leqq j \leqq M-2$ and $0 \leqq k \leqq N-2$, where, e.g., $\mathrm{rec}((j, k, 0), 12) = \{(j+1, k, 0)^{(0)}, (j, k+1, 0)^{(0)}\}$. For the system defined in Example 3, and the program and the input situation as described there, after performing the $6M+n$ instructions the receptive field of maximal cardinality $NM+1$ belongs to the register $(1, 0)$, i.e., to the accumulator of the top node PE.

Definition 5. Let SYS$\in$SIMD. For a set R of registers of SYS and a moment $t \geqq 0$ define the *local data transfer function* λ_{SYS} by

$$\lambda_{SYS}(R, t) = \max_{\pi} \max_{I} \max_{r \in R} \mathrm{card}\,(\mathrm{rec}^I(r, t)),$$

the *global data transfer function* γ_{SYS} by

$$\gamma_{SYS}(R, t) = \max_{\pi} \max_{I} \mathrm{card}\,\Big(\bigcup_{r \in R} \mathrm{rec}^I_\pi(r, t)\Big),$$

the *total data transfer function* τ_{SYS} by

$$\tau_{SYS}(R, t) = \max_{\pi} \max_{I} \sum_{r \in R} \mathrm{card}\,(\mathrm{rec}^I_\pi(r, t)).$$

By this definition, it follows immediately that the functions λ_{SYS}, γ_{SYS} and τ_{SYS} are monotonically increasing for any set R of registers of SYS and increasing values of t. Furthermore,

$$\lambda_{SYS}(R, t) \leqq \gamma_{SYS}(R, t) \leqq \tau_{SYS}(R, t) \tag{2}$$

for all models SYS$\in$SIMD, sets R of registers and moments $t \geqq 0$. Also note that for any model SYS, if within t steps of an arbitrary program π for SYS starting with an arbitrary input situation I for SYS at most $\omega_{SYS}(t)$ input data may be fed to the system, then

$$\gamma_{SYS}(R, t) \leqq \omega_{SYS}(t), \quad \text{and} \tag{3.1}$$

$$\tau_{SYS}(R, t) \leqq \lambda_{SYS}(R, t) \cdot \mathrm{card}\,(R), \tag{3.2}$$

for any set R of registers of SYS and $t \geqq 0$.

Example 4. In Section 4 we shall characterize the way to use these data transfer functions for obtaining lower time bounds for concrete computational problems. For serial data processing we shall apply the system RAM_L, cp. [2, Fig. 1.5], as model for computation, where $R_L=\{0, 1, 2, \ldots, L-1\}$, $L\geqq 1$, is assumed to be the set of all input/output registers of such a machine ($D_{\mathrm{CPU}}=\infty$, $N_{\mathrm{PE}}=0$, $W_{\mathrm{CPU}}=\infty$). For $t\geqq 0$, we have $\omega_{\mathrm{OFF-RAM}_L}(t)=L+t$ and $\omega_{\mathrm{ON-RAM}_L}(t)=t$. For OFF-RAM$=\bigcup_{L=1}^{\infty}$ OFF-RAM$_L$, note that $\omega_{\mathrm{OFF-RAM}}(t)=\max_L \omega_{\mathrm{OFF-RAM}_L}(t)$ is not defined. Furthermore, we have

$$\lambda_{\mathrm{OFF-RAM}_L}(R_L, t)=\begin{cases}2t+1 & \text{for } 0\leqq t\leqq \lfloor (L-1)/2\rfloor\\ \lfloor (L+1)/2\rfloor+t, & \text{otherwise,}\end{cases} \tag{4.1}$$

$$\gamma_{\mathrm{OFF-RAM}_L}(R_L, t)=L+t, \quad \text{and} \tag{4.2}$$

$$\tau_{\mathrm{OFF-RAM}_L}(R_L, t)=L(t-\lfloor L/2\rfloor+1) \quad \text{for} \quad t\geqq \lfloor L/2\rfloor, \tag{4.3}$$

in the case of using the RAM_L in off-line mode, and

$$\lambda_{\mathrm{ON-RAM}_L}(R_L, t)=\gamma_{\mathrm{ON-RAM}_L}(R_L, t)=t,$$

$$\tau_{\mathrm{ON-RAM}_L}(R_L, t)=\begin{cases}t(t+1)/2 & \text{for } t\leqq L\\ L(t-(L/2)+1/2) & \text{for } t\geqq L,\end{cases} \tag{4.5}$$

in the case of using the RAM_L in on-line mode. The maximal data flow for obtaining equation (4.1) is possible by indirect addressing $\mathrm{OP}_2{}^*m$, followed by $\mathrm{OP}_2=x$ operations. For (4.3), the same sequence of operations is extended by $L-1$ instructions STORE m. For (4.4), t operations of the type $\mathrm{OP}_2=x$ may be considered. For small t the exact derivation of the function $\tau_{\mathrm{OFF-RAM}_L}$ represents a sophisticated problem already, for this quite simple model of serial computation.

Example 5. For further illustration of the concrete derivation of these data transfer functions, let us consider both systems EXAMP1 and EXAMP3 as defined above.

For the system EXAMP1, first we see that $\omega_{\mathrm{EXAMP1}}(t)=MN+L+t$, for $t\geqq 0$. Let $R_{M,N}$ be the set $\{(j, k, 0) : 0\leqq j<M \text{ and } 0\leqq k<N\}$ of all PE input/iutput registers of the system. By using t operations of the type

[all PE's] ADD :0, 1, 2, 3

we obtain the maximal local and total data transfer within the field of PE accumulators, where

$$\lambda_{\mathrm{EXAMP1}}(R_{M,N}, t)=2t^2+2t+1, \tag{5.1}$$

$$(2t^2+2t+1)MN-\left(\frac{t+1}{3}-(t+1)^2+\frac{2(t+1)^3}{3}\right)(M+N)\leqq$$

$$\leqq \tau_{\mathrm{EXAMP1}}(R_{M,N}, t)\leqq (2t^2+2t+1)MN, \tag{5.2}$$

for $2t+1\leqq \min\{M, N\}$, by elementary combinatorial considerations and (3.2). For $t\geqq t_0=\lfloor M/2\rfloor\cdot\lfloor N/2\rfloor$ we have

$$MN+(t-t_0)\leqq \lambda_{\mathrm{EXAMP1}}(R_{M,N}, t)\leqq MN+L+t. \tag{4.3}$$

For $t \geqq t_0 = M+N-2$ we can easily see that

$$M^2N^2+(t-t_0) \leqq \tau_{\text{EXAMPI}}(R_{M,N}, t) \leqq MN(MN+L+t). \tag{5.4}$$

Finally, for the case of global data transfer we obtain

$$\gamma_{\text{EXAMPI}}(R_{M,N}, t) = \begin{cases} MN & \text{for } t = 0 \\ MN+2t+1 & \text{for } 2t+1 \leqq L \text{ and } t > 0 \\ MN+\lfloor (L-1)/2 \rfloor + t & \text{for } 2t+1 > L \end{cases} \tag{5.5}$$

where, for $2t+1 \leqq L$, the maximal global data transfer is possible by t operations of the type ADD $*m_t$ and one operation STORE (j, k), e.g.

For the system EXAMP3, at first we have $\omega_{\text{EXAMP3}}(t) = t \cdot N$, for $N = 2^{n-1}$ and $t \geqq 0$ by using t operations of the type

[leaf nodes] READ 0.

Let $R_0 = \{0, (1, 0)\}$ be the set of the two distinguished output registers of this syste EXAMP3. By using the instruction pair

[leaf nodes] READ 0,
[all PEs] ADD :1, 2

repeated $(m-1)$ times, $m \geqq 1$; the single instruction

[leaf nodes] READ 0

again; and finally $(n-1)$ instructions

[all PEs] ADD : 1, 2,

we obtain the maximal local data transfer for register $(1, 0)$ in any case $t \geqq m$. We have

$$\lambda_{\text{EXAMP3}}(R_0, t) = \begin{cases} 0 & \text{for } t = 0 \\ 2^{t-1} & \text{for } 1 \leqq t \leqq n-1 \\ m \cdot N & \text{for } t = n+2m-l, \quad m \geqq 1 \\ & \text{and } l = 1 \text{ or } l = 2, \end{cases}$$

for all $t \geqq 0$. Analogously, for the same set R_0 and $t \geqq 0$

$$\gamma_{\text{EXAMP3}}(R_0, t) = \begin{cases} 0 & \text{for } t = 0, \\ 2^{t-1} & \text{for } 1 \leqq t \leqq n-1, \\ m \cdot N & \text{for } t = n+2m-2, \quad m \geqq 1, \\ m \cdot N+1 & \text{for } t = n+2m-1, \quad m \geqq 1, \end{cases}$$

$$\tau_{\text{EXAMP3}}(R_0, t) \begin{cases} 0 & \text{for } t = 0, \\ 2^{t-1} & \text{for } 1 \leqq t \leqq n+1, \\ 2m \cdot N & \text{for } t = n+2m-1, \quad m \geqq 1. \\ 2m \cdot N+1 & \text{for } t = n+2m, m \geqq 1. \end{cases}$$

Of course, the values of $\lambda_{\text{EXAMF3}}, \gamma_{\text{EXAMP3}}$, and τ_{EXAMP3} depend on the choice of the set R_0, and may be quite different for some other sets of registers.

Definition 6. Let CLASS $\subseteq$ SIMD. The *general data transfer functions* are defined as follows, for such a set CLASS of models of computation, for $t, n \geqq 0$:

$\Lambda_{\text{CLASS}}(t)$ denotes the maximal value of all $\lambda_{\text{SYS}}(R, t)$,

$\Gamma_{\text{CLASS}}(n, t)$ denotes the maximal value of all $\gamma_{\text{SYS}}(R, t)$ with card $(R)=n$, and $T_{\text{CLASS}}(n, t)$ denotes the maximal value of all $\tau_{\text{SYS}}(R, t)$ with card $(R)=n$, where SYS is an arbitrary element of CLASS, and R denotes a set of registers of SYS.

Interesting examples of CLASS are sets like OFF-NET$_p$, ON-NET$_{p,q}$, OFF-SQUARE, OFF-BINTREE, or ON-HEXAGONAL, where these general data transfer functions are fully defined.

Theorem 1. For standard off-line network systems and $2 \leqq p < \infty$ we have

$$\Lambda_{\text{OFF-NET}_p}(t) = \begin{cases} 2t+1 & \text{for } p = 2 \\ p\left(\dfrac{(p-1)^t-1}{p-2}\right)+1 & \text{for } p \geqq 3, \end{cases}$$

and

$$\Gamma_{\text{OFF-NET}_p}(n, t) = T_{\text{OFF-NET}_p}(n,t) = n \cdot \Lambda_{\text{OFF-NET}_p}(t), \quad \text{for } n, t \geqq 0.$$

Proof. First, let us consider the local situation. For $p=2$, the maximal transfer of data units is possible by indirect addressing to the CPU accumulator, e.g. For $p \geqq 3$, there exist special OFF-NET$_p$ models SYS$_t$ such that, according to (OFF.3), at any moment $1 \leqq s \leqq t$ the maximal possible number of $p(p-1)^{s-1}$ new names of input registers may enter the receptive field of a certain register r, for $t \geqq 0$. Thus,

$$\lambda_{\text{SYS}_t}(\{r\}, t) = 1 + \sum_{s=0}^{t-1} p(p-1)^s = p\left(\frac{(p-1)^t-1}{p-2}\right)+1.$$

For the total and global situation note that by choosing sufficiently complex SYS$_{n,t}$, for $n, t \geqq 0$, the maximal local situations of data transfer characterized by receptive fields of cardinality $\Lambda_{\text{OFF-NET}_p}(t)$ at moment t may appear in n different registers and time t such that these registers are far enough from one another so that their receptive fields are pairwise disjoint. □

Example 6. By (4.1) and Theorem 1, it follows that $\Lambda_{\text{OFF-RAM}}(t) = \Lambda_{\text{OFF-NET}_2}(t) = 2t+1$, for $t \geqq 0$. Of course, this coincidence is not true in the total and global cases. According to Theorem 1 we have $\Gamma_{\text{OFF-NET}_2}(n, t) = T_{\text{OFF-NET}_2}(n, t) = n(2t+1)$, for $n, t \geqq 0$, but by elementary considerations $\Gamma_{\text{OFF-RAM}}(n, t) = 2t+n$, for $n \geqq 1$ and $T_{\text{OFF-RAM}}(n, t) = 2n(t-n+2)-2$, for $t \geqq n \geqq 2$.

In Table 4 the general local data transfer functions are collected for some classes of off-line systems as defined in Section 1. For these classes, the functions $\Lambda_{\text{OFF-NET}_p}$ as given in Theorem 1 act as upper bounds, where the proper value of p has to be specified. The classes OFF-LINEAR, OFF-PS, OFF-BINTREE and OFF-QUADTREE represent examples for the maximal transfer situations as characterized by Theorem 1, for $p=2, 3, 5$, respectively.

Some remarks about Table 4 and about the other networks which were defined in Table 1.

1. For the bintree, triangle and quadtree network note that the maximal receptive fields may be obtained for central nodes of these tree structures only, and not at the top node. The maximal possible cardinalities of receptive fields of top node accumulators are given for illustration of this fact.

Table 4. General local data transfer functions for offline systems

CLASS	P	$\Lambda_{\text{OFF-CLASS}}(t)$	$t=4$	$t=8$
LINEAR	2	$2t+1$	9	17
HEXAGONAL	3	$\frac{3}{2}t^2+\frac{3}{2}t+1$	31	109
SQUARE or ILLIAC	4	$2t^2+3t+1$	41	145
TRIAGONAL	6	$3t^2+3t+1$	61	215
DIAGONAL	8	$4t^2+4t+1$	81	289
PS	3	$3\cdot 2^t-2$	46	766
BINTREE	3	$3\cdot 2^t-2$	46	766
top node		$2^{t+1}-1$	31	511
TRIANGLE	5	$3\cdot 2^{t+1}+t^2-2t-5$	99	1,579
top node		$2^{t+1}-1$	31	511
QUADTREE	5	$(5\cdot 4^t-2)/3$	426	109,226
top node		$(4^{t+1}-1)/3$	341	87,381

2. For all examples of CLASS given in Table 4, we have $\Gamma_{\text{OFF-CLASS}}(n,t)=$ $=T_{\text{OFF-CLASS}}(n,t)=n\cdot\Lambda_{\text{OFF-CLASS}}(t)$, for $n,t\geqq 0$.

3. The hexagonal, square, triagonal, and diagonal networks are special examples of infinite graphs of constant degree p such that the general local data transfer function is equal to $\frac{p}{2}t^2+\frac{p}{2}t+1$. Such networks correspond to usual digital metrics for the orthogonal grid in a natural way, e.g., the metrics d_4 or d_8 as used in digital image processing, cp. [9], to the square or diagonal network, respectively.

4. For the networks CUBE^m, PM2I^m, WPM2I^m, LR2I^m, or LRUD2I^m, the derivation of the three general data transfer functions represents a very sophisticated problem. Of course, the values of these functions depend on the value of m, and the consideration of classes like

$$\text{CUBE} = \bigcup_{m\geqq 2} \text{CUBE}^m$$

would lead to undefined general data transfer functions. In [4] the general local data transfer functions were analyzed for some concrete SIMD systems similar to FIN-OFF-LR2I^m or FIN-OFF-LRUD2I^m systems like EXAMP2 which was defined above. But, for the present paper, we recommend data transfer analysis for specialized (finite) SIMD systems to the interested reader, and are satisfied with some hints:

CUBE^m: For this system, the exact derivation of the local transfer function should be a solvable task. We have

$$\Lambda_{\text{OFF-CUBE}^m}(t)\begin{cases}=\sum_{i=0}^{t}\binom{m}{i} & \text{for } t<m\\ \geqq 2^m & \text{for } t=m\\ \geqq 2^{m+1}(t-m) & \text{for } t>m.\end{cases}$$

For example, we have $\Lambda_{\text{OFF-CUBE}}256\,(4)=177{,}589{,}057$ and $\Lambda_{\text{OFF-CUBE}}256(8)$ is about $4\cdot 10^{14}$.

PM2I^m: For this, as for the other "power-of-two systems", the analysis of data flow represents quite a hard problem, cp. [4]. But, to give the reader some feeling about the complexity of the data transfer functions for these systems, some values will be collected:

$$\Lambda_{\text{OFF-PM2I}^m}(t)\begin{cases} = 1 & \text{for } t = 0 \\ = 2 & \text{for } t = 1 \\ = 2(m-1)(m-2)+4 & \text{for } t = 2 \\ \vdots & \vdots \\ \geqq 2^m & \text{for } t = \lceil m/2 \rceil \\ \geqq 2^{m+1}(t-\lceil m/2 \rceil) & \text{for } t > \lceil m/2 \rceil. \end{cases}$$

Note that exponential increase changes to linear increase at $t=\lceil m/2 \rceil$.

WPM2I^m: It may be that this is the most complicated situation of any network; we have

$$\Lambda_{\text{OFF-WPM2I}^m}(t)\begin{cases} = 1 & \text{for } t = 0 \\ = 2 & \text{for } t = 1 \\ \vdots & \vdots \\ \geqq 2^m & \text{for } t = \lceil m/2 \rceil \\ \geqq 2^{m+1}(t-\lceil m/2 \rceil) & \text{for } t \geqq \lceil m/2 \rceil. \end{cases}$$

This great difficulty in analyzing data paths should be a hint to the limited practical importance of this network.

LR2I^m: For brevity we shall use the function $\sigma(i)=\sum_{j=1}^{i} j^2=\frac{1}{6}(i+1)-\frac{1}{2}(i+1)^2+$ $+\frac{1}{3}1(i+1)^3$. We found the following interesting values:

$$\Lambda_{\text{OFF-LR2I}^m}(t)=\begin{cases} 1 & \text{for } t = 0 \\ 2m+1 & \text{for } t = 1 \\ 2(m-2)^2+4m+1 & \text{for } t = 2 \\ 1+6m+4(m-2)^2+2\cdot\sigma(m-4) & \text{for } t = 3 \\ 1+8m+6(m-2)^2+4\cdot\sigma(m-4)+ & \\ \quad +4\cdot\sum_{i=1}^{m-6}\sigma(i) & \text{for } t = 4 \\ 1+10m+8(m-2)^2+6\cdot\sigma(m-4)+ & \\ \quad +8\cdot\sum_{i=1}^{m-6}\sigma(i)+ & \text{for } t = 5 \\ \quad +8\sum_{i=1}^{m-8}\sum_{j=1}^{i}\sigma(j) & \\ \quad \vdots & \vdots \\ 2^m\cdot t-c_m & \text{for } t \geqq \lfloor (m-1)/2 \rfloor \end{cases}$$

The contents c_m depend on the value of m only, for example $c_2=-1$, $c_3=1$, $c_4=7$, $c_5=25$, $c_6=71$, $c_7=185$, $c_8=455$, $c_9=1081$, and $c_{10}=2503$. Because the LR2I^m is an infinite network $\Gamma_{\text{OFF—LR2I}^m}(n, t)=T_{\text{OFF—LR2I}^m}(n, t)=n\cdot\Lambda_{\text{OFF—LR2I}^m}(t)$, for $n, t\geqq 0$.

LRUD2I^m: Of course, we have

$\Lambda_{\text{OFF—LRUD2I}^m}(t)\geqq 2\cdot\Lambda_{\text{OFF—LR2I}^m}(t)-1$, for $t\geqq 0$, and, because LRUD2I^m is an infinite network we have $\Gamma_{\text{OFF—LRUD2I}^m}(n, t)=T_{\text{OFF—LRUD2I}^m}(n, t)=n\cdot\Lambda_{\text{OFF—LRUD2I}^m}(t)$, for $n, t\geqq 0$.

Theorem 2. For standard on-line network systems and $2\leqq p<\infty$, $1\leqq q\leqq p-1$,

$$\Lambda_{\text{ON—NET}_{p,q}}(t)=\begin{cases}0 & \text{for } t=0,\\ 2t-1 & \text{for } t\geqq 1 \text{ and } q=1,\\ (q^t-1)/(q-1) & \text{for } t\geqq 1 \text{ and } q\geqq 2,\end{cases}$$

and $\Gamma_{\text{ON—NET}_{p,q}}(n, t)=T_{\text{ON—NET}_{p,q}}(n, t)=n\cdot\Lambda_{\text{ON—NET}_{p,q}}(t)$, for $n, t\geqq 0$.

Proof. Consider the local data transfer situation first. At $t=1$ assume that a sufficiently large set of input registers obtain input data in parallel by a READ instruction. Then $(q-1)/(q-1)=2t-1=1$ for $q\geqq 2$, or $t=1$. For $q=1$, the maximal local transfer situation, i.e., the maximal transfer of data units to a given register, is possible by indirect addressing. Thus, $\Lambda_{\text{ON—NET}p,1}(t)=2t-1$ for $t\geqq 1$. For $q\geqq 2$, according to (ON.3) it follows that

$$\Lambda_{\text{ON—NET}_{p,q}}(t)=\sum_{i=0}^{t-1}q^i=(q^t-1)/(q-1),$$

where these maximal cardinalities of receptive fields may be obtained in certain PE accumulators. For given n, $t\geqq 0$, by choosing a sufficiently large field of PEs obtaining input data in their accumulators at the first instruction ($i=1$), n receptive fields of maximal cardinality $\Lambda_{\text{ON—NET}p,q}(t)$ may be pairwise disjoint. □

Example 7. By (4.4) we know that $\Lambda_{\text{ON—RAM}}(t)=\Gamma_{\text{ON—RAM}}(n, t)=t$, for $t\geqq 0$ and $n\geqq 1$, and thus $\Lambda_{\text{ON—RAM}}(t)<\Lambda_{\text{ON—NET}_{p,1}}(t)$ as well as $\Gamma_{\text{ON—RAM}}(n, t)<\Gamma_{\text{ON—NET}_{p,1}}(n, t)$ for $t\geqq 2$ and $n\geqq 1$. Furthermore, $T_{\text{ON—RAM}}(n, t)=n\left(t-\frac{n}{2}+\frac{1}{2}\right)$, for $t\geqq n\geqq 1$, and thus $T_{\text{ON—RAM}}(n, t)<T_{\text{ON—NET}_{p,1}}(n, t)$ for $t\geqq n\geqq 2$.

In table 5 for classes of on-line systems mentioned in Section 1 some results on the analysis of general local data transfer functions are collected. For these classes the functions given in Theorem 2 act as upper bounds where the proper values of p and q have to be correlated. By ON-IN$_{\{i_1, i_2, \ldots, i_q\}}$ we denote a special ON-IN system with fixed set $\{i_1, i_2, \ldots, i_q\}$ according to (ON.2). The classes ON-LINEAR$_{\{0\}}$, ON-BINTREE$_{\{1,2\}}$, and ON-QUADTREE$_{\{1,2,3,4\}}$ represent examples for maximal transfer situations as characterized by Theorem 2.

Some remarks about Table 5 and about the other networks which were defined in Table 1:

1. For all examples of CLASS in Table 5 we have $\Gamma_{\text{ON—CLASS}}(n, t)=T_{\text{ON—CLASS}}(n, t)=n\cdot\Lambda_{\text{ON—CLASS}}(t)$, for $n, t\geqq 0$.

Table 5. General local data transfer functions for on-line systems

CLASS	p	$\{i_1, i_2, \ldots, i_q\}$	$\Lambda_{\text{ON-CLASS}}(t)$	$t=4$	$t=8$
LINEAR	2	{0}	$2t-1$	7	15
HEXAGONAL	3	{0, 1}	$t(t+1)/2$	10	36
		{0}	$2t-1$	7	15
SQUARE or ILLIAC	4	{0, 1, 2}	t^2	16	64
		{0, 2}	$t(t+1)/2$	10	36
		{0, 1}, {0}	$2t-1$	7	15
TRIAGONAL	6	{0, 1, 2, 3, 4}	$\frac{5}{2}t^2-\frac{5}{2}t+1$	31	121
		{0, 2, 3, 4}	$\frac{3}{2}t^2-\frac{1}{2}t$	22	92
		{0, 2, 4}	t^2	16	64
DIAGONAL	8	{0, 1, 2, 3, 4, 6, 7}	$\frac{7}{2}t^2-\frac{7}{2}t+1$	43	197
BINTREE	3	{1, 2}	2^t-1	15	255
		{0, 1}	$t(t+1)/2$	10	36
TRIANGLE	5	{1, 2, 3, 4}	2^t-1	15	255
QUADTREE	5	{1, 2, 3, 4}	$(4^t-1)/3$	85	21,845
PS	3	{0, 1}	$([(1+\sqrt{5})^{t+3}-(1-\sqrt{5})^{t+3}]/\sqrt{5}\cdot 2^{t+3})-2$	11	87

2. The class ON-$PS_{\{0,1\}}$ denotes special SIMD systems using the PS network in its original [10] meaning. Let $f_0=1, f_1=1, f_2=2, \ldots, f_{n+2}=f_n+f_{n+1}, \ldots$, where

$$f_n=[(1+\sqrt{5})^{n+1}-(1-\sqrt{5})^{n+1}/\sqrt{5}\cdot 2^{n+1}$$

denotes the nth Fibonacci number, $n\geqq 0$. We have $\Lambda_{\text{ON-PS}_{\{0,1\}}}(t)=\sum_{n=1}^{t} f_n=f_{n+2}-2$, for $t\geqq 0$; cp. [3] for a similar result.

3. For the bintree, triangle, and quadtree network note that the maximal receptive fields may be obtained for the top node accumulator, for $\{i_1, i_2, \ldots, i_q\}$ equal to {1, 2}, {1, 2, 3, 4}, {1, 2, 3, 4}, respectively.

4. The analysis of the general data transfer functions for classes ON-$CUBE^m$, ON-$PM2I^m$, ON-$WPM2I^m$, ON-$LR2I^m$, and ON-$LRUD2I^m$ will not be considered in the present paper.

3. Local, global, and total data dependence measures

For parallel processing systems, the optimal time for the solution of a computational problem depends upon the data transfer abilities of the given system as well as on the principal possibilities of parallelization of a solution process for a given problem. The first may be characterized by the data transfer functions Λ_{SYS}, Γ_{SYS}, T_{SYS} by a general system analysis as considered in Section 2. The second property, however, requires individual consideration of the given computational problem.

For example, consider the multiplication of two $N \times N$ real matrices $A \cdot B = C$. For a given system SYS assume that all N^2 elements of matrix C have to be computed in N^2 different output registers represented by the set R_{OUT}. Let $r \in R_{OUT}$, $R_0 \subseteq R_{OUT}$, and R_1 be the set of N distinctive registers for outputing the N diagonal elements of C. Then it follows that $\lambda_{SYS}(r, t^*) \geqq 2N$, $\gamma_{SYS}(R_1, t^*) \geqq 2N^2$ and $\tau_{SYS}(R_0, t^*) \geqq 2N \cdot \text{card}(R_0)$ if the product $A \cdot B$ is to be computed on SYS within time t^*. Thus, if the functions Λ_{SYS}, Γ_{SYS} or T_{SYS} are known, lower time bounds are derivable from these inequalities for the solution time t^* immediately, where the maximal lower time bound from the three possible values is taken as the result. For example, according to our considerations in Section 2 for the system EXAMP1 we have $t^* \geqq \sqrt{N} - 1$ under the assumption that $M = 2N$. But note that a better lower time bound for this system and the matrix multiplication problem may be obtained by more specialized considerations as demonstrated by Gentleman [3, Theorem 1]. Because each data unit transfer from a certain register r_1 to a certain register r_2 of the system EXAMP1 may be performed in the reverse direction, from r_2 to r_1, in the same time, the proof of Theorem 1 in [3] matches the situation given by the system EXAMP1, i.e., for $r \in R_{OUT}$ we have $\lambda_{EXAMP1}(r, 2t^*) \geqq N^2$, and thus $t^* \geqq \frac{1}{4}(2N^2 - 1)^{1/2} - \frac{1}{4}$.

For a general approach to the derivation of lower time bounds for parallel processing systems we shall use the quantitative description of data dependencies of the desired output data in relation to the input data specification, for computational problems which may be identified with special functions as described later on.

Definition 7. Let $n, m \geqq 1$. Let f be an n-ary function defined on a certain set *domain* (f) of n-tuples of real numbers, and into the set of m-tuples of real numbers. For an n-tuple $(x_1, x_2, \ldots, x_n) \in$ *domain* (f), define

$$\text{sub}_i(x_1, x_2, \ldots, x_n) = \{j : 1 \leqq j \leqq n \,\&\, (\vee x' \neq x_j)(x_1, x_2, \ldots, x_{j-1}, x', x_{j+1}, \ldots, x_n) \in$$

$$\in \textit{domain}(f) \,\&\, \text{proj}_i(f(x_1, x_2, \ldots, x_n)) \neq \text{proj}_i(f(x_1, x_2, \ldots, x_{j-1}, x', x_{j+1}, \ldots, x_n))\}$$

to be the set of all positions j such that changes in the jth component of $(x_1, x_2, \ldots, x_n)$ have an effect on the projection $\text{proj}_i f$, for $1 \leqq i \leqq m$. Then, define

$$\lambda_f = \max_{(x_1, x_2, \ldots, x_n)} \max_{1 \leqq i \leqq m} \text{card}(\text{sub}_i(x_1, x_2, \ldots, x_n)),$$

$$\gamma_f = \max_{(x_1, x_2, \ldots, x_n)} \text{card}\left(\bigcup_{i=1}^{m} \text{sub}_i(x_1, x_2, \ldots, x_n)\right),$$

and

$$\tau_f = \max_{(x_1, x_2, \ldots, x_n)} \sum_{i=1}^{m} \text{card}\left(\text{sub}_i(x_1, x_2, \ldots, x_n)\right).$$

The function f is called *locally d-dependent* iff $d \leqq \lambda_f$, *globally d-dependent* iff $d \leqq \gamma_f$, and *totally d-dependent* iff $d \leqq \tau_f$, for an integer $d \geqq 0$.

By this definition, for arbitrary functions f defined on n-tuples of real numbers and into the set of m-tuples of real numbers, it follows immediately that $\lambda_f = \gamma_f = \tau_f$ if $m=1$, and for $m \geqq 1$

$$\lambda_f \leqq \gamma_f \leqq \tau_f, \tag{7.1}$$

$$\gamma_f \leqq n, \tag{7.2}$$

and

$$\tau_f \leqq m \cdot \lambda_f. \tag{7.3}$$

For example, in the case of the following function f,

$$f(x_1, x_2, x_3, x_4, x_5) = \begin{cases} x_1 + x_2 & \text{if } x_5 = 0 \\ x_3 + x_4 & \text{if } x_5 \neq 0, \end{cases}$$

we have $\text{sub}_1(x_1, x_2, x_3, x_4, 0) = \{1, 2, 5\}$ if $x_1 + x_2 \neq x_3 + x_4$, and $\text{sub}_1(x_1, x_2, x_3, x_4, 0) = \{1, 2\}$ if $x_1 + x_2 = x_3 + x_4$. Because of $\lambda_f = \gamma_f = \tau_f = 3$, this function is local, global, or total 1-, 2-, and 3-dependent, but not 4- or 5-dependent.

Now, in a sequence of examples, the data dependence measures as given by Definition 7 will be analyzed for certain computational problems. The results are collected in Table 6, i.e., the following examples may be considered as explanatory remarks to this table.

Example 8. The *multiplication of two $N \times N$ real matrices* may be considered as a $2N^2$-ary function into the set of N^2-tuples of real numbers. For this computational problem, it is evident that

$$\lambda_{\text{MATRIX–MULTIPLICATION}} = 2N,$$

$$\gamma_{\text{MATRIX–MULTIPLICATION}} = 2N^2, \quad \text{and} \quad \tau_{\text{MATRIX–MULTIPLICATION}} = 2N^3,$$

where these maximal values of data dependence are true for each input vector of length $2N^2$ containing non-zero values in all positions. By this example it follows that the upper bounds (7.2) and (7.3) cannot be reduced in general. The *inversion of an $N \times N$ real matrix in place* may be considered as an N^2-ary function into the set of N^2-tuples of real numbers. We have

$$\lambda_{\text{MATRIX–INVERSION–IP}} = \gamma_{\text{MATRIX–INVERSION–IP}} = N^2,$$

and

$$\tau_{\text{MATRIX–INVERSION–IP}} = N^4,$$

where this maximal case of data dependence appears for any matrix containing non-zero values in all N^2 positions. These data depence quantities may be considered as a direct consequence of the data dependence quantities for the determinant *of an $N \times N$ real matrix,*

$$\lambda_{\text{DETERMINANT}} = \gamma_{\text{DETERMINANT}} = \tau_{\text{DETERMINANT}} = N^2.$$

The solution of a *system of N linear equations* in N unknowns may be considered as an (N^2+N)-ary function into the set of N-tuples of real numbers. We obtain

$$\lambda_{\text{LINEAR–EQUATIONS}} = \gamma_{\text{LINEAR–EQUATIONS}} = N^2+N,$$

and

$$\tau_{\text{LINEAR–EQUATIONS}} = N^3+N^2.$$

Transposing an $N\times N$ real matrix in place may be considered as an N^2-ary function into the set of N^2-tuples of real numbers,

$$\lambda_{\text{TRANSPOSITION–IP}} = 1, \quad \text{and} \quad \gamma_{\text{TRANSPOSITION–IP}} = \tau_{\text{TRANSPOSITION–IP}} = N^2,$$

but for *binary operations on permutated $N\times N$ real matrices in place,*

$$(a_{ij})_{i,j=0,1,\ldots,N-1} \Rightarrow \left(\text{op}_2(a_{ij}, a_{\pi(i,j)})\right)_{i,j=0,1,\ldots,N-1},$$

considered as N^2-ary functions into the set of N^2-tuples of real numbers,

$$\lambda_{\text{MATRIX–}\pi\text{–IP}} = 2 \quad \text{for} \quad \pi \neq id,$$

$$\gamma_{\text{MATRIX–}\pi\text{–IP}} = N^2,$$

and

$$\tau_{\text{MATRIX–}\pi\text{–IP}} = 2N^2 - \text{card}\,\{(i,j): 0 \leqq i,j \leqq N-1 \;\&\; \pi(i,j) = (i,j)\},$$

the transposition may be considered as a special permutation π^*, $\tau_{\text{MATRIX–}\pi^*\text{–IP}} = 2N^2-N$, and op_2 as the exchange operation in this case, $\text{op}_2(a_{ij}, a_{\pi^*(i,j)}) = (a_{\pi^*(i,j)}, a_{ij})$, where the second component of these resulting tuples will be considered as a dummy result.

Example 9. In this example, three two-dimensional transforms of $N\times N$ pictures will be dealt with. First, the *Fourier transform of an $N\times N$ complex matrix* (2D-DFT, two-dimensional discrete Fourier transform, cp. [9]) may be considered as a $2N^2$-ary function into the set of $2N^2$-tuples of real numbers. In this case, we have

$$2N^2-4 \leqq \lambda_{\text{2D–DFT}} \leqq 2N^2-1,$$

$$\gamma_{\text{2D–DFT}} = 2N^2, \quad \text{and} \quad 2N^4 \leqq \tau_{\text{2D–DFT}} \leqq 4N^4-2N^2,$$

where these maximal values of data dependence are true for each input vector of length $2N^2$ containing non-zero values in all positions. For the exact determination of $\lambda_{\text{2D–DFT}}$ and $\tau_{\text{2D–DFT}}$, the influence of different values of N has to be studied. The *Walsh transform of an $N\times N$ real matrix* (2D-WT, two dimensional Walsh transform, cp. [9]) may be considered as an N^2-ary function into the set of N^2-tuples of real numbers,

$$\lambda_{\text{2D–WT}} = \gamma_{\text{2D–WT}} = N^2, \quad \text{and} \quad \tau_{\text{2D–WT}} = N^4,$$

where these maximal values of data dependence are true for any input vector of length N^2. The computation of the *parallel Roberts gradient* (see Example 1) on images of size $M\times N$ may be considered as an MN-ary function into the set of MN-tuples of real numbers. For this function,

$$\lambda_{\text{ROBERTS–GRADIENT}} = 4,$$

$$\gamma_{\text{ROBERTS–GRADIENT}} = MN, \quad \text{and} \quad \tau_{\text{ROBERTS–GRADIENT}} = 4MN-2M-2N-2,$$

by considering the case of non-zero values in all MN positions, and by paying attention to border effects.

Example 10. The computation of the *convex hull of a simple polygon,* cp. [5]' where the N extreme points of the polygon are given by coordinate tuples of real numbers starting with the uppermost-leftmost point, may be considered as a $2N$-ary function into the set of $2N$-tuples of real numbers. In the resulting vector of length $2N$, there appear all coordinate tuples of the extreme points of the convex hull of the given polygon in order, starting with the uppermost-leftmost point, and with the same run orientation as the given polygon. Positions actually not needed in this resulting $2N$-tuple contain value zero by assumption. In this case, it follows that

$$\lambda_{\text{CH—SIPOL}} = \gamma_{\text{CH—SIPOL}} = 2N, \quad \text{and} \quad 2N^2-8N+12 \leqq \tau_{\text{CH—SIPOL}} \leqq 4N^2$$

by analyzing the input situation of special convex polygons with N extreme points as illustrated in Fig. 2, for $N \geqq 4$. The computation of the *convex hull of N planar*

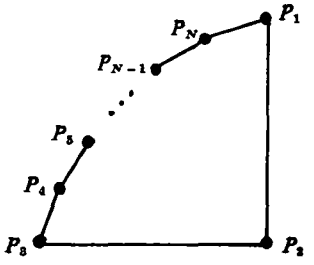


Figure 2.
Convex polygon for analyzing the maximal possible data dependence situation, for $N \geqq 4$

points, cp. [5], given by coordinate tuples of real numbers, may be considered as a $2N$-ary function into the set of $2N$-tuples of real numbers as described above, analogously to the simple polygon situation. For this problem,

$$\lambda_{\text{CH—POINT}} = \gamma_{\text{CH—POINT}} = 2N, \quad \text{and} \quad \tau_{\text{CH—POINT}} = 4N^2,$$

where these maximal values are true for any input situation. The computation of the *Voronoi diagram of N planar points,* cp. [5], given by coordinate tuples of real numbers, may be considered as a $2N$-ary function into the set of $(18N-33)$-tuples of real numbers in the following sense. The Voronoi diagram may have $2N-5$ vertices at most, and, as a special planar graph, $3N-6$ edges at most, for $N \geqq 3$. See Fig. 3 for an illustration of the construction of such a "maximal Voronoi diagram", where the number $v(N)$ of vertices, and the number $e(N)$ of edges satisfy the recursive equations

$$v(3) = 1, \quad e(3) = 3,$$

$$v(N+1) = v(N)+2, \quad \text{and} \quad e(N+1) = e(N)+3$$

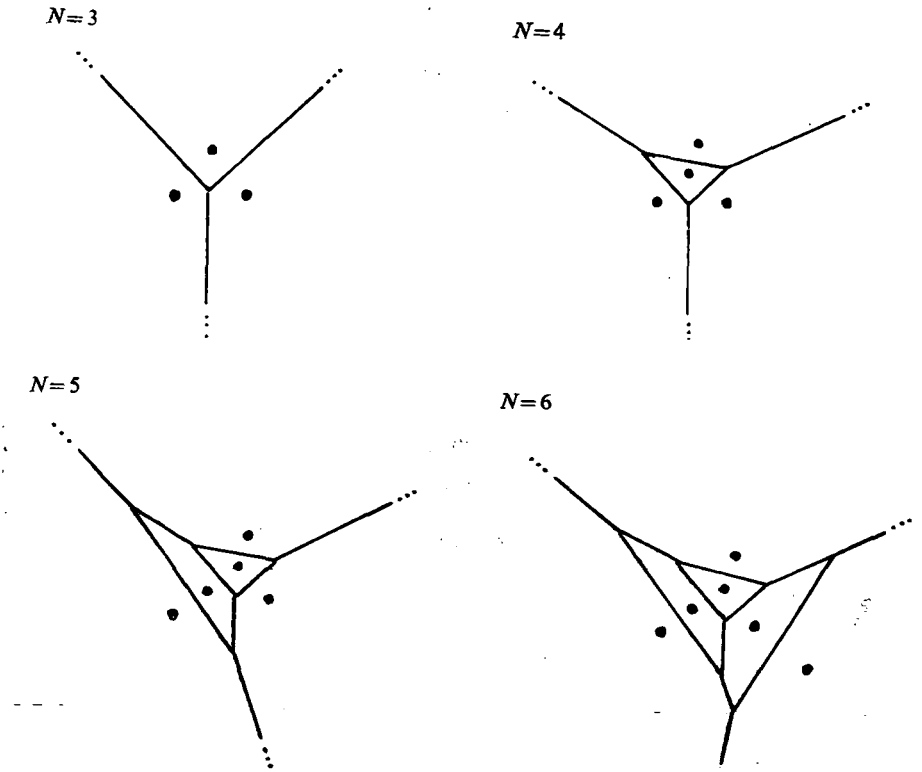


Figure 3.
Voronoi diagrams for $N=3, 4, 5, 6$ with $2N-5=1, 3, 5, 7$ vertices and $3N-6=3, 6, 9, 12$ edges, respectively

for $N \geqq 3$. The $18N-33=3(2N-5)+4(3N-6)$ positions of the resulting vector of a Voronoi diagram computation we consider as a unique characterization of a Voronoi diagram by linearization of adjacency lists for this special graph structure with the positions for each vertex where two are reserved for the coordinate values and one for a common pointer, and two times two positions for each edge — for the index of the vertex at the other end of the edge, of for the slope of the edge, and for a common pointer. For concrete inputs of N points, positions actually not needed in the resulting $(18N-33)$-tuple contain value zero by assumption. Then, we have

$$\lambda_{\text{VORONOI-DIAGRAM}} = \gamma_{\text{VORONOI-DIAGRAM}} = 2N,$$

and

$$12N-3 \leqq \tau_{\text{VORONOI-DIAGRAM}} \leqq 2N(18N-33),$$

for $N \geqq 3$, where the local and global case may be analyzed by using a regular N-gon, and for the total case a Voronoi diagram in the sense of Fig. 3, with $2N-5$ points, was used where each point of the diagram essentially depends on there input points, i.e., on six coordinate values.

Example 11. *Matching of a pattern* of length M against a string of length N ($M \leqq N$ and the elements of pattern and string are assumed to be reals) may be considered as a $(N+M)$-ary function into the set of $(N-M+1)$-tuples on $\{0, 1\}$ where, for

$$f_{\text{PATTERN—MATCHING}}(p_1, p_2, \ldots, p_m;\ s_1, s_2, \ldots, s_m)=(e_1, e_2, \ldots, e_{N-M+1})$$

we have $e_i=1$ iff $s_{i+j}=p_{j+1}$, for all $j=0, 1, \ldots, M-1$, and $e_i=0$ otherwise, for $i=1, 2, \ldots, N-M+1$. We have

$$\lambda_{\text{PATTERN—MATCHING}} = 2M,$$

$$\gamma_{\text{PATTERN—MATCHING}} = M+N, \quad \text{and} \quad \tau_{\text{PATTERN—MATCHING}} = 2M(N-M+1).$$

In all three cases, the maximal dependence may be analyzed for the trivial input situation $p_i=s_j=\text{const}$, for $i=1, 2, \ldots, M$ and $j=1, 2, \ldots, N$. *Detection of a pattern* of length M within a string of length N, $M \leqq N$, may be considered as an $(N+M)$-ary function into the set $\{0, 1\}$ where the output is equal to $\max\{e_i: i=1, 2, \ldots, N-M+1 \ \&\ f_{\text{PATTERN—MATCHING}}(p_1, p_2, \ldots, p_M; s_1, s_2, \ldots, s_N)= =(e_1, e_2, \ldots, e_{N-M+1})\}$ for input $(p_1, p_2, \ldots, p_M; s_1, s_2, \ldots, s_N)$. Then,

$$\max\{2M, M+\lfloor N/M \rfloor\} \leqq \lambda_{\text{PATTERN—SIGNALIZATION}} \leqq M+N.$$

Note that this represents the first example of a computational problem where the equality $\gamma_f=n$ remains an open problem, for an n-ary function f with $n=N+M$ in the case of pattern detection. As a last example, *sorting of N real numbers* may be considered as an N-ary function into the set of N-tuples of real numbers. For this very important problem, we have

$$\lambda_{\text{SORTING}} = \gamma_{\text{SORTING}} = N, \quad \text{and} \quad \tau_{\text{SORTING}} = N^2,$$

where these maximal values are true for N pairwise different input values.

4. Data transfer lemma and applications

Between the quantitative descriptions of data transfer for SIMD systems (Section 2) and of data dependence for computational problems (Section 3), the following direct relation holds.

Lemma 1. (Data Transfer Lemma). Let SYS$\in$SIMD, and let π be an arbitrary program for SYS for the computation of a function f which is n-ary and has m-tuple values. Let R denote the set of output registers of SYS where the m-tuples appear at the end of the computation ($\text{card}(R)=m$, off-line mode), or those output registers of SYS via which the computed values of the m-tuples leave SYS in certain waves of information ($\text{card}(R) \leqq m$, on-line mode). Then, the computation of $f(x_1, x_2, \ldots, x_n)$ on SYS by π requires at least t_0 steps of comdutation for a given input $(x_1, x_2, \ldots, x_0) \in \mathit{domain}(f)$, where $\Lambda_{\text{SYS}}(t_0) \geqq \lambda_f$, $\Gamma_{\text{SYS}}(\text{card}(R), t_0) \geqq \gamma_f$, and $T_{\text{SYS}}(\text{card}(R), t_0) \geqq \tau_f$.

Proof. Let us consider the local off-line or on-line situation. Assume that $\lambda_f=\text{card}(\text{sub}_i(x_1, x_2, \ldots, x_n))$, for a given input vector $(x_1, x_2, \ldots, x_n)$, and for

a given position i, $1 \leqq i \leqq m$. Let $\mathrm{sub}_i(x_1, x_2, \ldots, x_n) = \{j_1, j_2, \ldots, j_{\lambda_f}\}$. For any position i_k, $k = 1, 2, \ldots, \lambda_f$, either the name of an input register receiving value x_{j_k} at a given moment will be transfered to the receptive field $\mathrm{rec}_\pi^{(x_1, x_2, \ldots x_n)}(r^{(i)}, t^*)$ by some operational instructions only, if value $\mathrm{proj}_i(f(x_1, x_2, \ldots, x_n))$ appears in register $r^{(i)} \in R$ at time $t^* \leqq t_0$ of computation, or during the t^* steps of computation of $\mathrm{proj}_i(f(x_1, x_2, \ldots, x_n))$ at least one test instruction JGTZ, JZERO, or JLTZ must be performed where the contents of the CPU accumulator depends on the input value x_{j_k} at the moment of testing. In the second case, if the test instruction is followed by certain operational instructions directed to register $r^{(i)}$ the name of the input register receiving value x_{j_k} at a given moment will be transferred to the receptive field $\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)$, too; cp. (iv) in Definition 4. Without loss of generality, assume that $j_1, j_2, \ldots, j_v$, $v \leqq \lambda_f$, denote all the positions which have produced register names in the receptive field $\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)$. If $v = \pi_f$, then $\pi_f \leqq \mathrm{card}\,(\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)) \leqq \lambda_{\mathrm{SYS}}(t_0)$ follows immediately. For $v < \lambda_f$, let $t_1, t_2, \ldots, t_w$ be all the moments where test instructions have to be performed according to π and input $(x_1, x_2, \ldots, x_n)$ such that the contents of the CPU accumulator depend on one of the input values $x_{j_{v+1}}, \ldots, x_{j_{\lambda_f}}$ at least, at the moments of testing. Consider the following program π' computing something unspecified, produced by π and $(x_1, x_2, \ldots, x_n)$ in the following way:

— all test instructions at moments $t_1, t_2, \ldots, t_w$ will be deleted in π, and
— all other instructions of π will be performed according to π and input $(x_1, x_2, \ldots, x_n)$, in the same order, where all instructions LOAD α or $\mathrm{OP}_1\,\alpha$, for α equal to $=x, m, {}^*m$, or (j), will be replaced by $\mathrm{OP}_2\,\alpha$, for the same value of α, if such instructions appear in π.

Thus, the receptive field of register 0, i.e., the CPU accumulator, will increase monotonically according to π' and $(x_1, x_2, \ldots, x_n)$. After $t^* - w$ operations according to π', $\mathrm{rec}\,(0, t^* - w)$ contains all input register names for the input data $x_{j_{v+1}}, \ldots, x_{j_{\lambda_f}}$. This receptive field will be combined with $\mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^* - w) \geqq \mathrm{rec}_\pi^{(x_1, x_2, \ldots, x_n)}(r^{(i)}, t^*)$ at moment $t^* - w + 1 \leqq t^*$ by adding an instruction $\mathrm{OP}_2\,\alpha$ (see conditions (OFF.2) and (ON.6)) or $\mathrm{OP}_2(j)$ (see conditions (OFF.4) and (ON.7)) to π'. Thus, $\lambda_f \leqq \mathrm{card}\,(\mathrm{rec}_{\pi'}^{(x_1, x_2, \ldots, x_n)}(0, t^* - w + 1)) \leqq \Lambda_{\mathrm{SYS}}(t^* - w + 1) \leqq \Lambda_{\mathrm{SYS}}(t_0)$. Note that the off-line or on-line I/O convention is necessary to ensure that a non-accumulator PE register $r^{(i)}$ may be replaced by the accumulator of the same PE which is an output register, too. For this replacement, parallel STORE instructions may be replaced by parallel OP_1 instructions using the same masks for PE addresses.

What we have explained is one of the possible ways to ensure the necessary data transfer within time limit t_0, for the local off-line or on-line situation. The essential point in the program transformation from π to π' may be characterized by the word "linearization", because all test instructions could be deleted, in fact. This linearization approach may be used for the local, global and total situation in the following way.

For the given program π and an input situation I, all the performed instructions will be written as a linear sequence S_0. We obtain sequence S_1 by deletion of all instructions JLTZ, JZERO, JGTZ, JUMP, WRITE, and HALT in sequence S_0. Now, for the special case of an on-line program, if in sequence S_0 there were some STORE instructions in front of a WRITE instruction directed to certain output

registers $r \in R$, then these STORE instructions will be shifted to the end of sequence S_1. In the resulting sequence S_2, all serial or parallel $OP_1\,\alpha$ or LOAD α instructions will be replaced by an $OP_2\,\alpha$ instruction formally, in the same position for the same value of α. For the resulting sequence S_3 we have monotonically increasing receptive fields for all accumulators, for the CPU and PEs. Also, by the described step from S_1 to S_2, for sequence S_3 the receptive fields of output registers will be monotonically increasing for consecutive output waves of information. Now, if in the original sequence S_0 there was no test instruction, our program linearization is finished. In the other case, in S_3 we shall place an instruction JZERO, e.g., in that position where the last test instruction was located in sequence S_0. Now consider an arbitrary output register $r \in R$. If there is an operational instruction behind the JZERO instruction directed to r then register r will obtain the receptive field of the CPU accumulator containing all the register names corresponding to tested input values, cp. (iv) in Definition 4. If there is no operational instruction behind the JZERO instruction directed to r then we shift the last instruction directed to r in front of the JZERO instruction to a position behind this instruction. By consideration of all registers $r \in R$, our program linearization is finished. Note that the length of the resulting linear instruction sequence is restricted by the length of the original sequence S_0.

Now assume that $\lambda_f = \mathrm{card}\,(\mathrm{sub}_i(x_1, x_2, \ldots, x_n))$ for a certain i, $1 \leqq i \leqq n$, $\gamma_f = \mathrm{card}\,(\bigcup_{i=1}^{m} \mathrm{sub}_i(y_1, y_2, \ldots, y_n))$ and $\tau_f = \sum_{i=1}^{m} \mathrm{card}\,(\mathrm{sub}_i(z_1, z_2, \ldots, z_n)$, for certain input vectors $(x_1, x_2, \ldots, x_n)$, $(y_1, y_2, \ldots, y_n)$, $(z_1, z_2, \ldots, z_n)$. These input vectors characterize input situations I_x, I_y, I_z for SYS. By linearization of π according to these input situations we obtain linear programs π_x, π_y, π_z, respectively, all of length $\leqq t_0$. Thus, we have

$$\lambda_{\pi_x}^{(x_1, x_2, \ldots, x_n^T)}{}_{(R, t_0)} \geqq \lambda_f,$$

$$\gamma_{\pi_y}^{(y_1, y_2, \ldots, y_n)}{}_{(R, t_0)} \geqq \gamma_f,$$

$$\tau_{\pi_z}^{(z_1, z_2, \ldots, z_n)}{}_{(R, t_0)} \geqq \tau_f,$$

which proves our statements. □

Corollary 1. Let CLASS $\subseteq$ SIMD. For any system SYS $\in$ CLASS, the computation of a function f which is into the set of m-tuples of real numbers requires at least t_0 steps of computation in the worst case, where $\Lambda_{\mathrm{CLASS}}(t_0) \geqq \lambda_f$, $\Gamma_{\mathrm{CLASS}}(m, t_0) \geqq \gamma_f$, and $T_{\mathrm{CLASS}}(m, t_0) \geqq \tau_f$.

Proof. Immediately by Lemma 1 where the generalization about all programs computing the function f is used as well as about all systems of CLASS. For the on-line case note that there may already be a certain $m_0 \leqq m$ such that $\Gamma_{\mathrm{CLASS}}(m_0, t_0) \geqq \gamma_f$, and $T_{\mathrm{CLASS}}(m_0, t_0) \geqq \tau_f$. □

Example 12. Let CLASS = {EXAMP1} and consider the computation of the parallel Roberts gradient as described in Example 1. In this case we get the trivial lower time bound 1 only; an upper bound was 29. Now, let CLASS = {EXAMP3} and consider the computation of the arithmetical averages of M consecutive waves of information of length $N = 2^{n-1}$ as described in Example 3. Here, by Corollary 1

we obtain the lower time bound $n+2M-2=\max\{n-1,\ n+2M-2,\ n+M-1\}$, cp. equation (6.1), (6.2), (6.3), for values $\lambda_f=N, \gamma_f=N\cdot M$ and $\tau_f=N\cdot M$. An upper bound was $6M+n$.

Using common asymptotic notations, for both examples the optimal times $\theta(1)$ and $\theta(M+n)$ are known as a result.

Theorem 3. For any system SYS$\in$OFF-NET$_p$, $p\geqq 2$, the computation of a function f which is into the set of m-tuples of real numbers requires at least t_0 steps of computation in the worst case, where

$$t_0 \geqq \max\{(d_1-1)/2,\ (d_2-m)/2m,\ (d_3-m)/2m\}$$

for $p=2$, and for $p\geqq 3$

$$\begin{aligned} t_0 \geqq \max\{&\log_{p-1}(d_1(p-2)+2)-1.586,\\ &\log_{p-1}(d_2(p-2)+2)-\log_{p-1}m-1.586,\\ &\log_{p-1}(d_3(p-2)+2)-\log_{p-1}m-1.586\}, \end{aligned}$$

if f is locally d_1-dependent, globally d_2-dependent, and totally d_3-dependent.

Proof. Immediately by Theorem 1, Definition 7 and Corollary 1 where the relation $\log_{p-1}p>1.586$, $p\geqq 3$, was used. □

In Table 7 are collected, for the classes of off-line systems defined in Section 1, the lower time bounds that may be obtained by using Corollary 1. Because the classes OFF-LINEAR, OFF-PS, OFF-BINTREE and OFF-QUADTREE represent examples for the maximal transfer situation as characterized by Theorem 1, for these classes the lower time bounds are as given by Theorem 3. If a function f into the set of m-tuples is globally or totally d'-dependent, then the value d has to be replaced by d'/m in the lower time bounds given in Table 7, to obtain the corresponding values for the global or total situation.

Theorem 4. For any system SYS$\in$ON-NET$_{p,q}$, $2\leqq p<\infty$, $1\leqq q<p$, the computation of a function f which is into the set of m-tuples of real numbers requires at least t_0 steps of computation in the worst case, where

$$t_0 \geqq \max\{(d_1+1)/2,\ (d_2+m)/2m,\ (d_3+m)/2m\}$$

for $q=1$, and for $q\geqq 2$

$$\begin{aligned} t_0 \geqq \max\{&\log_q(d_1(q-1)+1),\quad \log_q(d_2(q-1)/m+1),\\ &\log_q(d_3(q-1)/m+1^T\}, \end{aligned}$$

if f is locally d_1-dependent, globally d_2-dependent, and totally d_3-dependent.

Proof. Immediately by Theorem 2, Definition 7 and Corollary 1. □

In Table 8 are collected, for the classes of on-line systems defined in Section 1, the lower time bounds that may be obtained by using Corollary 1. Because the classes ON-LINEAR$_{\{0\}}$, ON-BINTREE$_{\{1,2\}}$, and ON-QUADTREE$_{\{1,2,3,4\}}$ represent examples for maximal transfer situations as characterized by Theorem 2,

for these classes the lower time bounds are as stated by Theorem 4. As in the case of Table 7, if a function f into the set of m-tuples is globally or totally d'-dependent, then the value d has to be replaced by d'/m in the lower time bounds given in Table 8, for obtaining the corresponding values for the global or total situation. Note that value m may be replaced by a value $m_0 \leqq m$ for special ON-NET systems.

5. Conclusions

In this paper we have given a general framework for the description of parallel processing systems, and explained how data flow may be used for analyzing lower time bounds in general. Note that this approach may be applied to supercomputers as well as to on-chip realizations. Problems connected with the technical features

Table 6. Local, global and total data dependence measures

Computational problem f	n	m	λ_f	$\gamma_{f_\bullet}$	τ_f
MATRIX MULTIPLICATION	$2N^2$	N^2	$2N$	$2N^2$	$2N^3$
MATRIX INVERSION IP	N^2	N^2	N^2	N^2	N^4
DETERMINANT	N^2	1		N^2	
LINEAR EQUATIONS	N^2+N	N	N^2+N	N^2+N	N^3+N^2
TRANSPOSITION IP	N^2	N^2	1	N^2	N^2
MATRIX π IP	N^2	N^2	2 for $\pi \neq id$	N^2	$2N^2 - \#\{(i,j): \pi(i,j)=(i,j)\}$
2D—DFT	$2N^2$	$2N^2$	$\geqq 2N^2-4$ $\leqq 2N^2-1$	$2N^2$	$\geqq 2N^4$ $\leqq 4N^4-2N^2$
2D—WT	N^2	N^2	N^2	N^2	N^4
ROBERTS GRADIENT	MN	NM	4	MN	$4MN-2M-2N-2$
CH SIPOL	$2N$	$2N$	$2N$	$2N$	$\geqq 2N^2-8N+12$ $\leqq 4N^2$
VORONOI DIAGRAM	$2N$	$18N-33$	$2N$	$2N$	$\geqq 12N-30$ $\leqq 36N^2-66N$
PATTERN MATCHING	$N+M$	$N-M+1$	$2N$	$M+N$	$2M(N-M+1)$
PATTERN SIGNALIZATION	$N+M$	1	$\geqq \max\{2M, M+\lfloor N/M \rfloor\}$,		$\leqq M+N$
SORTING	N	N	N	N	N^2

of architecture elements were by passed by the selected level of abstract system description. Thus, in the discussion of parallel algorithms for a given model SYS$\in$ $\in$SIMD we may have in mind quite different technical implementations, but we may discuss parallel algorithms for all of them at once using the abstract model SYS$\in$SIMD. For example, an important problem is given by the necessary decision between different structures of parallel processing systems to ensure efficient algorithmic solutions for classes of computational problems such as mentioned in Example 8 (matrix-type computations), 9 (two-dimensional transforms), 10 (geometric problems), or 11 (combinatorial problems). According to our considerations in [4] the selection of parallel algorithms crucially depends on the given parallel processing system and comparisons between different SIMD systems on the basis of knowledge about optimal algorithms represents quite a hard task. Also, there are nearly as many different models for parallel processing as papers on this topic, making comparative studies of different parallel structures nearly impossible. In the present paper an attempt was made to propose a classification of special parallel processing systems which have been of widespread interest in the past. The proof of the practicability of the proposed exact definition of SIMD systems will be the subject of forthcoming papers; the first programs of the PARSIS project fit well into this framework.

By using Tables 6, 7, and 8 the interested reader may obtain lower time bounds for different combinations of SIMD systems and computational problems, e.g., the lower time bound $\log_2(N^2+1)$ for the two-dimensional Walsh transform on

Table 7. Lower time bounds for off-line systems in OFF-CLASS for computing a local d-dependent function

CLASS	p	lower time bound	$d=128$	$d=128^2$
LINEAR	2	$(d-1)/2$	64	8, 192
HEXAGONAL	3	$\left(\left(\frac{8}{3}d-\frac{5}{3}\right)^{1/2}-1\right)\Big/2$	9	105
SQUARE or ILLIAC	4	$((2d-1)^{1/2}-1)/2$	8	91
TRIAGONAL	6	$\left(\left(\frac{4}{3}d-\frac{1}{3}\right)^{1/2}-1\right)\Big/2$	7	74
DIAGONAL	8	$(d^{1/2}-1)/2$	6	64
PS	3	$\log_2(d+2)-1.586$	6	13
BINTRE	3	$\log_2(d+2)-1.586$	6	13
top node		$\log_2(d+1)-1$	7	14
TRIANGLE	5	$t_0 \geqq \log_2(d-t_0^2+2t_0+5)-2.586$	5	12
top node		$\log_2(d+1)-1$	7	14
QUADTREE	5	$\log_4(3d+2)-1.161$	4	7
top node		$\log_4(3d+1)-1$	5	7

Table 8. Lower time bounds for on-line systems in ON-CLASS for computing a local d-dependent function

CLASS	p	$\{i_1, \ldots, i_q\}$	Lower time bound	$d=128$	$d=128^2$
LINEAR	2	{0}	$(d+1)/2$	65	8,193
HEXAGONAL	3	{0, 1}	$((8d+1)^{1/2}-1)/2$	16	181
SQUARE or ILLIAC	4	{0, 1, 2}	$d^{1/2}$	12	128
TRIAGONAL	6	{0, 1, 2, 3, 4}	$\left(\left(\frac{8}{5}d-\frac{3}{5}\right)^{1/2}-1\right)\Big/2$	7	81
DIAGONAL	8	{0, 1, 2, 3, 4, 6, 7}	$\left(\left(\frac{8}{7}d-\frac{3}{7}\right)^{1/2}-1\right)\Big/2$	6	64
BINTREE	3	{1, 2}	$\log_2(d+1)$	8	15
TRIANGLE	5	{1, 2, 3, 4}	$\log_2(d+1)$	8	15
QUADTREE	5	{1, 2, 3, 4}	$\log_4(3d+1)$	5	8
PS	3	{0, 1}	$f_{t_0+2} \geqq d+2$ for the Fibonacci numbers $f_0, f_1, f_2, \ldots$	11	21

ON-TRIANGLE systems. The characterization of data dependencies for computational problems as given by Definition 7 may be refined, e.g., by consideration of changes of function values not only by changing arguments in one position but in several positions.

Abstract

Starting with an exact definition of classes of SIMD (single instruction, multiple data) systems, a general approach to obtaining lower time bounds by data flow analysis is presented. Several interconnection schemes, such as the square net, the perfect shuffle, the infinite binary tree, etc. are analyzed with respect to their data transfer possibilities. For some types of computational problems the data dependencies are analyzed in a quantitative way. From both types of analysis, lower time bounds result for many combinations of SIMD systems and computational problems, for example, $O(\log N)$ for on-line quadtree-net systems and the computation of Voronoi diagrams for N planar points, $O(N)$ for off-line diagonal-net systems and the two-dimensional discrete Fourier transform, and $O(\sqrt{N})$ for off- or on-line Illiac-net systems and sorting of N items.

CENTER FOR AUTOMATION RESEARCH
UNIVERSITY OF MARYLAND
COLLEGE PARK, MD 20742
U.S.A.

* PERMANENT ADDRESS:
FRIEDRICH SCHILLER UNIVERSITY
DEPARTMENT OF MATHEMATICS,
UNIVERSITY TOWER 17TH FLOOR,
DDR-6900 JENA,
GERMAN DEMOCRATIC REPUBLIC

The support of the U. S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Janet Salzman in preparing this paper. The author thanks the government of the German Democratic Republic for financial support and Azriel Rosenfeld for his efforts in making the author's stay in College Park possible and effective as well.

References

[1] H. ABELSON, Lower bounds on information transfer in distributed computations, J. ACM *27* (1980), 384—392.
[2] A. V. AHO, J. E. HOPCROFT, and J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).
[3] W. M. GENTLEMAN, Some complexity results for matrix computation on parallel processors, J. ACM *25* (1978), 112—115.
[4] R. KLETTE, Zeitkompliziertheit von Berechnungsproblemen der digitalen Bildverarbeitung — Vergleiche zwischen sequentieller und paralleler Datenverarbeitung (in Slovakian, to appear, VEDA Publish. House, Bratislava).
[5] R. KLETTE, Geometrische Probleme der digitalen Bildverarbeitung, BILD UND TON *35* (1982), 101—110.
[6] R. KLETTE and R. LINDNER, Zweidimensionale Vektormaschinen und ihr Leistungsvermögen bei der Lösung von Entscheidungsproblemen der Aussagenlogik, EIK *15* (1979), 37—46.
[7] T. LEGENDI, A cellular processor project, International Workshop on Parallel Processing by Cellular Automata, Berlin, GDR, Sept. 15—16, 1982.
[8] V. R. PRATT and L. J. STOCKMEYER, A characterization of the power of vector machines, J. Computer System Sciences *12* (1976), 118—121.
[9] A. ROSENFELD and A. C. KAK, *Digital Picture Processing* (Second Ed.), Academic Press, New York (1982).
[10] H. J. SIEGEL, A model of SIMD machines and a comparison of various interconnection networks, IEEE Trans. Computers *C-28*, (1979), 907—917.

Received May 13, 1983.

On the number of zero order interpolants

P. ECSEDI-TÓTH* and L. TURI**

1. Introduction

Our motivation for determining the set of all interpolants of arbitrarily given sentences φ and ψ is twofold, both originating in computer science.

Firstly, according to the well-known method of Floyd—Hoare in the theory of program verification, a program (or more precisely, a program schema) must be associated by so called assertions, which are, actually, first order open formulae. This association can be partially mechanized; the difficulty arises in associating assertions to loops. If φ is the assertion immediately before the loop and ψ is the one immediately after it, then the assertion associated to the loop is not so easy to look for. One possible escape is provided by the theory of interpolation: the assertion to be associated to the loop must be an interpolant of φ and ψ. The celebrated model theoretic result of W. Craig states the existence of an interpolant if φ and ψ are first order sentences and φ is a logical consequence of ψ. In the above mentioned problem, however, one needs more than one (possibly, all of the) interpolants to support the choice of the loop-assertion, on the one hand, and then, obviously, he must generalize to open formulae. At the first stage of this process, we aim the investigation of the set of all interpolants of any two first order sentences φ and ψ. Our method is traditional: we reduce φ and ψ into the zero order language, where matters are very much smoother. Thus, algorithmic generation of the set of all zero order interpolants of any two zero order sentences, the topic of the present paper, is a part of our treatment of the first order case.

Our second motivation can be paraphrased as follows: on the zero order level, an interpolant of φ and ψ can be considered as a generalization (or a relativization) of the well-known concept of "implicant". Indeed, taking φ as the false formula, the set of interpolants of φ and ψ concides with the set of implicants of ψ. This observation provides us with the possibility to consider "implicants of ψ relative to φ", which, in turn, may yield to a better understanding of synthesis problems of truth-functions and automata.

These considerations, however, will remain in the background in the present paper and will be published elsewhere. Our purpose here is much simpler: to investigate the case of zero order sentences and to present an algorithm which returns the set of all interpolants of arbitrarily given zero order sentences.

The method employed here is based on the isomorphism between the zero order Lindenbaum—Tarski algebra and the Boolean algebra of truth-functions associated to the equivalence classes of zero order sentences.

By an interpolant of φ and ψ, we mean a zero order sentence χ which is an interpolant in the sense of Craig [1] and χ is equivalent neither to φ nor to ψ; i.e. χ is proper. According to this strengthening, Craig's Theorem on the existence of (proper) interpolants no longer holds without additional assumptions: it may well happen, that for fixed φ and ψ, no proper interpolant exists: i.e. any interpolant (which exists in the sense of Craig) is equivalent to either φ or ψ.

To study the Boolean algebra of truth functions, we shall use trees. To every truth function, we associate a binary tree, the "valuation tree" of the function at hand. The valuation tree associated to a function is a compressed form of the truth-table of that function. Being so, the tree contains every information (up to logical equivalence) about the function [2]; and since interpolants are defined by means of logical consequence, the trees associated to (the arbitrarily given) φ and ψ contain every information about the set of their interpolants. On the other hand, the "geometrical content" of trees gives us the possibility of expressing semantical properties of functions, and in particular, of interpolants in a simple and "visualizable" way. Additionally, an easy method is imposed to calculate the exact number as well as the number and length of maximal chains of equivalence classes of interpolants. The conditions under which proper interpolants exist are formulated in terms of trees; they have, however, a natural and easily comprehensible meaning for sentences, too.

The organization of the paper is as follows. In the next section, we concretize our terminology and notations. In Section 3 we give conditions which are equivalent to the existence of proper interpolants. The method developed there will be applied to obtain our main results in Section 4 on the number of interpolants and chains of interpolants, respectively. We conclude a next to trivial consequence on the algebraic structure of interpolants in Section 5. Finally, we reformulate our results for sentences in terms of model theory, in Section 6.

2. Preliminaries

Throughout the paper we keep fixed a countably infinite set S, which will play the role of sentence symbols when we are dealing with formulae, while in case of truth functions, S will be considered as a set of variables.

2.1. Let F be the set of zero order sentences over S. Let $\equiv$ denote the logical equivalence relation on F. Clearly, $\equiv$ is an equivalence relation indeed; let us denote by $[\varphi]$ the equivalence class containing φ ($\varphi \in F$). It is well-known, that $\mathfrak{F}=\langle F/\equiv, \wedge, \vee, \neg, 0, 1\rangle$ is a Boolean algebra, the so called Lindenbaum—Tarski algebra of F, [1], where 0 denotes the class of unsatisfiable elements of F while 1 stands for the class of valid ones; the operations being defined in the natural way: $\neg[\varphi]=[\neg\varphi]$, $[\varphi]\wedge[\psi]=[\varphi\wedge\psi]$, $[\varphi]\vee[\psi]=[\varphi\vee\psi]$.

2.2. Let $B=\bigcup_{n\in\omega} B_n$, where $B_n=\{f \mid f: 2^n \to 2;\ 2=\{0,1\}\}$, the set of Boolean functions of finite number of variables taken from S. By an assignment we mean an element of the set ${}^{\omega}2=\{\langle \xi_0, \xi_1, \ldots\rangle \mid \xi_i \in \{0,1\} \text{ for } i\in\omega\}$. The value of $f\in B$

under an assignment $\xi\in{}^{\omega}2$ (in notation: $f(\xi)$) is obtained firstly, by substituting for all $i\in\omega$, the i-th component ξ_i of ξ for the i-th variable $s_i(\in S)$ everywhere in f provided s_i occurs in f (otherwise the i-th component of ξ has no effect on the value of f) and secondly, by calculating that value. We say, that f and $g(\in B)$ are equivalent, in notation: $f\sim g$, iff $f(\xi)=g(\xi)$ for all $\xi\in{}^{\omega}2$. It follows, that $\sim$ is an equivalence relation over B; the equivalence classes are denoted as those in F: i.e. for $f\in B$, the equivalence class containing f is denoted by $[f]$. We shall use the symbols 0 and 1 in B, too: $0=\{f\,|\,f(\xi)=0$ for all $\xi\in{}^{\omega}2\}$ and $1=\{f\,|\,f(\xi)=1$ for all $\xi\in{}^{\omega}2\}$. For $g, f\in B$, we can define the operations $+,\cdot,$ and $^{-}$(bar) as follows: for $\xi\in{}^{\omega}2, f(\xi)+g(\xi)=\max\{f(\xi), g(\xi)\}$, $f(\xi)\cdot g(\xi)=$ $=\min\{f(\xi), g(\xi)\}$ and $\bar{f}(\xi)=1-g(\xi)$, respectively. Since $\sim$ is compatible with these operations, we can carry them over classes in $B/\sim$: $\overline{[f]}=[\bar{f}]$, $[f]\cdot[g]=[f\cdot g]$, $[f]+[g]=[f+g]$. What is obtained is the well-known Boolean algebra $\mathscr{B}=$ $=\langle B/\sim, \cdot, +, ^{-}, 0, 1\rangle$. Obviously, $\mathscr{F}$ is isomorphic to $\mathscr{B}$. For the sake of simplicity, from now on, when we speak about functions, we shall tacitly mean the equivalence classes they do represent, and we shall omit brackets in notations, i.e. $f\in\mathscr{B}$ is always to be understood as $[f]\in B/\sim$. Legality of this seemingly abuse of terminology will be justified in Section 5, Theorem 14.

2.3. By a full binary tree of level n ($n\in\omega$) we mean an ordered pair $T=\langle V, E\rangle$ where V, the set of vertices is defined by

$$V=\bigcup_{j=0}^{n}\bigcup_{k=1}^{2^j}\{V_{jk}\}$$

and E, the set of edges is

$$E=\{\langle V_{jk}, V_{tl}\rangle\,|\,t=j+1,\ l=2\cdot k-\beta \text{ where } 0\leqq j\leqq n-1,\ 1\leqq k\leqq 2^j,\ \beta\in\{0,1\}\}.$$

In particular, if $n=0$, then $V=\{V_{01}\}$, $E=\emptyset$, i.e. the full binary tree of level 0 is a point. The indices j, k of a vertex $V_{jk}\in V$ mean that V_{jk} is the k-th point of T on the j-th level. We shall label the edge $\langle V_{jk}, V_{(j+1)(2k-\beta)}\rangle$ by s_{j+1}^{β}. Note, that the label s_{j+1}^{β} does not depend on k.

Let $T=\langle V, E\rangle$ be a full binary tree of level n. By a path p in T we mean a sequence of vertices $V_{0k_0}, V_{1k_1}, \ldots, V_{nk_n}$ such that $k_0=1$ and for all $j(0\leqq j\leqq n-1)$, $\langle V_{jk_j}, V_{(j+1)k_{j+1}}\rangle\in E$. The set of paths in T will be denoted by P_T. Clearly, card $P_T=2^n$. If $P\subseteqq P_T$, then P determines in the natural way a subtree of T. If we write "T_1 is a tree of level n", then we always mean, that T_1 is determined by a subset of paths P of a full binary tree T of level n. Similarly, "T_1 is a subtree of T_2" is to be understood, as both, T_1 and T_2 are determined by subsets P_1 and P_2 of a full binary tree T such that $P_1\subseteqq P_2$ (i.e. T_1, T_2 and T are of the same level). The set of all subtrees of a full binary tree T will be denoted by Sub T, and in each element of Sub T, the vertices will be indexed by the same indices as they were in T. If $T_1\in$ Sub T and $T_1\neq T$, then we write $T_1\subset T$. Similar notation applies to arbitrary binary tree. Obviously, if T is a full binary tree of level n, then card (Sub T)$=2^{2^n}$.

Let $T=\langle V, E\rangle$ be a full binary tree of level n and $\langle V_{0k_0}, \ldots, V_{jk_j}, \ldots, V_{nk_n}\rangle$ be a path of T. By FBT (V_{jk}) we mean a subtree of T, the vertices of which is determined by the set

$$\{V_{0k_0}, \ldots, V_{jk_j}\}\cup\bigcup_{t=j+1}^{n}\bigcup_{r=(k_j-1)2^{t-j}+1}^{k_j2^{t-j}}\{V_{tr}\}$$

and the set of edges is defined in the natural way; in other words, $\mathrm{FBT}(V_{jk})$ is determined by those paths of T, the initial segment of which is $\langle V_{0k_0}, \ldots, V_{jk_j}\rangle$ and are continued in all possible ways allowed by T.

2.4. Let $n\in\omega$ and T be a full binary tree of level n. We can define a mapping $\tau_1: B_n/\sim \to \mathrm{Sub}\, T$ by the following recurrence. Let $s_i^{\alpha_i}=s_i$ if $\alpha_i=1$ and otherwise $s_i^{\alpha_i}=\bar{s}_i$.

(i) $\tau_1(0)=\emptyset$, $\tau_1(1)=T$.

(ii) If $f=s_1^{\alpha_1}\cdot\ \ldots\ \cdot s_n^{\alpha_n}\in B_n$, then let $p=\langle V_{01}, \ldots, V_{nk_n}\rangle$ be that path of T for which $\langle V_{jk_j}, V_{(j+1)(k_j+1)}\rangle$ is labelled by $s_{j+1}^{\alpha_{j+1}}$ for all $j(0\leqq j\leqq n-1)$ and define $\tau_1(f)=p$.

(iii) Let $g=f_1+f_2+\ldots+f_m$ where each f_j is of the form $s_{j_1}^{\alpha_{j_1}}\cdot\ \ldots\ \cdot s_{j_n}^{\alpha_{j_n}}$ and define

$$\tau_1(g)=\bigcup_{j=1}^{m}\tau_1(f_j).$$

Since the cardinalities of $B_n/\sim$ and $\mathrm{Sub}\, T$ are equal, and every $g\in B_n$ has a form, determined uniquely up to the ordering of the variables, required by the clauses of the recursion, it follows that τ_1 is one-one and onto.

Let us define $\tau_0: B_n/\sim \to \mathrm{Sub}\, T$ by $\tau_0(f)=\overline{\tau_1(f)}$ where $\overline{\tau_1(f)}$ denotes a subtree of T determined by all paths of T which is not contained in $\tau_1(f)$; i.e. by the complement of $\tau_1(f)$ with respect to P_T. We have immediately,

Lemma 1. For all $f\in B_n/\sim$

(i) $\tau_0(f)=\tau_1(\bar{f})$,

(ii) $\tau_1(f)=\overline{\tau_1(\bar{f})}$.

Lemma 2 [4, Theorem 1]. Let $T_1\in\mathrm{Sub}\, T$ and assume, that T_1 is determined by the set of paths $\{p_1, \ldots, p_r\}$ and let $s_{1j}^{\alpha_{1j}}, \ldots, s_{nj}^{\alpha_{nj}}$ be the labels associated to the edges in p_j. Then,

$$\tau_1\left(\sum_{k=1}^{r}\left(\prod_{j=1}^{n}s_{jk}^{\alpha_{jk}}\right)\right)=T_1.$$

We call $\tau_1^{-1}(T_1)=\sum_{k=1}^{r}\left(\prod_{j=1}^{n}s_{jk}^{\alpha_{jk}}\right)$ the function to which T_1 is associated. Using Lemma 1 above, the dual of this assertion is easily obtained. In the sequel when speaking about associating a tree T to a function $f\in\mathscr{B}$ it will always mean the tree assigned by τ_1. (The duals of the assertions will not be mentioned because of being obtainable immediately.)

2.5. Let $f\in\mathscr{B}$. We say, that f does not depend on the variable $s_j\in S$, in other words, s_j is dummy for f, iff s_j occurs in f and for all $\xi, \xi'\in{}^{\omega}2$ for which $\xi'_j=1-\xi_j$ and $\xi'_k=\xi_k$ if $k\neq j$ we have $f(\xi)=f(\xi')$. It is easy to construct an algorithmic function δ, such that for all $f\in\mathscr{B}$, $\delta(f)$ is the set of variables occuring in f which are not dummy for f. Clearly, dummy variables do not effect the values of functions and thus they can freely be omitted or introduced when necessary. Let $p_1=\langle V_{0k_0}, \ldots, V_{jk_j}, V_{(j+1)k_{j+1}}, \ldots, V_{nk_n}\rangle$ and $p_2=\langle V_{0k_0}, \ldots, V_{jk_j}, V_{(j+1)l_{j+1}}, \ldots, V_{n,l_n}\rangle$ be two paths in a full binary tree T. We say, that p_1 and p_2 are amicable paths w.r.t. j iff all pairs of edges of the form $\langle V_{rk_r}, V_{(r+1)k_{r+1}}\rangle$ and $\langle V_{rl_r}, V_{(r+1)l_{r+1}}\rangle$

are labelled by the same label (which, of course depends on r) provided $r \neq j$ and either $l_{j+1}=k_{j+1}+1$ or $k_{j+1}=l_{j+1}+1$.

A path $p=\langle V_{0k_0}, \ldots, V_{j,k_j}, \ldots, V_{n,k_n}\rangle$ goes through V_{r,k_r} iff for some $j\ (0 \leqq j \leqq n)\ r=j$.

By definitions, we have

Lemma 3 [2, Special case of Theorem 15]. Let $f \in \mathscr{B}$ and assume that $T_1=\langle V_1, E_1\rangle$ is the tree associated to f. Then, for some $j\ (1 \leqq j \leqq n)$, s_j is dummy for f iff for all k such that $V_{(j-1)k} \in V_1$, all amicable paths w.r.t. $j-1$ going through $V_{(j-1)k}$ are paths of T_1.

2.6. Let $f, g \in \mathscr{B}$. We shall use the following notations: Δ_{fg} for $\delta(f) \cap \delta(g)$, the set of variables which are not dummy in both f and g. Let $\Phi_{fg}=\delta(f)-\Delta_{fg}$ and $\Gamma_{fg}=\delta(g)-\Delta_{fg}$, the sets of variables which are not dummy for f but do not occur in g and for g but do not occur in f, respectively. For the sake of convenience, we shall denote the elements of Δ_{fg} by $x_0, x_1, \ldots$, the elements of Φ_{fg} by $y_0, y_1, \ldots$ and the elements of Γ_{fg} by $z_0, z_1, \ldots$ throughout the paper; e.g. any appearence of x_j will always be meant as an element of $\Delta_{fg} \cap S$ e.t.c. Moreover, we tacitly assume that an ordering is fixed on these sets.

Since for given $f, g \in \mathscr{B}$, the case when $\Delta_{fg}=\emptyset$ is of no interest from our point of view, i.e. from the point of view of interpolants, we shall suppose that $\Delta_{fg} \neq \emptyset$ and distinguish the following four cases:

Case 1: $\Phi_{fg}=\Gamma_{fg}=\emptyset$.
Case 2: $\Phi_{fg} \neq \emptyset, \Gamma_{fg}=\emptyset$.
Case 3: $\Phi_{fg}=\emptyset, \Gamma_{fg} \neq \emptyset$.
Case 4: $\Phi_{fg} \neq \emptyset, \Gamma_{fg} \neq \emptyset$.

Let $f, g \in \mathscr{B}$. We shall supply both f and g with all variables from $\Delta_{fg} \cup \Phi_{fg} \cup \Gamma_{fg}$. One can distinguish the functions obtained in this way by $\tilde{f}$ and $\tilde{g}$, however, such distinction is not necessary. Indeed, by definition, the variables of Φ_{fg} will be dummy for g (and that of Γ_{fg} for f), hence f and $\tilde{f}$ (similarly, g and $\tilde{g}$) do represent the same equivalence class, thus, by our agreement on terminology, we can choose $\tilde{f}$ as the representative of that class. In fact, we shall do, and simply write f for $\tilde{f}$ (g for $\tilde{g}$). We shall fix an ordering of the variables occuring in f and g as follows: all elements of Δ_{fg} precede all elements of Φ_{fg} which, in turn, precede all elements of Γ_{fg} while we keep the previously fixed orderings inside Δ_{fg}, Φ_{fg} and Γ_{fg}. By this fixing of ordering, the construction of trees associated to f and g will be definitive.

Let $n=\operatorname{card}(\Delta_{fg} \cup \Phi_{fg} \cup \Gamma_{fg})$ and $i=\operatorname{card} \Delta_{fg}$ (recall, that $\Delta_{fg} \neq \emptyset$, hence $1 \leqq i \leqq n$ follows) and consider a full binary tree T of level n. For f, let $T_f=\langle V_f, E_f\rangle$ be that subtree of T which is associated to f. We introduce the following notations:

$$\mathscr{V}(f) = \{V_{ik} | V_{ik} \in V_f,\ 1 \leqq k \leqq 2^i\},$$

$$\mathscr{U}(f) = \{V_{ik} | V_{ik} \in \mathscr{V}(f) \quad \text{and} \quad FBT(V_{ik}) \notin \operatorname{Sub} T_f, \quad 1 \leqq k \leqq 2^i\}$$

$$\mathscr{W}(f) = \begin{cases} \mathscr{V}(f)-\mathscr{U}(f) & \text{provided} \quad i \neq n, \\ \mathscr{V}(f) & \text{otherwise.} \end{cases}$$

In the rest of the paper we shall keep the reference of the (lower case) letter i fixed, namely, $i=\operatorname{card} \Delta_{fg}$ and every occurence of i not in English words will always refer to this cardinality.

3. Existence of interpolants

3.1. Let $f, g \in \mathscr{B}$. We write $f \leqq g$ iff $\Delta_{fg} \neq \emptyset$ and for all $\xi \in {}^{\omega}2, f(\xi)=1$ entails $g(\xi)=1$; and $f<g$ iff $f \leqq g$ but $f \neq g$. The following assertion is immediate by definitions.

Lemma 4. Let $f, g \in \mathscr{B}$ and assume that T_f and T_g are the trees associated to f and g, respectively. Then $f \leqq g$ iff $T_f \in \operatorname{Sub} T_g$; in particular, $f<g$ iff $T_f \subset T_g$.

From now on, we shall fix (arbitrarily) $f, g \in \mathscr{B}$ such that $f<g, f \neq 0, g \neq 1$. All assertions in the rest are valid under these assumptions only, but, for the sake of being short we shall omit them everywhere when stating lemmata or theorems formally. Accordingly, every formal assertion is to be read as "If $f, g \in \mathscr{B}, f<g$, $f \neq 0, g \neq 1$ then" followed by the assertion written as such. This remark applies also for definitions.

First we set $I_{fg}=\{h \mid h \in \mathscr{B}, f<h, h<g$ and $\delta(h) \leqq \Delta_{fg}\}$. We say, that $h \in \mathscr{B}$ is an interpolant of f and g iff $h \in I_{fg}$. By Lemma 4, we have

Corollary 5. Let $h \in \mathscr{B}$ and T_f, T_g, T_h be the trees associated to f, g, h, respectively. Then,

(1) $h \in I_{fg}$ implies $T_f \subset T_h \subset T_g$, and
(2) $T_f \subset T_h \subset T_g$ and $\mathscr{W}(h)=\mathscr{V}(h)$ together imply $h \in I_{fg}$.

The following two lemmata readily follow from definitions by Lemma 4 and Corollary 5.

Lemma 6. Let $h \in \mathscr{B}$ and $h \in I_{fg}$. Then,

(1) $\mathscr{V}(f) \leqq \mathscr{V}(h)$,
(2) $\mathscr{W}(f) \subset \mathscr{W}(h)$,
(3) $\mathscr{V}(h)=\mathscr{W}(h)$,
(4) $\mathscr{W}(h) \leqq \mathscr{W}(g)$, and
(5) $\mathscr{V}(h) \subset \mathscr{V}(g)$.

Lemma 7. Let $h \in \mathscr{B}$. If

(1) $\mathscr{W}(f) \subset \mathscr{W}(h)$,
(2) $\mathscr{V}(h)=\mathscr{W}(h)$, and
(3) $\mathscr{V}(h) \subset \mathscr{V}(g)$

are satisfied, then $h \in I_{fg}$.

3.2. Recall that $\Phi_{fg}=\Gamma_{fg}=\emptyset$ in Case 1; $\Phi_{fg} \neq \emptyset, \Gamma_{fg}=\emptyset$ in Case 2; $\Phi_{fg}=\emptyset$, $\Gamma_{fg} \neq \emptyset$ in Case 3; and $\Phi_{fg} \neq \emptyset, \Gamma_{fg} \neq \emptyset$ in Case 4.

Lemma 8.

(1)	$\mathscr{U}(f)=\mathscr{U}(g)=\emptyset$	in Case 1.
(2)	$\mathscr{U}(f) \neq \emptyset$	in Cases 2 and 4,
	$\mathscr{U}(f)=\emptyset$	in Case 3.
(3)	$\mathscr{U}(g) \neq \emptyset$	in Cases 3 and 4,
	$\mathscr{U}(g)=\emptyset$	in Case 2.
(4)	$\mathscr{W}(g)=\mathscr{V}(g)$	in Cases 1 and 2.
(5)	$\mathscr{W}(f)=\mathscr{V}(f)$	in Cases 1 and 3.
(6)	$\mathscr{W}(g)-\mathscr{V}(f) \neq \emptyset$	in Case 1.
(7)	$\mathscr{U}(g)-\mathscr{V}(f) \neq \emptyset$	in Cases 3 and 4.

Proof. All statements exept (7) in Case 4 readily follow from Lemma 3 by definitions.

For proving (7) in Case 4, let us suppose, that $\mathcal{U}(g)-\mathcal{V}(f)=\emptyset$ and let $V_{ij}\in\mathcal{U}(g)$. We have either $V_{ij}\in\mathcal{U}(f)$ or $V_{ij}\in\mathcal{W}(f)$, immediately. Let us suppose first, that $V_{ij}\in\mathcal{U}(f)$ and let $k=\text{card}\,\Phi_{fg}$. Since f does not depend on elements of Γ_{fg}, there exists an l $(1\leqq l\leqq 2^{i+k})$, by Lemma 3, such that $\text{FBT}(V_{(i+k)l})\in\text{Sub}\,T_f$ (where T_f is the tree associated to f). On the other hand, since g does depend on elements of Γ_{fg}, it is impossible, again by Lemma 3, that the same is true for T_g (T_g being associated to g); i.e. there exist some vertices in $\text{FBT}(V_{(i+k)l})$ which are not contained in T_g. It follows, that $T_f\not\subset T_g$, a contradiction to Lemma 4. If $V_{ij}\in\mathcal{W}(f)$ then, using a similar argument, the assertion follows.

The next theorem gives necessary and sufficiant conditions under which proper interpolants exist.

Theorem 9. $I_{fg}\neq\emptyset$ iff $\text{card}\,(\mathcal{W}(g)-\mathcal{V}(f))\geqq\alpha$, where $\alpha=2$ in Case 1, $\alpha=1$ in Cases 2 and 3 and $\alpha=0$ in Case 4.

Proof. Let T_f and T_g be the trees associated to f and g, respectively.

For Cases 2 and 4, let T_1 be the tree obtained from T_f by adjoining $\text{FBT}(V_{ij})$ for all $V_{ij}\in\mathcal{U}(f)$ to T_f. By Lemma 8 (2), we have $\mathcal{U}(f)\neq\emptyset$ and hence, $T_f\subset T_1$ in both cases. In Case 4, $T_1\subset T_g$ follows from Lemma 8 (7). In Case 2, $\mathcal{W}(g)-\mathcal{V}(f)\neq\emptyset$ by assumption, thus $T_1\subset T_g$. Let h be the function to which T_1 is associated. By the construction of T_1, we have $\mathcal{W}(h)=\mathcal{V}(h)$, hence $h\in I_{fg}$, by Corollary 5 (2).

For Cases 1 and 3, let T_1 be constructed from T_f by adding to T_f the tree $\text{FBT}(V_{ij})$ for some $V_{ij}\in\mathcal{W}(g)-\mathcal{V}(f)$. Since $\mathcal{W}(g)-\mathcal{V}(f)$ is not empty by assumption, we have immediately, that $T_f\subset T_1$ (recall, that $\text{FBT}(V_{ij})$ is the path ending in V_{ij} in Case 1). In Case 3, $T_1\subset T_g$ is obtained by Lemma 8 (7), while in Case 1, this proper inclussion is entailed by the assumption, namely, by the fact, that $\mathcal{W}(g)-\mathcal{V}(f)-\{V_{ij}\}\neq\emptyset$ (where V_{ij} is the vertex used in the construction of T_1). Again, denoting by h the function to which T_1 is associated, $h\in I_{fg}$ follows from Corollary 5 (2) since $\mathcal{W}(h)=\mathcal{V}(h)$.

To prove the converse, let $I_{fg}\neq\emptyset$ and assume that $h\in I_{fg}$.

Case 1. $\text{card}\,(\mathcal{V}(g)-\mathcal{V}(h))\geqq 1$ and $\text{card}\,(\mathcal{V}(h)-\mathcal{V}(f))\geqq 1$ thus $\text{card}\,(\mathcal{W}(g)-\mathcal{V}(f))\geqq 2$ by Lemma 8 (4).

Case 2. $\mathcal{V}(f)\subseteqq\mathcal{V}(h)=\mathcal{W}(h)$ by Lemma 6 (1 and 3); $\mathcal{V}(h)\subset\mathcal{V}(g)$ by Lemma 6 (5) and $\mathcal{V}(g)=\mathcal{W}(g)$ by Lemma 8 (4). Summarizing up, $\mathcal{V}(f)\subset\mathcal{W}(g)$ and hence $\text{card}\,(\mathcal{W}(g)-\mathcal{V}(f))\geqq 1$.

Case 3. $\mathcal{V}(f)=\mathcal{W}(f)$ by Lemma 8 (5), $\mathcal{W}(f)\subset\mathcal{W}(h)=\mathcal{V}(h)$ by Lemma 6 (2 and 3) and finally, $\mathcal{W}(h)\subseteqq\mathcal{W}(g)$ by Lemma 6 (4). We have then $\mathcal{V}(f)\subset\mathcal{W}(g)$ which implies $\text{card}\,(\mathcal{W}(g)-\mathcal{V}(f))\geqq 1$.

3.3. We present here some counterexamples thus illustrating the very nature of proper interpolants.

Let the following functions be given: $f_1=x_1\cdot x_2$, $g_1=x_1\cdot x_2+\bar{x}_1\cdot\bar{x}_2$; $f_2=x_1\cdot x_2\cdot y_1$, $g_2=x_1\cdot x_2$; and $f_3=x_1\cdot x_2$, $g_3=x_1\cdot x_2+\bar{x}_1\cdot\bar{x}_2\cdot z_1$. The trees associated to these functions are indicated in bold line by Figs 1, 2 and 3, respectively.

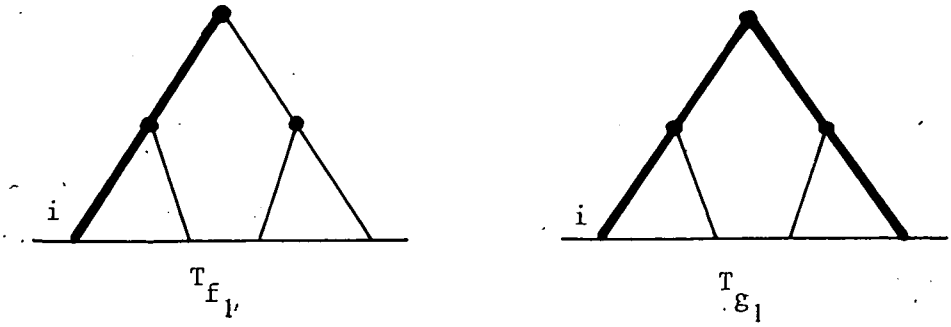


Fig. 1

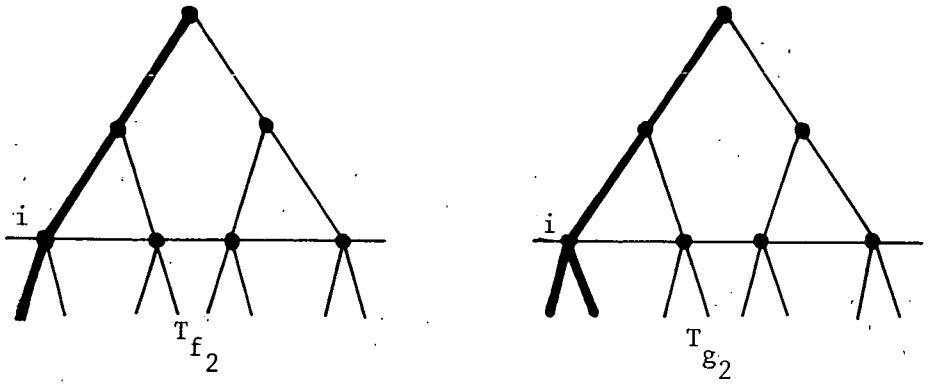


Fig. 2

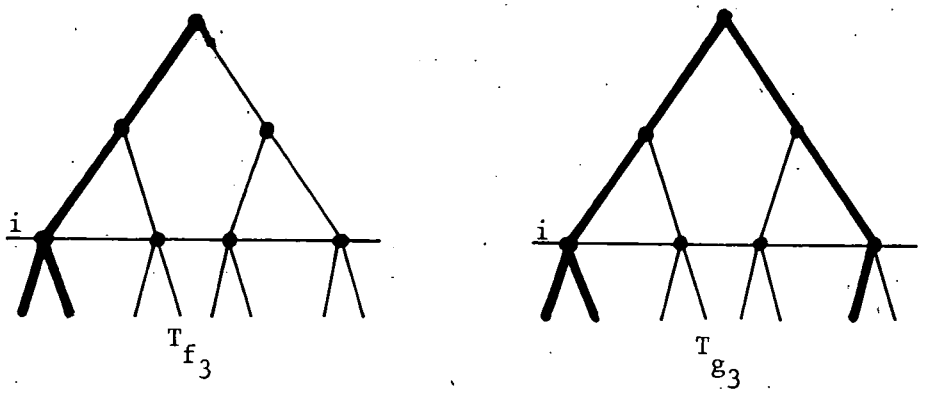


Fig. 3

It is clear, that $\Phi_{f_1 g_1}=\Gamma_{f_1 g_1}=\emptyset$; $\mathcal{V}(g_1)-\mathcal{V}(f_1)=\mathcal{W}(g_1)-\mathcal{V}(f_1)\neq\emptyset$ and card $(\mathcal{W}(g_1)-\mathcal{V}(f_1))=1$, nevertheless $I_{f_1 g_1}=\emptyset$. Similarly, $\Phi_{f_2 g_2}=\{y_1\}$, $\Gamma_{f_2 g_2}=\emptyset$ and $\mathcal{V}(g_2)-\mathcal{V}(f_2)=\mathcal{W}(g_2)-\mathcal{V}(f_2)=\emptyset$ and $I_{f_2 g_2}=\emptyset$. Finally, $\Phi_{f_3 g_3}=\emptyset$, $\Gamma_{f_3 g_3}=\{z_1\}$ and $\mathcal{W}(g_3)-\mathcal{V}(f_3)=\emptyset$, thus $I_{f_3 g_3}=\emptyset$.

4. The number of interpolants

4.1.

Theorem 10. Let $m=\operatorname{card}(\mathscr{W}(g)-\mathscr{V}(f))$. Then,

$$\operatorname{card}(I_{fg}) = 2^m-\alpha$$

where $\alpha=2$ in Case 1, $\alpha=1$ in Cases 2 and 3 and $\alpha=0$ in Case 4.

Proof. Let $M=\mathscr{W}(g)-\mathscr{V}(f)$. In all cases, if $M\neq\emptyset$ ($M=\emptyset$ can occur in Cases 2, 3 and 4 only, by Lemma 8 (6)), then the whole set I_{fg} can be constructed by the following recurrence.

Let us denote by T_1 the tree obtained by adjoining FBT(V_{ik}) to the tree T_f associated to f for all $V_{ik}\in\mathscr{U}(f)$. Obviously, $T_f\subseteqq T_1$.

Let T_2 be a tree such that $T_1\subseteqq T_2\subseteqq T_g$ and T_2 is associated to an interpolant h_2 of f and g (or, to f if $T_2=T_1=T_f$) and suppose, that $V_{ij}\in M$. Let T_3 be constructed from T_2 by adding FBT(V_{ij}) to T_2. Clearly, $T_f\subset T_3\subseteqq T_g$ and, for the function h_3 to which T_3 is associated, $\mathscr{W}(h_3)=\mathscr{V}(h_3)$. It follows from Corollary 5, that $h_3\in I_{fg}$ iff $T_3\subset T_g$, and from Lemma 4, that $h_2<h_3$. Let $M_1=M-\{V_{ij}\}$ and repeat this procedure with $V_{il}\in M_1$ and with T_3 (in place of T_2) until M is emptied.

Summarizing up, starting from T_1 and taking in all possible ways one, two, ..., m distinct elements from M (provided $M\neq\emptyset$) and proceeding as described above we can produce a set of functions $I=\{t_1, ..., t_r\}$ and it follows from the construction, that $I\cup\{f,g\}=I_{fg}\cup\{f,g\}$, i.e. any function which can be constructed in this way is either an element of I_{fg} or of $\{f,g\}$. Since one, two, ..., m distinct elements can be chosen from M in $\binom{m}{1}, \binom{m}{2}, ..., \binom{m}{m}$ possible ways, and $\sum_{j=1}^{m}\binom{m}{j}=$ $=2^m-1$, we have $\operatorname{card} I=2^m$.

It remains to investigate whether f and g do or do not appear in I. This will be done case by case.

Case 1. We have $\mathscr{U}(f)=\emptyset$ by Lemma 8 (1), hence $T_1=T_f$, i.e. $f\in I$. On the other hand, taking all elements from M, we obviously obtain a tree identical to T_g, thus $g\in I$. All the other elements of I are proper interpolants, indeed, that is $I_{fg}=I-\{f,g\}$. It follows, that $\operatorname{card}(I_{fg})=2^m-2$.

Case 2. By Lemma 8 (2), $\mathscr{U}(f)\neq\emptyset$ which entails $T_f\subset T_1$, i.e. the function to which T_1 is associated is in I_{fg} (cf. the proof of Theorem 9). Taking all elements from M in the procedure above, we arrive to T_g by Lemma 8 (3), hence $g\in I$. We have $I_{fg}=I-\{g\}$, hence $\operatorname{card}(I_{fg})=2^m-1$.

Case 3. $\mathscr{U}(f)=\emptyset$, by Lemma 8 (2), which implies $T_1=T_f$ and thus $f\in I$. Let T_r be the tree obtained by the procedure using all elements of M. Then by Lemma 8 (3) $T_r\subset T_g$. That is $g\notin I$, $I_{fg}=I-\{f\}$ and we have $\operatorname{card}(I_{fg})=2^m-1$.

Case 4. Since $\mathscr{U}(f)\neq\emptyset$ by Lemma 8 (2), we have $T_f\subset T_1$, i.e. $f\notin I$. On the other hand, taking all elements in M and constructing the tree T_r by the procedure, by Lemma 8 (3), $T_r\subset T_g$ holds. We obtain, that $g\notin I$ and so $I_{fg}=I$.

4.2. By a chain of interpolants we mean a finite sequence of distinct functions $h_0, \ldots, h_t$ such that the following clauses are satisfied:

(1) $h_0=f, h_t=g$,

(2) $h_j \in I_{h_{j-1}h_{j+1}}$ for $1<j<t$.

A chain $h_0, \ldots, h_t$ of interpolants is maximal iff for every j $(0 \leqq j<t)$, $I_{h_j h_{j+1}}=\emptyset$.

Corollary 11. Every maximal chain of interpolants of f and g has length $\operatorname{card}(\mathscr{W}(g)-\mathscr{V}(f))+\beta$ where $\beta=1$ in Case 1, $\beta=2$ in Cases 2 and 3 and $\beta=3$ in Case 4.

Proof. Immediate by the proof of Theorem 10.

Fig. 4 below indicates all of the four cases. h_1 stands for the function obtained from T_1 in the proof of the previous theorem, and h_M denotes the function which is constructed by using the whole set M.

Corollary 12. The set of all maximal chains of interpolants of f and g has cardinality given by

$$(\operatorname{card}(\mathscr{W}(g)-\mathscr{V}(f)))!.$$

Proof. The second (in Cases 1, 3) or the third member (in Case 2, 4) of a particular maximal chain is obtained by using exactly one element from the set $M=\mathscr{W}(g)-\mathscr{V}(f)$; this element can be taken in $\operatorname{card} M$ different ways. The next member of the chain can be taken in $\operatorname{card} M-1$ different ways, and so on. The assertion follows by induction.

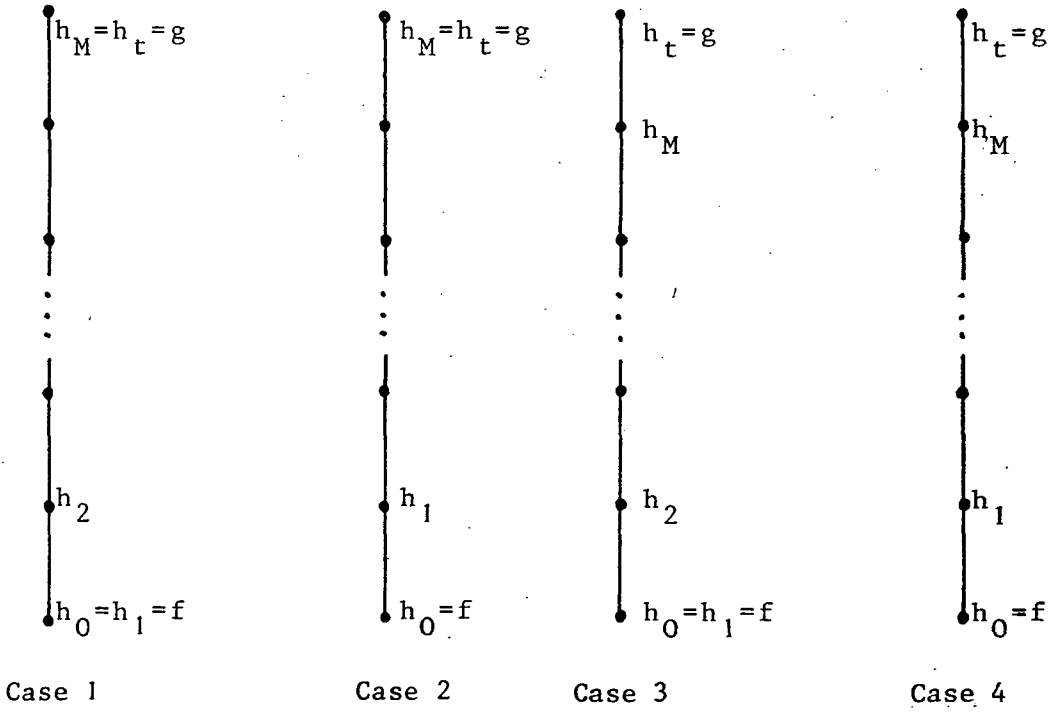


Fig. 4

5. The algebra of interpolants

Theorem 13. The algebra $\mathscr{I}=(I_{fg}\cup\{f,g\},\cdot,+,f,g)$ is a distributive sublattice of the Boolean algebra $\mathscr{B}=(B/\sim,\cdot,+,^-,0,1)$ with zero element f and unit element g.

Proof. The only thing to be proved is that $I_{fg}\cup\{f,g\}$ is closed under $+$ and $\cdot$. This is, however, obvious from the construction outlined in the proof of Theorem 10.

It is relatively easy to show using Lemma 1 and the construction of I_{fg}, that $I_{fg}\cup\{f,g\}$ is not closed under negation: the algebra $\mathscr{I}$ is not a Boolean algebra.

Theorem 14. Let $h_0,h_1\in I_{fg}\cup\{f,g\}$. Then in the Boolean algebra $\mathscr{B}$, the two equivalence classes $[h_0]$ and $[h_1]$ are identical iff their representatives h_0 and h_1 are such; i.e. $[h_0]=[h_1]$ iff $h_0=h_1$.

Proof. Obvious, by Lemma 2 and the construction of I_{fg}.

6. Conclusions

By using the isomorphism between $\mathscr{F}$ and $\mathscr{B}$, to every $\varphi\in\mathscr{F}$, there corresponds a class in $\mathscr{B}$ denoted by f_φ, and conversely, for $f\in\mathscr{B}$ one can associate an element φ_f in $\mathscr{F}$.

By a zero order model A we simply mean a subset of the set of sentence symbols S. Observe, that every assignment $\xi\in{}^\omega 2$ represents a zero order model in the sense of [1]: let $A_\xi=\{s_i|s_i\in S \text{ and } \xi_i=1\}$ where ξ_i is the i-th component of ξ. The converse is also valid: every model $A\subset S$ can be associated by an assignment ξ_A defined by

$$\xi_i=\begin{cases}1 & \text{iff } s_i\in A,\\ 0 & \text{otherwise.}\end{cases}$$

Clearly, for every φ and A, we have $A\models\varphi$ iff $f_\varphi(\xi_A)=1$. We set $A_\varphi=\{A|A\subset S, A\models\varphi\}$.

Let ξ and ζ be two assignments and $\varphi\in\mathscr{F}$. We say that the models A_ξ and A_ζ are φ-equivalent with respect to a subset Δ of $\delta(f)$, the set of nondummy variables of f, iff $A_\xi\cap\Delta=A_\zeta\cap\Delta$. This is indeed an equivalence relation and being so we can set for $A\subset S$, $\varphi\in\mathscr{F}$ and $\Delta\subset\delta(f)$:

$[A]^\Delta_\varphi=\{B|B\subset S \text{ and } B \text{ is } \varphi\text{-equivalent to } A \text{ with respect to } \Delta\}$.

Notice, that by choosing $\Delta=\delta(f)$, the class $[A]^{\delta(f_\varphi)}_\varphi$ is represented by one path in the tree T_{f_φ} associated to f_φ.

Let $f,g\in\mathscr{B}$ and consider the sets of variables, Δ_{fg}, Φ_{fg} and Γ_{fg}. Then, clearly, $\operatorname{card}\mathscr{V}(f)=\operatorname{card}(\{[A]^{\Delta_{fg}}_{\varphi_f}\})$ and similarly, $\operatorname{card}\mathscr{V}(g)=\operatorname{card}(\{[A]^{\Delta_{fg}}_{\varphi_g}\})$, that is, $\mathscr{V}(f)$ and $\mathscr{V}(g)$ identify all φ_f-equivalent and φ_g-equivalent classes of models with respect to the common set of nondummy variables of f and g, Δ_{fg}, respectively.

By definition, $\mathscr{U}(f)\cap\mathscr{W}(f)=\emptyset$, $\mathscr{U}(f)\cup\mathscr{W}(f)=\mathscr{V}(f)$ and similar equations hold for g. If for some k, $V_{ik}\in\mathscr{W}(g)$, then FBT (V_{ik}) is a subtree of the tree T_g associated to g, and all paths of FBT (V_{ik}) represent the same φ_g-equivalence class of models with respect to the set of all nondummy variables of g, $\delta(g)$, while

if $V_{ik} \notin \mathcal{W}(g)$ and hence $V_{ik} \in \mathcal{U}(g)$, then the paths of T_g going through V_{ik} will represent different classes (with respect to $\delta(g)$). We say that A is a respectable model (for φ_g) iff

$$[A]^{\Delta_{fg}}_{\varphi_g} = [A]^{\delta_{(g)}}_{\varphi_g}.$$

Since interpolants of f and g (hence of φ_f and φ_g) can depend on the variables of Δ_{fg} only, respectable models for φ_g are exactly the ones which are of interest from the point of view of interpolants.

The φ_g-equivalence classes of respectable models for φ_g, however, are identified by elements of $\mathcal{W}(g)$, according to the remark above.

Let us introduce the following notations:

$A^{\text{resp}}_{\varphi_g} = \{[A]^{\Delta_{fg}}_{\varphi_g} |\ A$ is a respectable model for $g\}$ and $A_{\varphi_f} = \{[A]^{\delta_{(f)}}_{\varphi_f} |\ A \in A_{\varphi_g}\}$. Then, the set $\mathcal{W}(g) - \mathcal{V}(f)$, playing a central role in our investigations, identifies those respectable model classes for φ_g which are not models of φ_f, and $\text{card}\,(\mathcal{W}(g) - \mathcal{V}(f)) = \text{card}\,(A^{\text{resp}}_{\varphi_g} - A_{\varphi_f})$ hence a reformulation of Theorem 9 in model theoretic terms can be easily obtained.

Summing up the results of the paper, for any two zero order formulae φ and ψ such that $\varphi \models \psi$, $\not\models \neg\varphi$, $\not\models \psi$, we can decide whether does or does not exist a proper interpolant for φ and ψ and if the answer is affirmative, we can give the number of equivalence classes of proper interpolants immediately, or we can construct the whole lattice of equivalence classes of interpolants when necessary. The method developed in the paper is much more effective (even if it is considered as inefficient in the more strict sense of [5]) than the one presented in [3].

Abstract

The number of equivalence classes of interpolants for arbitrarily given two zero order sentences are calculated using tree-theoretic arguments. As a by-product, the number of maximal chains and the algebraic structure of equivalence classes of interpolants are determined.

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

** DEPT. OF MATH. FACULTY OF CIVIL ENGENEERING
24000, SUBOTICA, YUGOSLAVIA

References

[1] C. C. Chang and H. J. Keisler, *Model Theory*, North-Holland Publ. Co. (Second Edition, 1977), Amsterdam, New York, Oxford.

[2] P. Ecsedi-Tóth, A Theory of Finite Functions, Part I: On Finite Trees Associated to Certain Finite Functions *Acta Cybernetica*, v. 6, 1983, pp. 213—225.

[3] P. Ecsedi-Tóth and L. Turi, On enumerability of interpolants, Proc. of Algebraic Conference '81, Novi Sad, (ed. K. Gilezan), Math. Inst. Novi Sad, Yugoslavia (1982), pp. 71—79.

[4] P. Ecsedi-Tóth and A. Varga and F. Móricz, On a Connection Between Algebraic and Graph-theoretic Minimization of Truth-functions, Coll. Math. Soc. J. Bolyai, Vol. 25. Algebraic Methods in Graph Theory, Szeged, Hungary (1978) (eds. L. Lovász and V. T-Sós) North-Holland Publ. Co., Amsterdam, New York, Oxford (1981) pp. 119—136.

[5] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman Publ. Co. San Francisco (1978).

Received Febr. 21, 1983.

A SZERKESZTŐ BIZOTTSÁG CÍME:

6720 SZEGED
SOMOGYI U. 7.

EDITORIAL OFFICE:

6720 SZEGED
SOMOGYI U. 7.
HUNGARY

INDEX—TARTALOM

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Gécseg Ferenc
A kézirat a nyomdába érkezett: 1983. november 23
Megjelenés: 1984. szeptember
Példányszám: 500. Terjedelem: 9,45 (A/5) ív
Készült monószedéssel, íves magasnyomással
az MSZ 5601 és az MSZ 5602—55 szabvány szerint
84-4957 — Szegedi Nyomda — F. v.: Dobó József igazgató