58725

felv

Tomus 4. Fasciculus 1.

ACTA CYBERNETICA

IN MEMORIAM LADISLAI KALMÁR

FORUM CENTRALE PUBLICATIONUM
CYBERNETICARUM HUNGARICUM

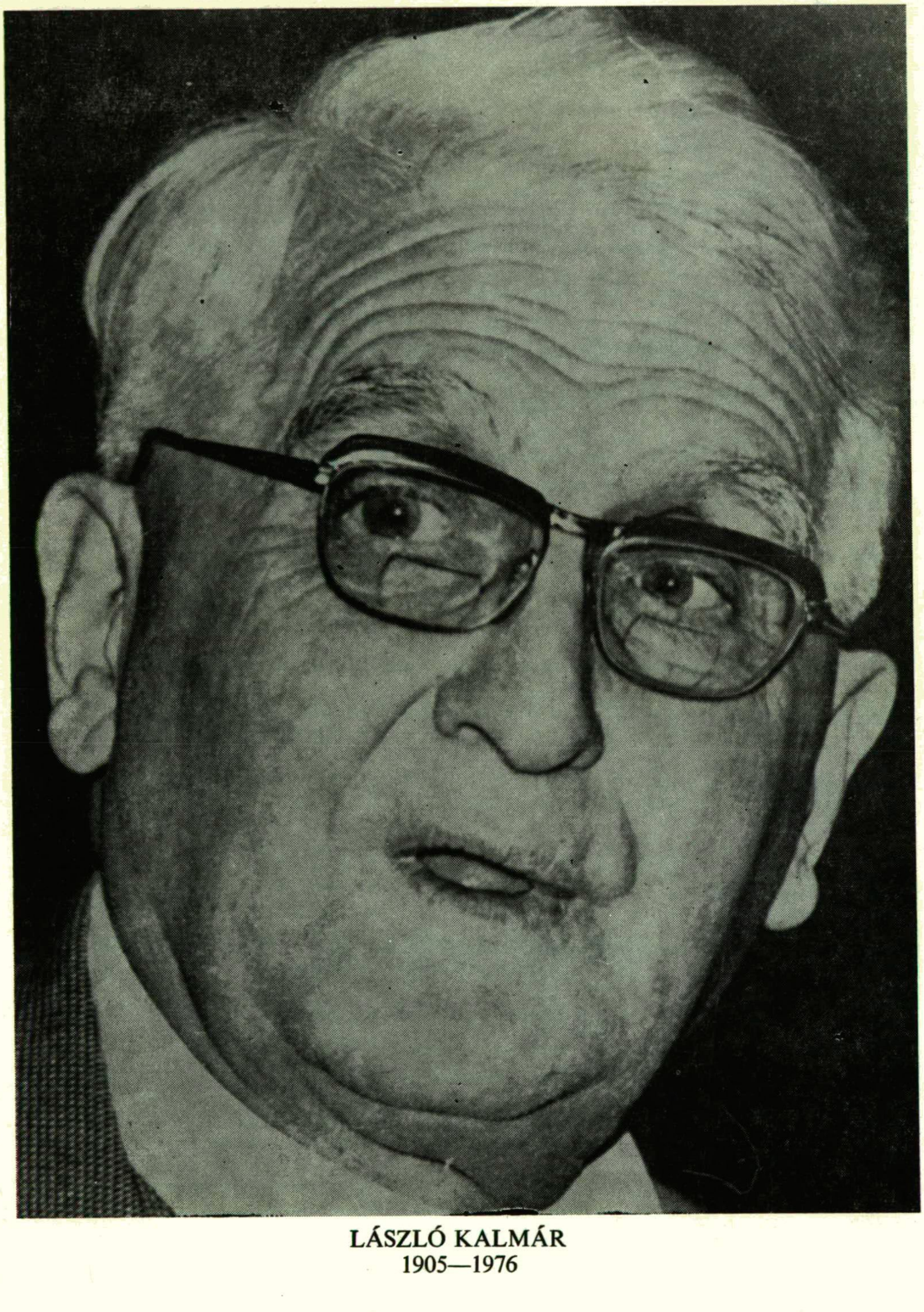
Szeged, 1978
Curat: Universitas Szegediensis de Attila József nominata

4. kötet 1. füzet

ACTA CYBERNETICA

A HAZAI KIBERNETIKAI KUTATÁSOK
KÖZPONTI PUBLIKÁCIÓS FÓRUMA

Szeged, 1978. december
A Szegedi József Attila Tudományegyetem gondozásában


LÁSZLÓ KALMÁR
1905—1976

The generalised completeness of Horn predicate-logic as a programming language*

By H. ANDRÉKA and I. NÉMETI

To the memory of Professor László Kalmár

Here we prove the "generalised" completeness of "Prolog-like" languages [1], [2] or "Horn-predicate logic as a programming language" [3], [4], [5], [6].

More precisely we prove the following. Let Fr be an arbitrary Herbrand-universe (in other words, Fr is a word algebra of an arbitrary finite type generated by the constant symbols). For any $f\colon Fr^n \to Fr$ Turing-computable partial function over Fr, there is a finite set C_f of Horn clauses over Fr (that is there are *no* other function or constant symbols in C_f but only those which occur in Fr) and a relation symbol F_f such that C_f defines f over Fr, more precisely:

$$(\forall \bar{\alpha}, \beta \in Fr)[f(\bar{\alpha}) = \beta \quad \text{iff} \quad C_f \models F_f(\bar{\alpha}, \beta)]$$

where $\bar{\alpha}$ is a vector of elements from Fr.

This means, that if we are given an arbitrary Herbrand-universe Fr and an arbitrary computable task over Fr, then we can write a Prolog program which solves this task and which does not contain other function or constant symbols but only those which occur in Fr. This is somehow a statement about the *adequateness* of Horn logic as a programming language: Any computable problem can be formulated in Horn logic without using auxiliary function symbols. That is without "coding" the data to be processed.

A special case of this theorem was proved by Robert Hill (unpublished, personal communication). He proved the above statement for the case when Fr is the set of natural numbers together with the successor function and constant 0. The proof stated here is a generalisation of his one. In generalising any proof from the natural numbers to arbitrary Herbrand universes Fr the difficulty originates from the unfortunate fact, that — as far as we know — there is very little work done on the "nice" characterisation of the computable functions over Fr.

Another related result has recently been proved by Tärnlund, c.f. "Sten Ake Tärnlund: Logic Information Processing", University of Stockholm, report

*Part of the research for this paper was done during the author's stay in the Department of Artificial Intelligence, University of Edinburgh.

TRITA—IBADB$\overset{\varrho}{-}$1034, 1975—II—24. He proves that if we are given an arbitrary Herbrand-universe *Fr* together with a computable function f over it, then there exists a set of binary Horn-clauses C_f defining f. However in Tärnlund's paper C_f is defined over a Herbrand-universe which is definitely larger than *Fr*. (In defining C_f he makes extensive use of auxiliary function symbols.)*

The main result proved in this paper is that C_f can be defined over *Fr* itself; in other words, that we can dispense with the auxiliary function symbols.

Remark. We believe that an alternative (perhaps more natural) proof can be given by starting from Emden's work [7] and investigating the generalisation of Kleene's recursion equations to arbitrary word algebras. To this end first it should be proved that any Turing-computable partial function f over an arbitrary word-algebra *Fr* can be defined by such a finite system of Emden's modified recursion equations (see [7]), in which system all the constant functions belong to *Fr*.

Theorem. Let *Fr* be an arbitrary Herbrand-universe (that is a word-algebra of arbitrary finite type generated by the empty set, in other words: generated by the constant symbols of the type).

Now, for any finitary Turing-computable partial function f over *Fr* (f: $Fr^n \to Fr$) there is a finite set C_f of Horn clauses over *Fr* (that is all the function symbols occuring in C_f also occur in *Fr*), and a relation symbol F_f such that

$$(\forall \bar{\alpha}, \beta \in Fr)[f(\bar{\alpha}) = \beta \quad \text{iff} \quad C_f \models F_f(\bar{\alpha}, \beta)]$$

where $\bar{\alpha}$ is a vector of elements of *Fr*.

Moreover, C_f can be effectively computed from the Turing-definition of f.

Proof. Let ω denote the set of natural numbers. The idea of the proof is the following:

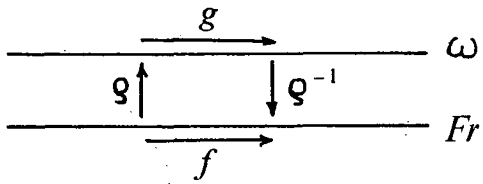


Fig. 1

First we define a one-one function ϱ from *Fr* onto ω, such that ϱ as well as ϱ^{-1} are Turing-computable. Now if f: $Fr^n \to Fr$ is Turing-computable, then $g = \varrho \circ f \circ \varrho^{-1}$ is a Turing-computable function on ω, and $f = \varrho^{-1} \circ g \circ \varrho$. But every Turing-computable function on ω is recursive. Thus *every Turing-computable function f over Fr is the image* by ϱ of some recursive function g over ω ($f = \varrho^{-1} \circ g \circ \varrho$). By this it is enough to prove for any recursive function g over ω, that the function $\varrho^{-1} \circ g \circ \varrho$ is Horn-definable over *Fr* (see figure).

Let the type t be denoted as: $t \overset{\mathrm{d}}{=} \{\langle f_j^i, i\rangle : i \leq k, j \leq m_i\}$. In other words: there are numbers k and m_i (for every $i \leq k$) such that f_j^i is the j-th i-ary function symbol for $k \geq i$, $j \leq m_i$. Note that $\{f_j^0 : j \leq m_0\}$ is the set of constant symbols.

Now we define the function ϱ. To this end we first define the auxiliary functions F and Sz by a simultaneous recursion.

* Thus Tärnlund's result is different from Hill's one in two respects:

1. Tärnlund says more than Hill by allowing arbitrary Herbrand-universes and using only binary Horn-clauses.

2. On the other hand Tärnlund says less than Hill, since he says nothing about the number of auxiliary functions symbols.

The intuition behind the following definitions of F and Sz is explained later in the proof of the first lemma.

The only important property of F is that F enumerates the word algebra Fr. Any other recursively defined function with this property could be substituted for F without changing the rest of the proof. The function Sz is only an auxiliary function in the definition of F. That is, we use Sz only to define F.

We define F by a definition scheme which can be translated into a definition for any given type, that is for any fixed numbers k and m_i. In this scheme the text: "for $i \leqq k$, $j \leqq m_i$, $0 < p \leqq i$: $f\ldots$" is written in a metalanguage and can be translated by copying "$f\ldots$" as many times as i's, j's and p's are possible.

$$F(0) \overset{\mathrm{d}}{=} f_0^0$$

$$F(n+1) \overset{\mathrm{d}}{=}$$

$$\overset{\mathrm{d}}{=} \begin{cases} \text{for } i \leqq k, j \leqq m_i \text{ and } 0 < p \leqq i: & \\ f_j^i(F(n_1), \ldots, F(n_p+1), F(0), \ldots, F(0)) & \text{if } F(n) = f_j^i(F(n_1), \ldots, F(n_i)) \text{ and} \\ & (\forall p < z \leqq i) Sz(n_z+1) \geqq Sz(n) \\ & \text{and } Sz(n_p+1) < Sz(n) \\ \text{for } i \leqq k, j < m_k: & \\ f_{j+1}^i(F(0), \ldots, F(0)) & \text{if } F(n) = f_j^i(F(n_1), \ldots, F(n_i)) \text{ and} \\ & (\forall 0 < z \leqq i) Sz(n_z+1) \geqq Sz(n) \\ \text{for } i < k: & \\ f_0^{i+1}(F(0), \ldots, F(0)) & \text{if } F(n) = f_{m_i}^i(F(n_1), \ldots, F(n_i)) \text{ and} \\ & (\forall 0 < z \leqq i) Sz(n_z+1) \geqq Sz(n) \\ f_0^0 & \text{if } F(n) = f_{m_k}^k(F(n_1), \ldots, F(n_k)) \text{ and} \\ & (\forall 0 < z \leqq k) Sz(n_z+1) \geqq Sz(n) \end{cases}$$

$$Sz(0) \overset{\mathrm{d}}{=} 0,$$

$$Sz(n+1) \overset{\mathrm{d}}{=} \begin{cases} Sz(n)+1, & \text{if } F(n) = f_{m_k}^k(F(n_1), \ldots, F(n_k)) \text{ and} \\ & (\forall 1 \leqq z < k) Sz(n_z+1) \geqq Sz(n), \\ Sz(n), & \text{otherwise.} \end{cases}$$

It is easy to see that the above simultaneously recursive definition is correct, that is it really defines the functions F and Sz.

In the following definitions we use the recursion theoretic μ-operator. Remember that $\mu x(R(x))$ is the smallest number x for which $R(x)$ is true.

$$E(n) \overset{\mathrm{d}}{=} \begin{cases} \text{True, if } (\forall j < n) F(j) \neq F(n), \\ \text{False, if otherwise} \end{cases}$$

$$S(\tau) \overset{\mathrm{d}}{=} F(\mu n(\mu k(F(k) = \tau) < n \ \&\ E(n))),$$

$$\xi(0) \overset{\mathrm{d}}{=} F(0),$$

$$\xi(n+1) \overset{\mathrm{d}}{=} S(\xi(n)),$$

$$\varrho(\tau) \overset{\mathrm{d}}{=} \mu n(\xi(n) = \tau).$$

Lemma. *a)* $\varrho: Fr \to \omega$ is one-one and onto,
b) ϱ and ϱ^{-1} are Turing computable.

Proof. ad *a)* Note, that any total function f with domain ω can be considered as a "listing" or an enumeration of the range of f.

Now, we define a system of subsets H_i $(i \in \omega)$.

$$H_{-1} \overset{\mathrm{d}}{=} \{f_0^0, \dots, f_{m_0}^0\},$$

$$H_{l+1} \overset{\mathrm{d}}{=} \{f_j^i(\tau_1, \dots, \tau_i): \tau_1, \dots, \tau_i \in H_l,\ i \leqq k,\ j \leqq m_i\}.$$

For each $i \in \omega$, on the set H_i a linear ordering can be defined in a natural way: For H_{-1}: $f_i^0 < f_j^0$ iff $i < j$. To define the ordering on H_{l+1}, suppose, that the ordering on H_l has been defined.

Now for any two elements of H_{l+1}:

$$f_j^i(\tau_1, \dots, \tau_i) < f_{j'}^{i'}(\tau_1', \dots, \tau_{i'}') \quad \text{iff} \quad \langle i, j, \tau_1, \dots, \tau_i\rangle < \langle i', j', \tau_1', \dots, \tau_{i'}'\rangle$$

according to the lexicographic ordering obtained from the ordering on natural numbers and the ordering $<$ defined on H_l.

It is easy to check, by the definition of F, that the function F first enumerates H_0 in accordance with the above defined ordering on H_0, then enumerates similarly H_1, then H_2... etc. Since $\bigcup_{i=1}^{\infty} H_i = Fr$, the function F enumerates the whole Fr. However, unfortunately, F might enumerate an element of Fr more than once, in other words, the function F is not one-one. To deal with this, the relation E marks those places in ω where an element occurs (is listed) first. The function ξ picks out only those occurrences (of elements of Fr) which are marked by E. Thus ξ is already one-one, while since F is onto, ξ is also onto.

ad *b)* From the fact that ξ is one-one it follows that $\varrho = \xi^{-1}$, and from their definition it is easy to see that both ϱ and ξ are Turing-computable. (For, from their definition it is easy to construct a computer program which computes ϱ and ξ.) And by this the lemma is proved.

Lemma. To every partial recursive function g over ω $g: \omega^n \to \omega$, the function $f \overset{\mathrm{d}}{=} \varrho^{-1} \circ g \circ \varrho$ is Horn-definable over Fr, that is there is a set of Horn-clauses C_f and a relation symbol F_f such that

$$(\forall \bar{\alpha}, \beta \in Fr)[\varrho^{-1} \circ g \circ \varrho \quad (\bar{\alpha}) = \beta \quad \text{iff} \quad C_f \models F_f(\bar{\alpha}, \beta)].$$

Proof. By the definition of recursive functions, it suffices to prove the above statement for the:

zero function $Z(x) \overset{\mathrm{d}}{=} 0$;

the successor function $S(x) \overset{\mathrm{d}}{=} x+1$;

the projection functions $I_m^n(x_1, \dots, x_n) \overset{\mathrm{d}}{=} x_m$, and to prove that if the above statement holds for the functions h, $g, g_1, \dots, g_n$ then it also holds for the functions obtained from these by

substitution $\quad f(x) \overset{d}{=} h(g_1(x), \dots, g_n(x))$

recursion $\quad f(x, 0) \overset{d}{=} g(x)$

$\quad f(x, n+1) \overset{d}{=} h(x, n, f(x, n))$

the μ-operator $\quad f(x) \overset{d}{=} \mu y(g(x, y) = 0)$.

Note, that μ has already been defined in the definition of the function ϱ. In writing Horn-clauses we use the notation of Kowalski [3].

a) *the zero function:*

$\varrho^{-1} \circ Z \circ \varrho$ is Horn definable:

$$C_z \overset{d}{=} \{F_z(x, f_0^0) \leftarrow\}$$

It is easy to see that C_z defines exactly the function $\varrho^{-1} \circ Z \circ \varrho$. Here we give the detailed proof of this statement, but we shall omit the proofs of the following statements about the successor function, etc. because they are mechanical analogues of the present one.

Now we prove that $C_z \models F_z(\tau, \sigma)$ iff $\sigma = f_0^0$.

1. for all $\tau \in Fr$, we immediately have $C_z \models F_z(\tau, f_0^0)$.

2. To prove the implication in the other direction.

Let $\sigma \neq f_0^0$, and $\tau, \sigma \in Fr$.

In this case $C_z \not\models F_z(\tau, \sigma)$, because we can construct a model of C_z in which $F_z(\tau, \sigma)$ fails. Let on the Herbrand-universe Fr the interpretation of the relation symbol F_z be the relation $R \overset{d}{=} \{\langle \tau, f_0^0 \rangle : \tau \in Fr\}$. In the model obtained this way C_z is valid while $F_z(\tau, \sigma)$ is clearly false. Thus, $C_z \not\models F_z(\tau, \sigma)$.

b) *the successor function:*

$\varrho^{-1} \circ S \circ \varrho$ is Horn definable:

This is the only more laborious step: Here we need an explicit and constructive description of the function ϱ. We shall, not do anything but translate the definition of ϱ into Horn-clausal form. To this end however we first have to "code" the natural numbers by elements of Fr. For any number $n \in \omega$, the symbol $\hat{n}$ stands for the code of n in Fr. We define the code recursively:

$$\hat{0} \overset{d}{=} f_0^0, \text{ and } \widehat{n+1} \overset{d}{=} f_0^1(\hat{n}).$$

Remark. If $m_1 = -1$ then let i be the smallest number such that $m_i \geqq 0$. Now $\widehat{n+1} \overset{d}{=} f_0^j(\hat{n}, f_0^0, \dots, f_0^0)$.

$$\begin{aligned} C_s \overset{d}{=} \{ & \leqq (x, x) \leftarrow, \\ & \leqq (x, y) \leftarrow \leqq (f_0^1 x, y), \\ & < (x, y) \leftarrow \leqq (f_0^1 x, y)\} \cup \end{aligned}$$

$\{F(f_0^0, f_0^0) \leftarrow\} \cup$

$$\Big\{F\big(f_0^1y, f_j^i(x_1,\ldots,x_{p-1},v,f_0^0,\ldots,f_0^0)\big) \leftarrow \quad F\big(y,f_j^i(x_1,\ldots,x_i)\big)\wedge\bigwedge_{z=1}^{i}F(y_z,x_z)\wedge$$

$$\bigwedge_{z=1}^{i}Sz(f_0^1y_z,w_z)\wedge Sz(y,w)\wedge\bigwedge_{z=p+1}^{i}\leqq(w,w_z)\wedge$$

$$<(w,w_p)\wedge F(f_0^1y,v)$$

$$:i\leqq k,\ j\leqq m_i,\ 0<p\leqq i\Big\}\cup$$

$$\Big\{F\big(f_0^1y,f_{j+1}^i(f_0^0,\ldots,f_0^0)\big)\leftarrow \quad F\big(y,f_j^i(x_1,\ldots,x_i)\big)\wedge\bigwedge_{z=1}^{i}F(y_z,x_z)\wedge$$

$$\bigwedge_{z=1}^{i}Sz(f_0^1y_z,w_z)\wedge Sz(y,w)\wedge\bigwedge_{z=1}^{i}\leqq(w,w_z)$$

$$:i\leqq k,\ j<m_i\Big\}\cup$$

$$\Big\{F\big(f_0^1y,f_0^{i+1}(f_0^0,\ldots,f_0^0)\big)\leftarrow \quad F\big(y,f_{m_i}^i(x_1,\ldots,x_i)\big)\wedge\bigwedge_{z=1}^{i}F(y_z,x_z)\wedge$$

$$\bigwedge_{z=1}^{i}Sz(f_0^1y_z,w_z)\wedge Sz(y,w)\wedge\bigwedge_{z=1}^{i}\leqq(w,w_z)$$

$$:i<k\Big\}\cup$$

$$\Big\{F(f_0^1y,f_0^0)\leftarrow \quad F\big(y,f_{m_k}^k(x_1,\ldots,x_k)\big)\wedge\bigwedge_{z=1}^{k}F(y_z,x_z)\wedge$$

$$\bigwedge_{z=1}^{k}Sz(f_0^1y_z,w_z)\wedge Sz(y,w)\wedge\bigwedge_{z=1}^{k}\leqq(w,w_z)\Big\}$$

$$\cup$$

$$\Big\{Sz(f_0^0,f_0^0)\leftarrow,$$

$$Sz(f_0^1y,f_0^1w)\leftarrow F\big(y,f_{m_k}^k(x_1,\ldots,x_k)\big)\wedge\bigwedge_{z=1}^{k}F(y_z,x_z)\wedge\bigwedge_{z=1}^{k}Sz(f_0^1y_z,w_z)\wedge Sz(y,w)\wedge$$

$$\bigwedge_{z=1}^{k}<(w,w_z)\Big\}$$

$$\cup$$

$$\{\neq(f_i^0,f_j^0)\leftarrow \qquad :i\neq j\}\cup$$

$$\neq\big(f_j^i(x_1,\ldots,x_i),f_{j'}^{i'}(y_1,\ldots,y_{i'})\big)\leftarrow \qquad :\langle i,j\neq\langle i',j'\rangle\}\cup$$

$$\neq\big(f_j^i(x_1,\ldots,x_i),f_j^i(y_1,\ldots,y_i)\big)\leftarrow\ \neq(x_p,y_p):i\leqq k,j\leqq m_i,0<p\leqq i\}\cup$$

$$\{NE(y,f_0^0)\leftarrow,$$

$$NE(y,f_0^1w)\leftarrow NE(y,w)\wedge\neq(x,v)\wedge F(y,x)\wedge F(w,v),$$

$$E(y)\leftarrow NE(y,y)\}\cup$$

$$\{\neg E(y)\leftarrow\ <(z,y)\wedge F(z,w)\wedge F(y,w),$$

$$N(x,f_0^0)\leftarrow,$$

$$N(x,f_0^1y)\leftarrow N(x,y)\wedge\neq(w,x)\wedge F(y,w),$$

$$M(x,y)\leftarrow N(x,y)\wedge F(y,x),$$

$$N_1(y, f_0^0) \leftarrow$$
$$N_1(y, f_0^1 z) \leftarrow N_1(y, z) \wedge \leqq (z, y),$$
$$M_1(y, f_0^1 z) \leftarrow N_1(y, z) \wedge \neg E(z),$$
$$S(x, w) \leftarrow M(x, z) \wedge M_1(z, y) \wedge F(y, w)\}.$$

c) *the projection function:*
$\varrho^{-1} \circ I_m^n \circ \varrho$ is Horn definable:

$$C_I \overset{d}{=} \{F_I(x_1, \ldots, x_n, x_m) \leftarrow \}.$$

Now for the following steps suppose that $C_h, C_g, C_{g_1}, \ldots, C_{g_n}$ define $\varrho^{-1} \circ h \circ \varrho$, $\varrho^{-1} \circ g \circ \varrho$, ... respectively.

d) *substitution:*
$f \overset{d}{=} \varrho^{-1} \circ Su(h, g_1, \ldots, g_n) \circ \varrho$ is Horn definable; where $Su(h, g_1, \ldots, g_n)$ is the function defined by substitution from $h, g_1, \ldots, g_n$.

$$C_f \overset{d}{=} \{F_f(x, y) \leftarrow F_h(y_1, \ldots, y_n, y) \wedge F_{g_1}(x, y_1) \wedge \ldots \wedge F_{g_n}(x, y_n)\} \cup$$
$$C_h \cup C_g \cup \ldots \cup C_{g_n}.$$

To prove that C_f really defines f, note that

$$\varrho^{-1} \circ Su(h, g_1, \ldots, g_n) \circ \varrho(\bar{x}) = \varrho^{-1}(h(g_1(\varrho(\bar{x})), \ldots, g_n(\varrho(\bar{x}))) =$$
$$= \varrho^{-1} \circ h \circ \varrho(\varrho^{-1} \circ g \circ \varrho(\bar{x}), \ldots, \varrho^{-1} \circ g_n \circ \varrho(\bar{x})).$$

(Similar remarks will be omitted in the following.)

e) *recursion:*
$f \overset{d}{=} \varrho^{-1} \circ R(g, h) \circ \varrho$ is Horn definable, where $R(g, h)$ is the function defined by recursion from g and h.

$$C_f \overset{d}{=} \{F_f(\bar{x}, f_0^0, y) \leftarrow F_g(\bar{x}, y),$$
$$F_f(\bar{x}, w, y) \leftarrow F_s(z, w) \wedge F_f(x, z, y_1) \wedge F_h(\bar{x}, z, y_1, y)\} \cup$$
$$C_g \cup C_s \cup C_h.$$

Remember that C_s defines the function $\varrho^{-1} \circ S \circ \varrho$, where S is the successor function on ω.

f) *the μ-operator:*
$f \overset{d}{=} \varrho^{-1} \circ Myg \circ \varrho$ is Horn definable, where Myg is the function defined by the μ-operator from g.

$$C_f \overset{d}{=} \{N(f_0^0) \leftarrow, N(w) \leftarrow S(z, w) \wedge N(z) \wedge F_g(\bar{x}, z, y) \wedge S(y_1, y),$$
$$F_f(\bar{x}, y) \leftarrow N(y) \wedge F_g(\bar{x}, y, f_0^0)\} \cup C_g \cup C_s.$$

Abstract

The adequacy of Horn clauses as a programming language is demonstrated by proving that any computable problem can be formulated in Horn logic without using auxiliary function symbols.

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

References

[1] WARREN, D. WARPLAN, A system for generating plans, *DCL Memo*, No. 76, 1974.
[2] BATTANI, G., H. MELONI, Interpreteur du language de programmation PROLOG, Université d'Aix Marseille, 1973.
]3] KOWALSKI, R., Predicate logic as programming language *DCL Memo*, No. 70, University of Edinburg, 1973, and Proc. IFIP Congr. 1974.
[4] KOWALSKI, R., Logic for problem solving. *DCL Memo*, No. 75, University of Edinburg, 1974.
[5] VAN EMDEN, M., First-order predicate logic as a high-level program language. School of AI, University of Edinburg, MIP—R—106, 1974.
[6] VAN EMDEN, M., R. KOWALSKI, The semantics of predicate logic as a programming language, *DCL Memo*, No. 73, University of Edinburg, 1974.
[7] VAN EMDEN, M., Recursion equations as predicate-logic programs, to be published.

(Received Jan. 13, 1978)

Tree transformations and the semantics of loop-free programs

By M. A. Arbib* and E. G. Manes**

In memory of László Kalmár

Alagić [1975] gave a category-theoretic treatment of natural state transformations which generalized the work of Thatcher [1970], and so, in particular, gave an elegantly general perspective on tree transformations. Arbib and Manes [1977] modified Alagić's approach to provide a somewhat more concrete category-theoretic approach to what they called *process transformations,* which they showed to embrace recursion theory, bottom-up tree transformations and linear systems. Section 1 of the present note specializes the theory of process transformations to show how pure bottom-up tree transformations may be expressed in category-theoretic form. Section 2 then shows how this formulation may provide insight into the semantics of loop-free programs. Later papers will consider the effect of loops. Necessary category-theoretic background may be found in Arbib and Manes [1975], especially Chapter 7 and Section 10.1.

1. Bottom-up tree transformations: A category-theoretic characterization

We first recall the 'machines in a category' approach to tree automata (i.e. Ω-algebras).

1. Definition. An *operator domain* Ω is a sequence $(\Omega_n | n \in \mathbf{N})$ of (possibly empty) disjoint sets. An *Ω-algebra* is a pair (Q, δ) where Ω is a set and $\delta = (\delta_n)$ is a sequence of maps $\delta_n \colon Q^n \times \Omega_n \to Q$. We write δ_ω for $\delta(-, \omega) \colon Q^n \to Q$ for $\omega \in \Omega_n$. Q is the *carrier* of the algebra.

Given Ω, we define a functor X_Ω: **Set** $\to$ **Set** by

$$QX_\Omega = \bigcup_{n \geqq 0} Q^n \times \Omega_n \tag{2}$$

while, for $h \colon Q \to Q'$

$$hX_\Omega(q_1, \ldots, q_n, \omega) = (hq_1, \ldots, hq_n, \omega). \tag{3}$$

We now observe that an X_Ω-dynamics in the sense of Arbib and Manes [1974] — i.e. a map $QX_\Omega \to Q$ — is just an Ω-algebra, and that an X_Ω-dynamorphism is just an *Ω-homomorphism*, since the equation $\delta' \cdot hX_\Omega = h \cdot \delta$ which characterizes a map $h\colon Q \to Q'$ as a dynamorphism $h\colon (Q, \delta) \to (Q', \delta')$ unpacks to

$$h\delta_\omega(q_1, \dots, q_n) = \delta'_\omega(hq_1, \dots, hq_n) \quad \text{for} \quad \omega \in \Omega_n, \quad (q_1, \dots, q_n) \in Q^n.$$

Moreover, X_Ω is a recursion process (which is the same as an input process in the sense of Arbib—Manes), which means that there exists an Ω-algebra $(AX_\Omega^@, A\mu_0)$ equipped with an inclusion of generators $A\eta\colon A \to AX_\Omega^@$ such that for any Ω-algebra (Q, δ) we may extend each map $\tau\colon A \to Q$ uniquely to a homomorphism $r\colon (AX_\Omega^@, A\mu_0) \to (Q, \delta)$. $AX_\Omega^@$ is the carrier of the well-known free Ω-algebra generated by A, and may be defined by the usual inductive definition (Birkhoff [1935]):

$$A \subset AX_\Omega^@$$

$$\text{If} \quad \omega \in \Omega_n,\ t_1, \dots, t_n \in AX_\Omega^@, \quad \text{then} \quad \omega t_1 \dots t_n \in AX_\Omega^@. \tag{4}$$

Thus the elements of $AX_\Omega^@$ may be regarded as finite rooted trees, with nodes of outdegree n labelled by elements of Ω_n, save that some leaves (nodes of outdegree 0) may be labelled by elements of A. We abbreviate $X_\Omega^@$ to T_Ω. We may define

$$\begin{aligned} &A\eta\colon A \to AT_\Omega, \quad a \mapsto a \\ &A\mu_0\colon AT_\Omega X_\Omega \to AT_\Omega\colon (t_1, \dots, t_n, \omega) \mapsto \omega t_1 \dots t_n. \end{aligned} \tag{5}$$

If (Q, δ) is any Ω-algebra and $\tau\colon A \to Q$ is any map

$$\begin{tikzcd} A \arrow[r, "A\eta"] \arrow[dr, "\tau"'] & AT_\Omega \arrow[d, "r"] & AT_\Omega X_\Omega \arrow[l, "A\mu_0"'] \arrow[d, "rX_\Omega"] \\ & Q & QX_\Omega \arrow[l, "\delta"] \end{tikzcd} \tag{6}$$

then the unique dynamorphic extension $r\colon AT_\Omega \to Q$ of τ is given by

$$\begin{aligned} r(a) &= \tau(a) \\ r(\omega t_1 \dots t_n) &= \delta_\omega(rt_1, \dots, rt_n). \end{aligned} \tag{7}$$

Note that this reduces to the dynamics $\delta\colon Q \times X_0 \to Q$ of a sequential machine if we take $\Omega_1 = X_0$ while $\Omega_n = \emptyset$ for $n \neq 1$.

Suppose that Ω and Σ are two operator domains. We consider 'bottom up' (i.e. working from the leaves to the root) transformations of trees in AT_Ω into trees in BT_Σ: (The following transformations are 'pure' in that no internal state is used in processing the trees. The more general definition is given in Arbib and Manes [1979].)

8. Definition. Given operator domains Ω and Σ, and sets A and B, a *bottom-up tree transformation* $(A, \Omega) \to (B, \Sigma)$ is given by a map $\alpha: A \to B$, together with a sequence $\beta = (\beta_n)$ of maps

$$\beta_n: \Omega_n \to \{1, \ldots, n\} T_\Sigma. \tag{9}$$

The *response* of (α, β) is $\gamma: AT_\Omega \to BT_\Sigma$ defined inductively by:

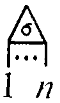
Basis step:

$$\gamma(a) = \alpha(a) \tag{10}$$

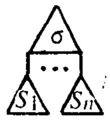
Induction step: To define

$$\gamma(\omega t_1 \ldots t_n), \quad \text{let} \quad \gamma(t_j) = s_j, \tag{11}$$

and let

$\beta(\omega) =$ 

Then

$\gamma(\omega t_1 \ldots t_n) =$ 

The following result in the style of the Yoneda Lemma (Mac Lane [1971]) allows us to view β as a natural transformation. (For an exposition of the concept of a natural transformation of functors, see Arbib and Manes [1975, Section 7.3].) This theorem is generalized in (Arbib and Manes [1977]).

12. Theorem. Let Ω be an operator domain, and let Y be any functor **Set**$\to$**Set**. Then there exists a canonical bijection

$$\frac{X_\Omega \xrightarrow{\beta} Y}{\Omega_n \xrightarrow{\beta_n} nY} \tag{13}$$

between natural transformations β and sequences (β_n) of functions. Mutually inverse passages are given by

$$\beta_n = \Omega_n \xrightarrow{k} nX_\Omega \xrightarrow{n\beta} nY \quad \text{where} \quad k(\omega) = (1, \ldots, n, \omega) \tag{14}$$

$$A\beta: AX_\Omega \to AY, \quad (a_1, \ldots, a_n, \omega) \mapsto (a_1, \ldots, a_n) Y \cdot \beta_n(\omega). \tag{15}$$

To explain the notation in (15), $(a_1, \ldots, a_n)$ is a function $g: n \to A$. Thus $(a_1, \ldots, a_n) Y$ is a function $gY: nY \to AY$.

Proof. To see that (15) describes a natural transformation, we must verify

$$\begin{tikzcd} AX_\Omega \arrow[r, "A\beta"] \arrow[d, "hX_\Omega"'] & AY \arrow[d, "hY"] \\ BX_\Omega \arrow[r, "B\beta"] & BY \end{tikzcd}$$

for arbitrary $h: A \to B$. But starting from $(g, \omega) \in A^n \times \Omega_n$, the upper path yields $hY \cdot gY(\beta_n(\omega))$ and the lower path yields $(hg) Y \cdot \beta_n(\omega)$ and these are equal since Y is a functor.

We now verify that (14) and (15) are inverse.
Now if $(\beta_n) \mapsto \beta \mapsto (\bar{\beta}_n)$, we have

$$\begin{aligned}\bar{\beta}_n(\omega) &= n\beta(1, \ldots, n, \omega)\\ &= n\beta(\mathrm{id}_n, \omega) \quad \text{for} \quad \mathrm{id}_n \in n^n\\ &= \mathrm{id}_n Y \cdot \beta_n(\omega) = \beta_n(\omega).\end{aligned}$$

Conversely, if $\beta \mapsto \beta_n \mapsto \bar{\beta}$, then for $g \in A^n$ we have the naturality square

$$\begin{tikzcd} nX_\Omega \arrow[r, "n\beta"] \arrow[d, "gX_\Omega"'] & nY \arrow[d, "gY"] \\ AX_\Omega \arrow[r, "A\beta"] & AY \end{tikzcd}$$

so that

$$\begin{aligned}(A\bar{\beta})(g, \omega) &= (gY)(\beta_n(\omega))\\ &= (gY)(n\beta(\mathrm{id}_n, \omega))\\ &= (A\beta)gX_\Omega(\mathrm{id}_n, \omega)\\ &= (A\beta)(g, \omega). \quad \square\end{aligned}$$

We thus conclude

16. Observation. A bottom-up tree transformation from Ω-trees to Σ-trees s equivalently given by a natural transformation

$$\beta\colon X_\Omega \to T_\Sigma$$

together with a map $\alpha\colon A \to B$. The *response* $\gamma\colon AT_\Omega \to BT_\Sigma$ is uniquely defined by the diagram

$$\begin{tikzcd} A \arrow[r, "A\eta^\Omega"] \arrow[d, "\alpha"'] & AT_\Omega \arrow[d, "\gamma"] & & AT_\Omega X_\Omega \arrow[ll, "A\mu_0^\Omega"'] \arrow[d, "\gamma X_\Omega"] \\ B \arrow[r, "B\eta^\Sigma"'] & BT_\Sigma & BT_\Sigma T_\Sigma \arrow[l, "B\mu^\Sigma"] & BT_\Sigma X_\Omega \arrow[l, "BT_\Sigma\beta"] \end{tikzcd} \tag{17}$$

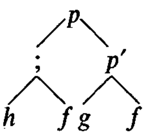
Proof. The left-hand square provides the basis step of the inductive definition of τ given in Definition (8), while the right-hand square expresses the way in which $\gamma(\omega t_1 \ldots t_n)$ depends on $\gamma(t_j)$ for $1 \leq j \leq n$. $\square$

2. Transforming loop-free flow diagrams

In this section, we capture the essential ideas of Reynolds' [1977] "Semantics of the domain of flow diagrams" by giving a succinct account of the relation between general flow diagrams and linear flow diagrams which provides the paradigm for the other relations discussed in that paper. We fix a set P of predicate symbols and a set F of function symbols. A general flow diagram may be represented by a Σ-tree where

$$\Sigma_0 = F, \quad \Sigma_1 = \emptyset, \quad \Sigma_2 = P \cup \{;\} \tag{18}$$

and we interpret the following element of $\emptyset T_\Sigma$



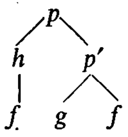
(19)

as "If the p-test yields true, execute h then f; whereas if the test yields false, carry out the p'-test, executing g if the outcome is true, f if the outcome is false."

A linear flow diagram is one in which we cannot compose arbitrary operations using ";", but instead apply one f at a time. They correspond to Ω-trees where

$$\Omega_0 = F\times\{0\}, \quad \Omega_1 = F\times\{1\}, \quad \Omega_2 = P \tag{20}$$

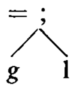
and (19) corresponds to the following element of $\emptyset T_\Omega$



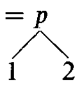
(21)

We now show that that transformation from linear flow diagrams (as represented by Ω-trees) to general flow diagrams (as represented by Σ-trees) is given by the tree transformation $\beta_n : \Omega_n \to \{1, \ldots, n\} T_\Sigma$ where

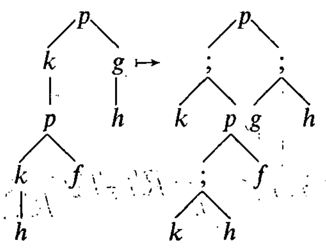
$$\beta_0(f, 0) = f$$

$\beta_1(g, 1) =$


(22)

$\beta_2(p) =$


The response $\emptyset T_\Omega \to \emptyset T_\Sigma$ does indeed transform (21) into (19), and the reader may see that it also yields the following typical transformation:



(23)

Now Reynolds provides for each direct (resp., continuation) semantics for general flow diagrams a corresponding semantics for linear flow diagrams. But each semantics for a general (respectively linear) flow diagram is nothing more nor less than a Σ- (respectively Ω-) algebra. Any particular choice of a transformation of semantics which "preserves meaning" with respect to a particular transformation of flow diagrams is subsumed in the following result (which works just as well when T_Σ and T_Ω are replaced by arbitrary algebraic theories T_1 and T_2, see Manes [1976, Section 3.2]):

24. Proposition. Let Ω and Σ be operator domains, and let $\xi\colon RX_\Sigma \to R$ be a given Σ-algebra. Further, let the family of maps

$$\beta_n\colon \Omega_n \to \{1, \ldots, n\}T_\Sigma$$

define a tree transformation. Then there exists an Ω-algebra $\delta\colon RX_\Omega \to R$ such that the result of running δ on any Ω-tree equals the result of running ξ on the transformed Σ-tree.

Proof. By (13), β_n is equivalent to a natural transformation

$$\beta\colon X_\Omega \to T_\Sigma$$

yielding, in particular, the map

$$R\beta\colon RX_\Omega \to RT_\Sigma. \tag{25}$$

Now we define the run map $\xi^@\colon RT_\Sigma \to R$ of (R, ξ) by the diagram (compare (6))

$$\begin{tikzcd}
R \arrow[r, "R\eta^\Sigma"] \arrow[dr, "\mathrm{id}_R"'] & RT_\Sigma \arrow[d, "\xi^@"] & RT_\Sigma X_\Sigma \arrow[l, "R\mu_0^\Sigma"'] \arrow[d, "\xi^@ X_\Sigma"] \\
& R & RX_\Sigma \arrow[l, "\xi"]
\end{tikzcd} \tag{26}$$

and we may then define an Ω-algebra (δ, R) by

$$\delta = RX_\Omega \xrightarrow{R\beta} RT_\Sigma \xrightarrow{\xi^@} R. \tag{27}$$

To show that δ has the claimed property, we must look at the response $\gamma\colon RT_\Omega \to RT_\Sigma$ of the tree transformation with $A = B = R$ and $\alpha = id_R$. Then (17) becomes:

$$\begin{tikzcd}
R \arrow[r, "R\eta^\Omega"] \arrow[dr, "R\eta^\Sigma"'] & RT_\Omega \arrow[d, "\gamma"] & & RT_\Omega X_\Omega \arrow[ll, "R\mu_0^\Omega"'] \arrow[d, "\gamma X_\Omega"] \\
& RT_\Sigma & RT_\Sigma T_\Sigma \arrow[l, "R\mu^\Sigma"] & RT_\Sigma X_\Omega \arrow[l, "RT_\Sigma\beta"]
\end{tikzcd} \tag{28}$$

We have to show that $\delta^@ = RT_\Omega \xrightarrow{\gamma} RT_\Sigma \xrightarrow{\xi^@} R$ to complete the proof of the proposition. But this is immediate from the following diagram:

$$\begin{tikzcd}
R \arrow[r, "R\eta^\Omega"] \arrow[ddr, "R\eta^\Sigma"'] \arrow[ddddr, bend right, "\mathrm{id}_R"'] & RT_\Omega \arrow[dd, "\gamma"] & & RT_\Omega X_\Omega \arrow[ll, "R\mu_0^\Omega"'] \arrow[dd, "\gamma X_\Omega"] \\
& \mathrm{I} & \mathrm{II} & \\
& RT_\Sigma \arrow[dd, "\xi^@"] & RT_\Sigma T_\Sigma \arrow[l, "R\mu^\Sigma"'] \arrow[dd, "\xi^@ T_\Sigma"] & RT_\Sigma X_\Omega \arrow[l, "RT_\Sigma\beta"'] \arrow[dd, "\xi^@ X_\Omega"] \\
\mathrm{III} & & \mathrm{IV} & \mathrm{V} \\
& R & RT_\Sigma \arrow[l, "\xi^@"'] & RX_\Omega \arrow[l, "R\beta"'] \arrow[ll, bend left, "\delta"] \\
& & \mathrm{VI} &
\end{tikzcd}$$

(29)

where I and II are just (28), III and IV extend (26), V is a naturality square for θ, and VI is the definition of δ. Thus $\xi^@ \cdot \gamma$ satisfies the diagram which defines $\delta^@$ uniquely. □

* COMPUTER AND INFORMATION SCIENCE
UNIVERSITY OF MASSACHUSETTS
AMHERST, MASSACHUSETTS 01003, USA

** MATHEMATICS DEPARTMENT
UNIVERSITY OF MASSACHUSETTS
AMHERST, MASSACHUSETTS 01003, USA

References

[1] ALAGIĆ, S., Natural state transformations, *J. Comput. System Sci.*, v. 10, 1975, pp. 266—307.
[2] ARBIB, M. A. and E. G. MANES, Machines in a category, an expository introduction, *SIAM Rev.*, v. 16, 1974, pp. 163—192.
[3] ARBIB, M. A. and E. G. MANES, *Arrows, structures, and functors: The categorical imperative*, Academic Press, New York, 1975.
[4] ARBIB, M. A. and E. G. MANES, Intertwined recursion, tree transformations and linear systems, *Information and Control*, 1979, In Press.
[5] BIRKHOFF, G., On the structure of abstract algebras, *Proc. Cambridge Philo. Soc.*, v. 31, 1935, pp. 433—454.
[6] MAC LANE, S., *Categories for the working mathematician*, Springer-Verlag, 1971.
[7] MANES, E. G., *Algebraic theories*, Springer-Verlag, 1976.
[8] REYNOLDS, J. C., Semantics of the domain of flow diagrams, *J. Assoc. Comput. Mach.*, v. 24, 1977, pp. 484—503.
[9] THATCHER, J. W., Generalized^2 sequential machine maps, *J. Comput. System Sci.*, v. 4, 1970, pp. 339—367.

(Received April 5, 1978)

Mixed computation in the class of recursive program schemata

By A. P. Ershov

To the memory of Professor László Kalmár

Let some class $\mathfrak{A}$ of algorithms be prescribed by a set $\mathcal{P}$ of programs P, a domain $\mathcal{X}$ of input data X, a domain $\mathcal{Y}$ of results Y and a computation $\mathbf{V}$ being a universal process which is defined for any P and X and is either infinite or resultless (yielding an *abort*) or yields some Y as a function of P and X: $Y=\mathbf{V}(P, X)$. *Mixed computation* [1] in $\mathfrak{A}$ is a universal process $\mathbf{M}$ which is defined for any P, X and a parameter μ (specifically characterizing the process). The process is either infinite or resultless or it generates some *residual program* $\mathbf{M}_G(P, M, \mu)$ and yields *partial results* $\mathbf{M}_C(P, X, \mu)$. A mixed computation is correct if for any P, X and μ the following functional identity holds

$$\mathbf{V}(P, X) = \mathbf{V}(\mathbf{M}_G(P, X, \mu),\ \mathbf{M}_C(P, X, \mu)).$$

It has been shown [2] that mixed computation and such related concepts as partial evaluation [3], computation over incomplete information [4], "progonka" [5] may be a basis for solution of many programming problems where efficiency has to be traded off with universality.

It is natural to seek a correct formalism of mixed computation for the most common abstract models of program. The correctness of mixed computation for ALGOL-like programs has been shown in [6]. In this note a correct procedure of mixed computation in the class of recursive program schemata is presented. This class reflects such properties of algorithmic languages as recursion and proceduring.

We shall introduce some notations. If $\mathcal{M}$ is a set of elements m then M^n is an n-tuple of elements from $\mathcal{M}$. The length of a tuple used as an argument of a functional symbol f is always equal to its arity $\varrho(f)$. $\tau[B]$ is a term τ constructed over a set B of basic symbols, $\tau(A)$ is a term τ for which its arguments (variables or constants) A are shown.

According to [7] a *recursive program schema* is specified as a system of equalities *(function declarations)*

$$f_i(X_i^{\varrho(f_i)}) = \tau_i[X_i, C, \{f_1, \dots, f_k\}, \Pi, \Phi] \quad (i = 1, \dots, k),$$

where f_i are *defined functions.* $\mathscr{X}$ and $\mathscr{C}$ are countable sets of variables x and constants c, Π and Φ are finite sets of predicate and functional symbols, respectively, of fixed arities.

Predicate terms π are used to define *conditional* terms $\{\pi|t_1|t_2\}$ where t_1 and t_2 are functional or conditional terms. Terms τ_i are arbitrary terms *(function bodies)* over specified sets of symbols.

Let an interpretation of the basic symbols (constants, functions and predicates) converting a schema into a recursive program be given. A system of functions $\varphi_1, \ldots, \varphi_k$ is called a *fixed point* of a recursive program if, having been combined with the system of basic functions Π and Φ, it makes (after substituting φ_i for each f_i) the function declarations identities.

We say that a function φ_1 *covers* a function φ_2 if the graph of φ_1 contains that of φ_2. Under natural assumptions on basic functions and their regions of definiteness each recursive program has a single so called *lowest fixed point* (LFP) covered by any other fixed point of the program [7].

Let T and C be tuples of terms and constants respectively. A *call* is a term in the form $f(T)$; a *bound call* is a term in the form $f(C)$; a *semi-bound* call is a term in the form $f(C^n, T^m)$ where $n+m=\varrho(f)$; a *transitively* bound call is a call having no variables.

Let one function declaration $f(X)=\tau$ in a program be treated as a *leading declaration* and C be a tuple of $\varrho(f)$ constants. A (sequential) *computation* V over a program P is a step-wise process of constructing a sequence of terms $\tau^0=f(C)$, $\tau^1, \tau^2, \ldots$ which either is developed infinitely or ends by an (resultless) *abort* or (sucessfully) by a constant which is taken as the value $\varphi(C)$ of the function $\varphi(X)$ computed over the given program for its leading declaration.

Each step of the construction of τ^{i+1} from τ^i consists of two parts.

1. *Rewriting.* In τ^i somehow a call $f_j(T)$ is chosen. This call is replaced by a term t. The latter is obtained from the function body τ_j of the declaration $f_j(X_j)=\tau_j$ by replacement of variables from X_j by corresponding components of the tuple T. Let τ' be the rewritten term.

2. *Simplification.* Inductively, all such subterms in τ' are evaluated which contain only constants and basic functions and predicates. The evaluated functional terms are replaced by their value, conditional terms are replaced by their *if-* or *else*-part depending on the value of the predicate. If the simplification yields either an abort or a constant c then the process in terminated yielding either the abort or c as a successful result. Otherwise, the simplified term is taken as τ^{i+1}.

Similarly, a *partial* computation is defined which allows τ^0 to be an arbitrary term with variables. Partial computation is terminated when the simplified term contains no available transitively bound calls.

A variety of computations is determined by the method of selection of subterms subjected to rewriting. In the general case a computation provides with a function covered by the LFP of a given recursive program. A computation which guarantees LFP is called *safe*. An example of safe computation is the execution of the "left outermost" call that corresponds to the "call by name". An unsafe computation is the execution of the "left innermost" call (call by value).

Let the first function declaration $f_1(X_1)=\tau_1$ of a recursive program P be leading and let a partition μ of variables X_1 ($X_1=X' \cup X$) and a semi-bound call

$f_1(B, X)$ be given. Let a computation **V** provide the leading declaration with a function $\varphi(X', X)$. A correct mixed computation **M** of the program P for the given partition μ and tuple of constans B is an arbitrary process of transformation of the program P into a program P_B with a leading declaration $f_0(X)=\tau_0$ such that the function $\varphi_B(X)$ provided by **V** for the program P_B satisfies the identity $\varphi(B, X)=$ $=\varphi_B(X)$.

We shall describe a transformation of P which we call an *execution* of the *semi-bound* call $f_1(B, X)$. Let us take a copy of the term τ_1 and replace in it all occurences of variables from X' by the corresponding constants from B with all subsequent simplifications; we will obtain a term τ_0 as a result. Then we take a new functional symbol f_0 of a defined function $f_0(X)$ and replace, in all terms $\tau_0, \tau_1, \ldots, \tau_k$, all semi-bound calls in the form $f_1(B, T)$ by the calls $f_0(T)$, thus obtaining the terms $\tau_0^*, \tau_1^*, \ldots, \tau_k^*$. Let us denote by P^* the program which is obtained from P by attaching to it the equality $f_0(X)=\tau_0^*$ as leading declaration and by replacing the bodies $\tau_1, \ldots, \tau_k$ by the terms $\tau_1^*, \ldots, \tau_k^*$.

Lemma 1. Let $\varphi_1(X', X); \varphi_2, \ldots, \varphi_k$ and $\psi_0, \psi_1, \ldots, \psi_k$ be LFP of the programs P and P^*, respectively. Then $\psi_i=\varphi_i$ $(i=1, \ldots, k)$ and $\psi_0(X)=\varphi_1(B, X)$.

The proof is based on Kleene's theorem on recursion [8]: it can be shown that subsequent approximations of P and P^* to their LFPs satisfy the lemma at each step.

Let us introduce a *reachability* relation over the defined functions $f_1, \ldots, f_k$ of a recursive program: f_j is reachable from f_i if the body of f_i contains calls for f_j. We will also consider the transitive closure of the reachability.

We shall formulate two obvious lemmas.

Lemma 2. Deleting from a program P the declaration of a function which is transitively unreachable from the function of the leading declaration preserves the 1st component of the LFP of P.

Lemma 3. Replacing in P a call $f(T)$ for the function with a declaration $f(X)=\tau(X)$ by the term $\tau(T)$ preserves the 1st component of the LFP of P.

Now we can describe a correct mixed computation with respect to some computation **V**.

Initial step. A semi-bound call $f_1(B, X)$ is given. It is declared to be the *start* of the first cyclic step.

Cyclic step (transformation of P into P'). Let a start $f(B, X)$ be given. The corresponding declaration in P is considered as the leading one. P is transformed into P^* with the leading declaration $f_0(X)=\tau_0$ according to the rules of execution of a semi-bound call. A partial computation **V** with τ_0 as the initial term and τ_0^* as the result (if any) is undertaken. P^* is then transformed into P' by replacing τ_0 by the term τ_0^*, in the declaration $f_0(X)=\tau_0$.

After each cyclic step we look at τ_0^* whether it contains a semi-bound call $f(C, T)$. If so then the term $f(C, Y)$, where Y are variables from the declaration of f which correspond the terms T, is taken as a start for the next cyclic step. Otherwise the mixed computation is terminated yielding the program after the last step with the leading declaration from the first cyclic step as the residual program. Afterwards, the residual program may be simplified according to lemmas 2 and 3.

Example A. (Power function x^n)

$$\text{pow}(x, n) = \{n = 0|1|\{n \text{ is even } |\text{pow}^2(x, n/2)|x\times\text{pow}(x, n-1)\}\}.$$

Let $\text{pow}(x, N) = N(x)$. The residual program for $\text{pow}(x, 5)$ before simplification

$5(x) = x \times 4(x);$

$4(x) = (2(x))^2;$

$2(x) = (1(x))^2;$

$1(x) = x \times 0(x);$

$0(x) = 1;$

$\text{pow}(x, n) = \{n = 0|1|\{n \text{ is even}|\text{pow}^2(x, n/2)|x \times \text{pow}(x, n-1)\}\}.$

The residual program after simplification:

$5(x) = x \times ((x \times 1)^2)^2.$

Let $\text{pow}(5, n) = \exp(n)$. The residual program $\text{pow}(5, n)$ after simplification:

$\exp(n) = \{n = 0|1|\{n \text{ is even}|\exp^2(n/2)|5 \times \exp(n-1)\}\}.$

Example B. (Akkerman function)

$A(x, y) = \{x = 0|y+1|\{y = 0|A(x-1, 1)|A(x-1, A(x, y-1))\}\}.$

Let $A(3, y) = \exp(y)$; $A(2, y) = \text{mult}(y)$; $A(1, y) = \text{add}(y)$, $A(m, n) = amn$.

The residual program for $A(3, y)$ after simplification:

$\exp(y) = \{y = 0|a21|\text{mult}(\exp(y-1))\};$

$\text{mult}(y) = \{y = 0|a11|\text{add}(\text{mult}(y-1))\};$

$\text{add}(y) = \{y = 0|a01|\text{add}(y-1)+1\}.$

Let $A(x, N) = aN(x)$. The residual program for $A(x, 3)$ before simplification:

$a3(x) = \{x = 0|4|A(x-1, a2(x))\};$

$a2(x) = \{x = 0|3|A(x-1, a1(x))\};$

$a1(x) = \{x = 0|2|A(x-1, a0(x))\};$

$a0(x) = \{x = 0|1|a1(x-1)\};$

$A(x, y) = \{x = 0|y+1|\{y = 0|a1(x-1)|A(x-1, A(x, y-1))\}\}.$

Notice, that elimination of non-recursive declarations can be made in different ways due to the mutual recursion of $a0$ and $a1$. Eliminating $a0$ and $a2$ we obtain (exploiting the logical dependencies):

$a3(x) = \{x = 0|4|A(x-1, A(x-1, a1(x)))\};$

$a1(x) = \{x = 0|2|A(x-1, a1(x-1))\};$

$A(x, y) = \{x = 0|y+1|\{y = 0|a1(x-1)|A(x-1, A(x, y-1))\}\}.$

COMPUTING CENTRE
SIBERIAN BRANCH OF THE USSR AC. SCI.
NOVOSIBIRSK 630090, USSR

References

[1] Ershov, A. P., Об одном теоретическом принципе системного программирования *Dokl. Akad. Nauk SSSR*, v. 223, No. 2, 1977, pp. 272—275.

[2] Ершов, А. П., О сущности трансляции, *Программирование*, No. 5, 1977, pp. 21—39.

3] Beckman, L., A. Haraldson, Ö. Oskarsson, E. Sandewall., A partial evaluator, and its use as a programming tool, *Artificial Intelligence*, v. 7, No. 4, 1976, pp. 319—357.

[4] Бабич, Г. Х., Л. Ф. Штернберг, Т. И. Юганова, Алгоритмический язык инкол для выполнения вычислений с неполной информацией, *Программирование*, No. 4, 1976, pp. 24—32.

[5] Турчин, В. Ф., Эквивалентные преобразования программ на Рефале, Автоматизированная система управления строительством, *Труды ЦНИПИАСС*, Moscow, No. 6, 1974, p. 36.

6] Ershov, A. P. and V. E. Itkin, Correctness of mixed computation in Algol-like programs, *Lecture Notes in Computer Science*, v. 53, 1977, pp. 59—77.

7] Manna, Z., S. Ness, J. Vuillemin, Inductive methods for proving properties of programs, *Comm. ACM*, v. 16, No. 8, 1973, pp. 491—502.

[8] Kleene, S. C., *Introduction to metamathematics*, Amsterdam—Groningen, 1952.

(Received August 1, 1978)

Certain operations with the sets of discrete states

By M. A. Gavrilov

In memory of László Kalmár

Discrete devices are nowadays widely used in various fields. Since the contemporary discrete devices are very complex, multipurpose and high-dimensional, considerable changes in conventional design techniques which rest upon the so-called "finite automaton" model [1] are necessary.

The basic disadvantage of the existing techniques for the description of control discrete devices, viz., flow tables (for sequential machines) and state tables (for combinational automata) is that each input, internal and output state should be dealt with separately, which limits significantly the dimensionality of the problems.

A way to increase the dimensionality is to use functions which are characteristic of sets of states with some special properties such as having the same distance between states, the same value of certain variables, etc.

Some operations with characteristic functions of the sets of states are described below. Development of these operations was necessary for the design of computer-aided logical design of discrete devices.

1. Proximity of functions

Let two Boolean functions F_i and F_j be given as their sets of permit (one meaning) M^1 and forbid (zero meaning) M^0* states $M_i = M_i^1 \cap M_i^0$; $M_j = M_j^1 \cap M_j^0$ characterized by the functions F_i^1, F_i^0, F_j^1, F_j^0.**

Let us distinguish the following sets of states: M_{ij}^{s1}, the subset of permit states identical for both M_i and M_j; M_{ij}^{s0}, the subset of forbid states identical for both

* A permit (forbid) state is the state in which the function is equal to one (zero). Besides, there are "don't care" states ($M^{\sim}$) which are indifferent to the value of the function (it may equal either 1 or 0).

Sets of states: M^1, M^0 and $M^{\sim}$ are nonintersect in pairs and $M^1 \cup M^0 \cup M^{\sim}$ is equal to the set of all states, i.e., its power is 2^n, where n is the number of varibles of the functions F_i and F_j.

** Statement "function $F(A)$ characterized sets of states M^*" means that:

$$F(A) = \begin{cases} 1 & \text{if } A \in M^* \\ 0 & \text{if } A \notin M^* \end{cases}$$

M_i and M_j; $M_i^{t_1}$, the subset of permit states only for M_i not contained in M_j; $M_i^{t_0}$, the subset of forbid states only for M_i not contained in M_j, $M_j^{t_1}$, the subset of permit states only for M_j not contained in M_i; $M_j^{t_0}$, the subset of forbid states only for M_j not contained in M_i; $M_i^{r_1}$, the subset of permit states in M_i contained

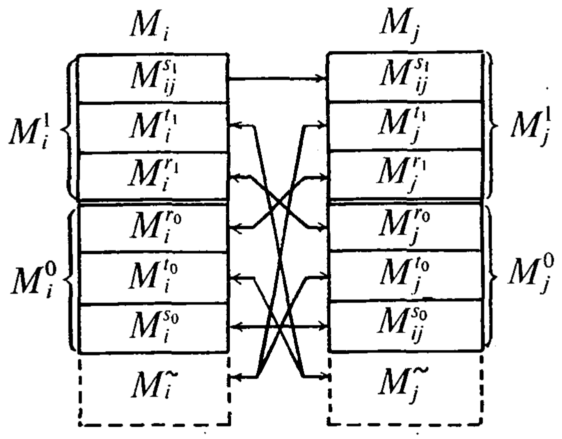


Fig. 1

in the forbid states set in M_j $(M_i^{r_1}=M_j^{r_0})$; $M_i^{r_0}$, the subset of forbid states in M_i contained in the permit states set in M_j $(M_i^{r_0}=M_j^{r_1})$ that is (fig. 1):

$$M_{ij}^{s_1} = M_i^1 \cap M_j^1;\ M_{ij}^{s_0} = M_i^0 \cap M_j^0;$$

$$M_i^{t_1} = M_i^1 \cap M_j^{\sim};\ M_i^{t_0} = M_i^0 \cap M_j^{\sim};\ M_j^{t_1} = M_j^1 \cap M_i^{\sim};\ M_j^{t_0} = M_j^0 \cap M_i^{\sim};$$

$$M_i^{r_1} = M_j^{r_0} = M_i^1 \cap M_j^0;\ M_i^{r_0} = M_j^{r_1} = M_i^0 \cap M_j^1.$$

If the functions F_i and F_j are given by the sets of their permit and forbid states then the sets of states of classes: s, t and r are characterized by the functions:

$$\begin{aligned} F_{ij}^{s_1} &= F_i^1 F_j^1;\ F_{ij}^{s_0} = F_i^0 F_j^0; \\ F_i^{t_1} &= F_i^1 F_j^{\sim} = F_i^1 \bar{F}_j^1 \bar{F}_j^0;\ F_i^{t_0} = F_i^0 F_j^{\sim} = F_i^0 \bar{F}_j^1 \bar{F}_j^0; \\ F_j^{t_1} &= F_j^1 F_i^{\sim} = F_j^1 \bar{F}_i^1 \bar{F}_i^0;\ F_j^{t_0} = F_j^0 F_i^{\sim} = F_j^0 \bar{F}_i^1 \bar{F}_i^0; \\ F_i^{r_1} &= F_j^{r_0} = F_i^1 F_j^0;\ F_i^{r_0} = F_j^{r_1} = F_i^0 F_j^1. \end{aligned} \tag{1}$$

Let us present the sets of states F^1 and F^0 as the join of the above subsets of states. The function will then be represented as (fig. 1):

$$\begin{aligned} M_i &= [M_i^1, M_i^0] = [(M_{ij}^{s_1} \cup M_i^{r_1} \cup M_i^{t_1}),\ (M_{ij}^{s_0} \cup M_i^{r_0} \cup M_i^{t_0})] \\ M_j &= [M_j^1, M_j^0] = [(M_{ij}^{s_1} \cup M_j^{r_1} \cup M_j^{t_1}),\ (M_{ij}^{s_0} \cup M_j^{r_0} \cup M_i^{t_0})]. \end{aligned} \tag{2}$$

The proximity of the functions is measured as the power of the subsets of states M^r. If $M_i^{r_1}, M_i^{r_0}, M_j^{r_1}, M_j^{r_0}$, are empty $M_{ij}^{s_1} = M_{ij}^{s_0}$ are empty the functions F_i and F_j after introduction of the additional don't care states may be realized by the same structure (fig. 2) but in the second case the output of one of the functions was taken from an additional invertor (fig. 3). Assume the proximity of

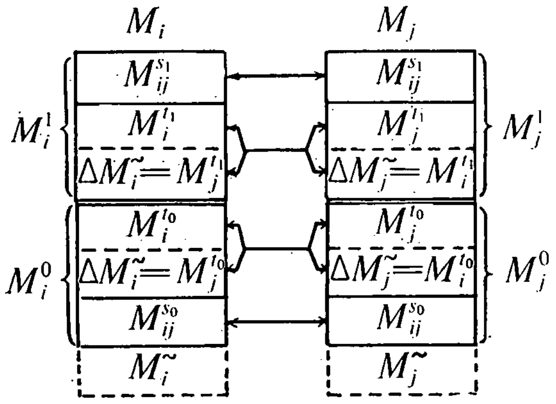


Fig. 2

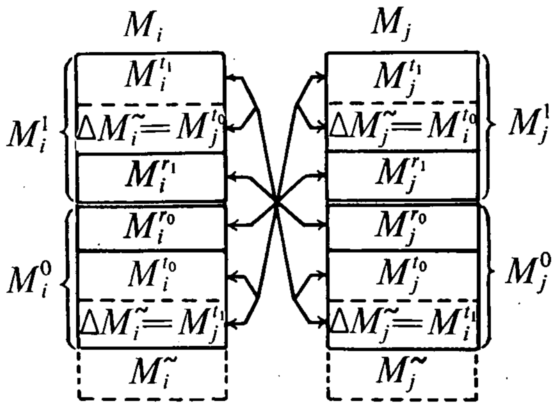


Fig. 3

the functions F_i and F_j is absolute (the distance is zero) with the corresponding functions completely connected in the first case, and maximal, with the corresponding functions inverse-completely connected in the second case.

The concept of the proximity of functions made use of in defining optimal or near-optimal architecture of realizing functions in multioutput structures. The design technique for such a realization builds the so-called "connectivity nodes" of the structure, viz., a set of functions "completely" or "inverse-completely" connected.

For functions which do not enter the connectivity nodes the distance to one of these nodes is to be found and the question answered whether the realization of these functions is connected with a connective node or a separate one.

To define these structures the operations of union intersection and complementation of subsets of states are used. If one has two sets of states M_i and M_j written in the form (2), one may write for the operations of union, intersection and complementation:

$$M_i \cup M_j = [(M_i^1 \cup M_j^1),\ (M_i^0 \cap M_j^0)] = [((M_{ij}^{s_1} \cup M_i^{t_1} \cup M_i^{r_1}) \cup$$
$$\cup (M_{ij}^{s_1} \cup M_j^{t_1} \cup M_j^{r_1})),\ ((M_{ij}^{s_0} \cup M_i^{t_0} \cup M_i^{r_0}) \cap (M_{ij}^{s_0} \cup M_j^{t_0} \cup M_j^{r_0}))].$$

The intersection of subsets of states $M_i^{r_0}$ and $M_j^{r_0}$ is empty subsets $M_i^{t_0}$ contains in subsets $M_j^{\sim}$ and subsets $M_j^{t_0}$ contains in subsets $M_i^{\sim}$. Therefore we shall have:

$$M_i \cup M_j = [(M_i^1 \cup M_j^1),\ (M_{ij}^{s_0} \cup M_i^{t_0} \cup M_j^{t_0})] \tag{3a}$$

Similarly,

$$M_i \cap M_j = [(M_i^1 \cap M_j^1), (M_i^0 \cup M_j^0)] = [(M_{ij}^{s_1} \cup M_i^{t_1} \cup M_j^{r_1}), (M_i^0 \cup M_j^0)] \tag{3b}$$

$$\overline{M}_i = [\overline{((M_i^1), (M_i^0))}] = [(M_i^0), (M_i^1)] \tag{3c}$$

2. Determination of the power of the state sets

In the above technique (as well as in determining some other criteria for the realization of these functions) the power of some subsets of states is to be found. The characteristic functions of these subsets can be described in an arbitrary form.

For this purpose [2] offers techniques for the transformation of an arbitrary Boolean expression into some "canonical" form enabling the computations of powers of various state subsets as a sum and product of the powers of the state subsets which correspond to separate parts of the function analyzed, thus significantly simplifying the computations. The use of the analytical form of the functions permits one to take full account of the information contained in the state table which corresponds to the analyzed function with no need to construct the table itself.

Let us enumerate the parentheses denoting by 1 the outer parentheses of the parenthetic expression of the Boolean function and increasing the index with the rank of the parenthesis. The subfunction in the i-th parenthesis will be referred to as the i-th disjunctive or conjunctive term depending on the outer logical operation of this subfunction (i.e., depending on the sign of the $(i+1)$ st terms contained in the expression). Inversion over the expressions will be denoted by square parenthesis and similarly enumerated.

A canonical parenthetic form which may be used to find the number of states is the form where any pair of terms included into a disjunctive term is orthogonal and all the terms of a conjunctive term should contain no coinciding variables.

The transfer to the canonical parenthetic form is done by means of the decomposition of a given parenthetic expression by variables using Shannon's rule. It is obvious that for the disjunctive term i of the canonical form the number of states equals the sum of the numbers of states of the $(i+1)$ conjunctive terms contained in this disjunctive term. The number of states of the conjunctive term will be $\alpha_i = 2^{n-k}\beta_1\beta_2\ldots\beta_m$ where n is the total number of variables, k is the number of variables contained in the conjunctive term, $\beta_1, \beta_2, \ldots, \beta_m$ is the sum of the numbers of states of the disjunctive terms contained in the conjunctive term. A term with square parenthesis (inversion) has the number of states defined as

$$\beta_j = 2^r - \beta_j^*$$

where r is the total number of variables contained in this inversion term and β_j^* is the number of states of this term.

This follows from the fact that the power of the sets of states, characterized by the inversion function is equal to addition up to 2^r (r — is the number of the variables of this function) from the power of the sets of states, characterized by function, wich is under the symbol of inversion.

Let the function

$$F = [x_i x_j \vee \bar{x}_j x_k x_n].$$

be given.

The number of states, characterised by the function, which is inside of square parenthesis (under symbol of inversion) is: $\beta^* = 6$. The number of variables of this function is: $r = 4$. Therefore the number of states, characterized by the given function is: $N_F = 2^4 - 6 = 10$.

Let us have a certain function specified by its permit (F^1) and don't care ($F^{\sim}$) states

$$F^1 = x_5 \vee \bar{x}_1 x_5 x_6 \vee x_1(\bar{x}_5 x_6 \vee x_2 \bar{x}_5 \bar{x}_6 x_{10} x_{11}) \vee x_2 x_1(\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_7 \vee \bar{x}_8 \vee \bar{x}_9),$$
$$F^{\sim} = x_3 x_4 \vee \bar{x}_8 \bar{x}_9.$$

Obviously the functions, characterized by the sets of permit and forbid states without the don't care ones ($F^{1\sim}$ and $F^{0\sim}$) will be described as

$$F^{1\sim} = F^1 \overline{F^{\sim}} = \big(x_5 \vee \bar{x}_1 x_5 x_6 \vee x_1(\bar{x}_5 x_6 \vee x_2 \bar{x}_5 \bar{x}_6 x_{10} x_{11}) \vee$$
$$\vee x_2 x_1(\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_7 \vee \bar{x}_8 \vee \bar{x}_9)\big)[x_3 x_4 \vee \bar{x}_8 \bar{x}_9],$$
$$F^{0\sim} = \bar{F}^1 \bar{F}^{\sim} = [x_5 \vee \bar{x}_1 x_6 x_6 \vee x_1(\bar{x}_5 \bar{x}_6 \vee x_2 \bar{x}_5 x_6 x_{10} x_{11}) \vee$$
$$\vee x_2 x_1(\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_7 \vee \bar{x}_8 \vee \bar{x}_9)][x_3 x_4 \vee \bar{x}_8 \bar{x}_9].$$

Transform these expressions to canonical form using Shannon's rule in order variable: x_1, x_2, x_3, x_4.* Denoting the upper index of parenthesis by the number of states in the form $2^{n-k}\beta_1\beta_2\ldots\beta_m$ and the lower index by rank of parenthesis, we shall have:

$$F^1 = {}_1\big({}_2({}^{2^8}x_1 x_2 x_3 \bar{x}_4\, {}_3[{}^{2^4}\bar{x}_8 \bar{x}_9]_3^{2^2\cdot 3})_2 \vee {}_2({}^{2^8}x_1 x_2 \bar{x}_3\, {}_3[{}^{2^5}\bar{x}_8 \bar{x}_9]_3)_2^{2^3\cdot 3} \vee {}_2\big({}^{2^8}x_1 \bar{x}_2\, {}_3({}^{2^6}x_5 \vee$$
$$\vee \bar{x}_5 x_6)_3^{2^4\cdot 3}\, {}_3(\bar{x}_3 \vee x_3 \bar{x}_4)_3^{2^2\cdot 9}\, {}_3[\bar{x}_8 \bar{x}_9]_3\big)_2^{2^0\cdot 27} \vee {}_2\big({}^{2^8}\bar{x}_1\, {}_3({}^{2^7}x_5 \vee \bar{x}_5 x_6)_3^{2^5\cdot 3}\, {}_3(\bar{x}_3 \vee$$
$$\vee x_3 \bar{x}_4)_3^{2^3\cdot 9}\, {}_3[\bar{x}_8 \bar{x}_9]_3\big)_2^{2^1\cdot 2^8}\big)_1$$
$$F^0 = {}_1\big({}_2^{2^8}(x_1 \bar{x}_2\, {}_3[{}^{2^6}x_5 \vee \bar{x}_5 x_6]_3^{2^4\cdot 1}\, {}_3(\bar{x}_3 \vee x_3 \bar{x}_4)_3^{2^2\cdot 3}\, {}_3[\bar{x}_8 \bar{x}_9]_3)_2^{2^0\cdot 9} \vee {}_2\big({}^{2^8}\bar{x}_1\, {}_3[{}^{2^7}x_5 \vee$$
$$\vee \bar{x}_5 x_6]_3^{2^5\cdot 1}\, {}_3(\bar{x}_3 \vee x_3 \bar{x}_4)_3^{2^3\cdot 3}\, {}_3[\bar{x}_8 \bar{x}_9]_3\big)_2^{2^1\cdot 9}\big)_1.$$

* For determining the order of variables, which give the expression, approaching the smallest amount of letters, it is useful to apply the heuristic criterion (5) or (6) (see page 7 and 8), as statistical experiments show.

Thus the powers of the sets of permit and forbid states for the given function will be

$$N^1 = 2^2 \cdot 3 + 2^3 \cdot 3 + 2^0 \cdot 27 + 2^1 \cdot 27 = 117$$

$$N^0 = 2^0 \cdot 9 + 2^1 \cdot 9 = 27.$$

3. Decomposition of Boolean functions

Realization of a given Boolean function in given elements is essentially a problem of decomposing this function into subfunctions in accord with the logical properties of the element. Obtaining the accurate solution for a problem of minimizing a Boolean function, or transformation to the form with the smallest number of operations and letters is a complex problem of combinatorial search [3, 4]. With the number of input variables as high as 20 or 30 the problem becomes hardly solvable even on computers. Therefore presently minimization of Boolean functions is achieved by means of heuristic methods with local optimization which we call the "directional search".

One of the first attempts to eliminate combinatorial search was introduced in [5] and widely used afterwards. This was the procedure of finding additional letters of the terms which describe the function in a contradictory way (the so-called "insufficient minterms"). Further in [6] a method of directional search was suggested for the case when a Boolean function was given by its table of states. The method contained criteria for selecting the so-called "inessential" variables* and finding minimal terms of the kernel as well as the minimal set of insufficient minterms.**

The fact that the function should be specified by its table of states significantly limits, however, the dimensionality of such problems. Ref. [7] suggested a technique in which minimization procedure rests upon the record of the given function and all its intermediate forms obtained in the course of minimization in an arbitrary analytical form thus considerably increasing the dimensions of the problems.

A more general technique was developed afterwards for realization of a function or a system of functions using "arbitrary" elements, or those whose logical properties are described by arbitrary Boolean functions [8].

The first stage of this technique implies elimination of the so-called "inessential" variables i.e. such whose elimination from F^1 and F^0 does not change the values of the function.

To determine inessential variables, a notion of Boolean "derivative" is used, introduced in Ref. [9]. The derivative of the given function with respect to an inessential variable is equal to zero.

$$\frac{dF}{dx_k} = F^1_{(x_k=1)} F^0_{(x_k=0)} \vee F^1_{(x_k=0)} F^0_{(x_k=1)} \tag{4}$$

* An inessential variable is a variable for which no pair of permit and forbid states exist differing only by the value of this variable. Elimination of this variable does not change the value of the functions. If a pair of permit and forbid states differs by the value of one variable, the values of this variable in these states are called obligatory letters.

** Minterm of the kernel is the conjunction of obligatory letters which describe only a subset of permit states or only a subset of forbid ones. Such terms should be included into all d-n. f. versions of a given function.

To obtain optimal realization* the order of elimination of inessential variables is important. A heuristic criterion is used for this purpose which estimates the proximity between the variable and the constant

$$R_k = n^1_{1,k} n^0_{0,k} + n^1_{0,k} n^0_{1,k} \tag{5}$$

where $n^1_{1,k}$ and $n^1_{0,k}$ are the number of permit states in the function in which the variable x_k takes on the values of 1 and 0, respectively, and $n^0_{1,k}$ and $n^0_{0,k}$ is the same for borbid states.

This criterion gives exact results in utmost cases, when the variable x_k is constant or a given function equal to the letter x_k or $\bar{x}_k$.

In the first case $n^1_{0,k} = n^0_{0,k} = 0$ or $n^1_{1,k} = n^0_{1,k} = 0$ and therefore $R = 0$. In the second case $n^1_{0,k} = n^0_{1,k} = 0$ or $n^1_{1,k} = n^0_{0,k} = 0$. It is possible to show that in these cases $R = \max$.

First an inessential variable is eliminated for which we have the least value of the criterion R. After the variable is eliminated from the function F, the values of R are recomputed and the next variable is eliminated until all the variables left are essential.

Let us assign as the inputs $y_1, y_2, \ldots, y_q$ of the output element φ a certain set of input variables $x_i, x_j, \ldots, x_q$. At the output of the element we shall have then the functions h and g.**

Then it is clear that if

$$F^1 \bar{h} = 0 \quad F^0 \bar{g} = 0$$

the function can be realized by a single element with a given assignment of variables as inputs of this element. If these expressions are not equal zero, the realization of the function will be contradictory, i.e., for some states from M^1 "0" will appear at the output of φ the element, and for some states, from M^0, "1".

Two problems arise here:

a) find a set of variables assigned as the inputs of the output element such that the functions h and g be as proximate as possible to the functions F^1 and F^0, that provides optimization of the entire structure, and

b) design the "additional" functions with the minimal necessary number of states assigned as inputs of the output element for elimination of contradictions.

The first problem is solved by the calculation of the value of the heuristic criterion for every variable x_k

$$b_k = \frac{n^1_{1,k}}{N^1} - \frac{n^1_{0,k}}{N^0} \tag{6}$$

where: $n^1_{1,k}$ and $n^1_{0,k}$ — have the same sense, as in criterion (5); N^1 is the power of the set M^1 and N^0 is the power of the set M^0. If b_k is positive then x_k is without

* By an optimal realisation we understand the obtaining of a function, nearing to such one, which has a minimal number of variables.

** The functions h and g specify the states which in the function realized by the element are permit and forbid states, respectively.

the sign of inversion. If b_k is negativ then x_k is with the sign of inversion. The variable x_k is selected with the maximal value of b_k.

The second problem is solved by determination of the sets of permit and forbid states of the so-called "additional" function, i.e., such function by the replacement of which variable x_k, will remove or decrease contradiction in realization of the given function.

Design of these functions for an arbitrary element is a rather bard task achieving which illustrate well the problem of isolating, from the sets of states of the given function, certain subsets with given properties, which was mentioned at the beginning of this paper.

Let us examine in a more detailed manner what states have to be permit and forbid for this function.

Let us introduce notion of "partial" derivative and "ranks" of partial derivative variables.

The partial derivatives of first rank for the variable x_k are:

$$\partial^1(F)_{x_k} = F^1_{(x_k=1)}F^0_{(x_k=0)} \tag{7a}$$

and

$$\partial^1(F)_{\bar{x}_k} = F^1_{(x_k=0)}F^0_{(x_k=1)} \tag{7b}$$

Expression (7a) characterizes the set of permit states in which variable $x_k=1$ and x_k is essential. Accordingly in the set of forbid states $x_k=0$ and x_k is also essential.

Expression (7b) characterizes the set of permit states in which the variable $x_k=0$ and x_k is essential. Accordingly in the set of forbid states $x_k=1$ and x_k is essential.

In these cases, for every state from M^1 and M^0 there will be found accordingly exactly one state in M^0 and M^1 which differs by the significance of the variable x_k from the given state.

Let us consider that in these cases these states are in distance "one".

Let us understand as derivatives of rank of "j" ($\partial^j(F)_{x_k}$ and $\partial^j(F)_{\bar{x}_k}$ functions which characterise the states from F^1 or F^0 having in ratio to given state j variables (including x_k), which have opposite significance. Let us consider, that in these cases these states are in distance "j".

For the analysis of states included in additional functions the expressions

$$S(F^1) = \bigvee_{j=2}^{n} \partial^j(E)_{x_k} \tag{8a}$$

and

$$S(F^0) = \bigvee_{j=2}^{n} \partial^j(F)_{\bar{x}_k} \tag{8b}$$

will be useful.

These expressions characterise disjunctions of states for the partial derivatives of all ranks, without the first.

To obtain the expressions $S(F^1)$ and $S(F^0)$ one should consider those for each rank of partial derivatives and then join them up.

Let us consider a technique of finding second rank partial derivatives. First find the functions $D^1(F^1)$ and $D^1(F^0)$ describing the "remaining" states in M^1 and M^0 after elimination of the states included in the first rank partial derivatives.

$$D^1(F^1) = F^1\overline{\tau^1(\varphi^1)}$$

$$D^1(F^0) = F^0\overline{\tau^0(\varphi^0)}$$

where

$$\tau^1(\varphi^1) = \Sigma(y_i\partial^1(\varphi^1)_{y_i}) \vee \Sigma(\bar{y}_i\partial(\varphi^1)_{\bar{y}_i})$$

and

$$\tau^1(\varphi^0) = \Sigma(\bar{y}_i\partial^1(\varphi^1)_{y_i}) \vee \Sigma(y_i\partial(\varphi^1)_{\bar{y}_i}).$$

The second rank partial derivatives are those of the first rank for $D^1(F^1)$ and $D^1(F^0)$ over $y_i, \bar{y}_i$ with respect to $\tau^1(\varphi^1)$ and $\tau^1(\varphi^0)$.

Higher rank partial derivatives are determined in a similar way.

Let functions F^1 and F^0 which characterise permit and forbid states M^1 and M^0 for some function F be given. Let an element φ be also given, having q inputs: $y_1, y_2, \ldots, y_q$. Permit and forbid states of this element are characterized by functions: $\varphi^1 = r(y_1, y_2, \ldots, y_q)$ and $\varphi^0 = s(y_1, y_2, \ldots, y_q)$. In addition, let the set of variables: $x_i, x_j, \ldots, x_k$ be determined as assigned by inputs of element φ, which result realisation of functions: $h(x_i, x_j, \ldots, x_k)$ and $g(x_i, x_j, \ldots, x_k)$ on the output of this element, accordingly with permit and forbid of given function F, but realize it contradictory.

In [10] the following formulas are given characterising permit ($f^1_{y_i}$) and forbid ($f^0_{y_i}$) states of additional function on input y_i of element

$$f^1_{y_i} = F^1(\partial^1(h)_{y_i} \vee S(g)_{\bar{y}_i}) \vee F^0(\partial^1(h)_{\bar{y}_i} \vee S(h)_{\bar{y}_i}) \tag{9a}$$

$$f^0_{y_i} = F^1(\partial^1(h_{\bar{y}_i}) \vee S(g)_{y_i}) \vee F^0(\partial^1(h)_{y_i} \vee S(h)_{y_i}). \tag{9b}$$

Let us show, that these formulas reflect category of states including in additional function correct and completely. To the set of permit states belong follows:

a) The states in which $x_k = 1$ or $x_k = 0$ and the given function is realized correctly at the output of the element, and the change of the value x_k changes the output value which becomes contradictory. It is clear that these values should be preserved in the additional function which provide for the replacement of the variable x_k.

b) The states in which x_k also is either "1" or "0" but the function F is realised at the output of the element contradictory and the change of the value x_k leads to elimination of the contradiction. Here as in the previous case the letter x_k is an obligatory letter and in order to eliminate the contradiction the state should be replaced with the one from the opposite set of states.

Functions characterising states of categories of a) and b) will be expressed therefore in the following form

$$\Delta^1(f^1_{y_i}) = F^1\partial^1(h)_{y_i} \vee F^0\partial^1(h)_{\bar{y}_i}.$$

If the states of the element φ, leading to contradictory realisation, belong only to categories a) and b), then the function $\Delta^1(f^1_{y_i})$ completely eliminates the contradictions.

c) If the state under consideration differs from those of the opposite set of the table by values of several variables, the change of the values x_k via additional function is still helpful, since it decreases the "distance" between the given state and the one which correctly realises the given function thus simplifying the realization of additional functions at the other inputs of the element.

The function, characterising these states, will be expressed in following form

$$\Delta''(f^1_{y_i}) = F^1 S(g)_{\bar{y}_i} \vee F^0 S(h)_{\bar{y}_i}.$$

a) The states, in which correct realisation of permit states of a given function F is provided write help of other variables and therefore the change of the value of given variable don't change the significance of the output of the element, belong to don't care states of additional function.

The disjunction: $\Delta'(f^1_{y_i}) \vee \Delta''(f^1_{y_i})$ gives formula (9a). The correctness and completeness of formula (9b) prove analogous.

Successive application of formulas (9a) and (9b) for all inputs of element φ and for received additional functions give the convergent process of elimination of contradictory in the realization of the given function F.

4. Algebraic model of a discrete device

A number of problems in the analysis of discrete devices (revealing statistical and dynamic races, reliability analysis, determination of check and diagnosis tests, etc.) are very difficult because of the lack of adequate models which would describe in a compact way the internal structure of the device as well as its operating algorithm.

The model without this defect [10] uses the fact that introduction of each internal variable (a function of the same input variables) doubles the number of states of the function and exactly one half of them should belong to the states of $M^{\sim}$. Indeed, if we have some element for the function

$$\varphi_i = f_i(x_1, x_2, \dots, x_n)$$

where $x_1, x_2, \dots, x_n$ are the input variables, then the function $\Delta_i = \varphi_i \overline{f_i(x_1, x_2, \dots, x_n)} \vee \vee \bar{\varphi} f_i(x_1, x_2, \dots, x_n) \equiv 0$ i.e. it describes the subset of states $M^{\sim}$.

Additional internal variables associated with outputs of the elements are introduced for each l-th output of the structure of a discrete device by eliminating from M^1 and M^0 the states characterised by the functions Δ_i.

Similarly as above, let us denote the functions characterizing the subsets of states M^1 and M^0 at the l-th output of the structure containing k elements may be described as follows

$$F_l^{1,k} = F_l^1(x_1, x_2, \dots, x_n)\bar{\psi}$$

$$F_l^{0,k} = F_l^0(x_1, x_2, \dots, x_n)\bar{\psi}$$

where

$$\psi = \bigwedge_{i=1}^{k} (\varphi_i \overline{f_i(x_1, x_2, \dots, x_n)} \vee \bar{\varphi}_i f_i(x_1, x_2, \dots, x_n).$$

INSTITUTE OF CONTROL SCIENCES
MOSCOW, USSR

References

[1] Клини, С. К., Представление событий в нервных сетях и конечных автоматах, Сб. Автоматы. Изд. Иностр. литер., 1956.

[2] Копыленко В. М., Синтез структур релейных устройств при задании исходых логических функций в произвольной скобочной форме, Сб. Проектирование информационно-логических устройств для ирригации, Изд. ИЛИМ, 1973.

[3] QUINE, W., The problem of simplifying truth function, *Amer. Math. Monthly*, v. 59, No. 8, 1952.

[4] MC CLUSKEY J., Minimization of Boolean functions, *Bell System Tech. J.*, v. 35, No. 6, 1956, pp. 1417—1444.

[5] Гаврилов, М. А., Минимизация булевых функций, характеризующих релейные цепи, *Автоматика и телемеханика*, v. 20, No. 9, 1959.

[6] Гаврилов, М. А., В. М. Копыленко, Метод минимизации структур бесконтактных релейных устройств на функционально-полных наборах элементов, Сб. Теория дискретных автоматов, Изд. Зинатне, Рига, 1967.

[7] GAVRILOV, M. A., Development of the finite automata theory in the discrete circuit design, *Problems of Control and Information Theory*, (Jubilee Number) v. 4, 1975, pp. 77—96.

[8] Гаврилов, М. А., Декомпозиция комбинационных автоматов, Сб. Математические структуры, вычислительная математика, математическое моделирование, София, 1975, pp. 37—56.

[9] AKERS, S., On a theory of Boolean function, *J. Soc. Indust. Appl. Math.*, v. 7, No. 4, 1959.

[10] Базарбаева, Т. Г., З. И. Вострова, В. М. Копыленко, В. М. Якутин, Проектирование структур релейных устройств с учетом их надежности и некоторых особенностей технической реализации, Дискретные сустемы, Международный симпозиум, v. 2, Изд. Зинатне, Рига, pp. 7—15.

(Received March 9, 1978)

Minimal ascending tree automata

By F. GÉCSEG and M. STEINBY

To the memory of Professor László Kalmár

Here an 'ascending tree recognizer' is a finite, deterministic automaton that reads trees starting at the root proceeding then towards the leaves along all branches. It accepts or rejects the tree depending on the states at which it arrives at the leaves. In the literature they have also been called 'climbing automata', 'top-down tree recognizers' and 'root-to-frontier automata'. They were first studied by MAGIDOR and MORAN [5]. Although various forms of ascending tree transducers have been studied (cf. [3], for example) the ascending tree recognizers have received little attention. A brief discussion can be found in THATCHER's [7] survey paper.

The minimization of (frontier-to-root) tree recognizers was first considered by BRAINERD [2]. Another formulation was given by ARBIB and GIVE'ON [1]. It turned out that Nerodes theorem (cf. [6]) and the classical minimization algorithms can be extended to them.

In this paper the minimization problem of ascending tree recognizers is studied. First we define some basic algebraic concepts for them (such as homomorphisms). In order to be able to generalize the results and procedures from the case of ordinary recognizers we have to restrict ourselves to 'normalized' ascending tree recognizers. However, every ascending tree recognizer is equivalent to such a normalized recognizer. From a connected normalized ascending tree recognizer a minimal recognizer can be obtained as a quotient recognizer. Also, it turns out that any two equivalent normalized minimal ascending tree recognizers are isomorphic. All steps involved in the process of transforming a given ascending tree recognizer into a minimal one are effective so a minimization algorithm results.

1. Trees and ascending tree recognizers

We shall define trees as polynomial symbols in the sense of GRÄTZER [4]. In this paper

$$F = \bigcup (F_m | m \geqq 1)$$

will be a finite set of *operational symbols.* For any $m \geqq 1$, F_m is the set of m-ary operational symbols and the sets F_m $(m \geqq 1)$ are assumed to be pairwise disjoint. Note that we exclude here 0-ary operational symbols.

For every $n \geqq 1$,

$$X_n = \{x_1, \dots, x_n\}$$

is a fixed set of *variables* and the set $T_{F,n}$ of *n-ary F-trees* is defined as the smallest set U such that

(1) $X_n \subseteq U$ and

(2) $f(p_1, \dots, p_m) \in U$ whenever $p_1, \dots, p_m \in U$ and $f \in F_m$ for some $m > 0$.

Definitions and proofs concerning trees will usually follow the inductive pattern of this definition.

An *n-ary (deterministic) ascending F-recognizer* ($n > 0$) is a system

$$\mathfrak{A} = (A, F, a_0, \mathbf{a}),$$

where

(1) A is the finite nonempty set of *states,*

(2) $a_0 \in A$ the *initial state,*

(3) $\mathbf{a} = (A_1, \dots, A_n) \in (2^A)^n$ the *final state vector* and

(4) every $f \in F_m$ ($m > 0$) is realized as a mapping

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

Henceforth $\mathfrak{A}$ and $\mathfrak{B}$ will be the n-ary ascending F-recognizers $(A, F, a_0, \mathbf{a})$ and $(B, F, b_0, \mathbf{b})$, respectively. Here $\mathbf{b} = (B_1, \dots, B_n)$. Since F and n are always given (although arbitrary) we shall often speak about ascending tree recognizers or simply about recognizers.

The operation of $\mathfrak{A}$ can be described as follows. The recognizer begins the examination of a given tree $p \in T_{F,n}$ at the 'root' in its initial state. If the root is labelled by $f \in F_m$, then it has m direct successors which are the roots of the corresponding subtrees and it will continue its operation by examining these subtrees starting in the states $a_1, \dots, a_m$, respectively, where $(a_1, \dots, a_m) = f^{\mathfrak{A}}(a_0)$. The process is repeated until $\mathfrak{A}$ has reached the 'leaves' along every branch of the tree. Every leaf is labelled by a variable. If a given leaf is labelled by x_i then $\mathfrak{A}$ should reach it in a state belonging to A_i. The tree p is accepted if this condition is satisfied for every leaf. It is easier to formalize this procedure by tracing it from the leaves back to the root. To this end we define a map

$$\alpha_{\mathfrak{A}}: T_{F,n} \to 2^A$$

as follows:

(1) $\alpha_{\mathfrak{A}}(x_i) = A_i$, for all $x_i \in X_n$, and

(2) $\alpha_{\mathfrak{A}}(p) = \{a \in A \mid f^{\mathfrak{A}}(a) \in \alpha_{\mathfrak{A}}(p_1) \times \dots \times \alpha_{\mathfrak{A}}(p_m)\}$, if $p = f(p_1, \dots, p_m)$ with $m > 0$, $f \in F_m$ and $p_1, \dots, p_m \in T_{F,n}$.

The *forest recognized* by $\mathfrak{A}$ can now be defined as

$$T(\mathfrak{A}) = \{p \in T_{F,n} \mid a_0 \in \alpha_{\mathfrak{A}}(p)\}.$$

The recognizers $\mathfrak{A}$ and $\mathfrak{B}$ are *equivalent* if $T(\mathfrak{A}) = T(\mathfrak{B})$. Furthermore, $\mathfrak{A}$ is called *minimal* if $|B| \geqq |A|$ whenever $\mathfrak{B}$ is a recognizer equivalent to $\mathfrak{A}$.

2. Some algebraic concepts

We shall now adapt some central algebraic notions for n-ary ascending tree automata.

A *homomorphism* of $\mathfrak{A}$ *onto* $\mathfrak{B}$ is a mapping $\varphi\colon A\to B$ onto B such that

(1) for all $m>0$, $f\in F_m$ and $a\in A$, $f^{\mathfrak{B}}(a\varphi)=(a_1\varphi, \dots, a_m\varphi)$, where $(a_1, \dots, a_m)=f^{\mathfrak{A}}(a)$,

(2) $a_0\varphi=b_0$ and

(3) for all $i=1, \dots, n$, $A_i\varphi=B_i$ and $B_i\varphi^{-1}=A_i$.

If φ is a homomorphism of $\mathfrak{A}$ onto $\mathfrak{B}$, we write $\varphi\colon \mathfrak{A}\to\mathfrak{B}$ and call $\mathfrak{B}$ a *homomorphic image* of $\mathfrak{A}$. If φ is also bijective, then it is called an *isomorphism*. We say that $\mathfrak{A}$ and $\mathfrak{B}$ are *isomorphic* and write $\mathfrak{A}\cong\mathfrak{B}$ if there exists an isomorphism $\varphi\colon\mathfrak{A}\to\mathfrak{B}$. Obviously, $\cong$ is a reflexive, symmetric and transitive relation among n-ary F-recognizers.

Let ϱ be an equivalence relation on a set S. Then

(i) s/ϱ is the ϱ-class determined by a given element $s\in S$,

(ii) $\mathbf{s}/\varrho=(s_1/\varrho, \dots, s_n/\varrho)$, if $\mathbf{s}=(s_1, \dots, s_n)\in S^n$ $(n\geqq 1)$,

(iii) $U/\varrho=\{u/\varrho \mid u\in U\}$, if $U\subseteq S$ and

(iv) $\mathbf{U}/\varrho=(U_1/\varrho, \dots, U_n/\varrho)$, if $\mathbf{U}=(U_1, \dots, U_n)\in(2^S)^n$

A *congruence relation* of the recognizer $\mathfrak{A}$ is now defined as an equivalence relation ϱ on A such that

(1) for all $m>0, f\in F_m$ and $a, a'\in A$, $a/\varrho=a'/\varrho$ implies $f^{\mathfrak{A}}(a)/\varrho=f^{\mathfrak{A}}(a')/\varrho$, and

(2) for all $i=1, \dots, n$ and $a\in A, a\in A_i$ implies $a/\varrho\subseteq A_i$.

If ϱ is a congruence relation of $\mathfrak{A}$, then the *quotient recognizer* of $\mathfrak{A}$ determined by ϱ is the n-ary F-recognizer

$$\mathfrak{A}/\varrho = (A/\varrho, F, a_0/\varrho, \mathbf{a}/\varrho)$$

where, for all $m>0$, $f\in F_m$ and $a\in A$,

$$f^{\mathfrak{A}/\varrho}(a/\varrho) = f^{\mathfrak{A}}(a)/\varrho.$$

It is easy to see that $\mathfrak{A}/\varrho$ is well-defined. As indicated in the next theorem the three concepts defined above are related to each other the same way their counterparts in algebra are. The straightforward proof is omitted.

Theorem 1. Let $\mathfrak{A}$ and $\mathfrak{B}$ be n-ary ascending F-recognizers.

a) If ϱ is a congruence of $\mathfrak{A}$, then $\mathfrak{A}/\varrho$ is a homomorphic image of $\mathfrak{A}$.

b) If $\varphi\colon\mathfrak{A}\to\mathfrak{B}$ is a homomorphism of $\mathfrak{A}$ onto $\mathfrak{B}$, then the kernel $\varrho=\varphi\varphi^{-1}$ of φ is a congruence relation of $\mathfrak{A}$ and $\mathfrak{B}\cong\mathfrak{A}/\varrho$.

The following observation will be used later.

Theorem 2. If $\mathfrak{B}$ is a homomorphic image of $\mathfrak{A}$, then $T(\mathfrak{A})=T(\mathfrak{B})$.

Proof. Let $\varphi\colon\mathfrak{A}\to\mathfrak{B}$ be a homomorphism of $\mathfrak{A}$ onto $\mathfrak{B}$. We show by induction on trees that for any $a\in A$, $a\in\alpha_{\mathfrak{A}}(p)$ iff $a\varphi\in\alpha_{\mathfrak{B}}(p)$.

(1) If $p=x_i\in X_n$, then it holds since $\alpha_{\mathfrak{A}}(p)=A_i$, $\alpha_{\mathfrak{B}}(p)=B_i$ and $\alpha_{\mathfrak{A}}(p)\varphi=\alpha_{\mathfrak{B}}(p)$, $\alpha_{\mathfrak{B}}(p)\varphi^{-1}=\alpha_{\mathfrak{A}}(p)$.

(2) Let $p=f(p_1, \dots, p_m)\in T_{F,n}$ be such that $\alpha_{\mathfrak{B}}(p_i)=\alpha_{\mathfrak{A}}(p_i)\varphi$ and $\alpha_{\mathfrak{B}}(p_i)\varphi^{-1}=\alpha_{\mathfrak{A}}(p_i)$ $(i=1, \dots, m)$.

Suppose $a\in\alpha_{\mathfrak{A}}(p)$. If $f^{\mathfrak{A}}(a)=(a_1,\ldots,a_m)$, then $a_1\in\alpha_{\mathfrak{A}}(p_1),\ldots,a_m\in\alpha_{\mathfrak{A}}(p_m)$. Hence $a_1\varphi\in\alpha_{\mathfrak{B}}(p_1),\ldots,a_m\varphi\in\alpha_{\mathfrak{B}}(p_m)$ which implies

$$f^{\mathfrak{B}}(a\varphi)=(a_1\varphi,\ldots,a_m\varphi)\in\alpha_{\mathfrak{B}}(p_1)\times\ldots\times\alpha_{\mathfrak{B}}(p_m).$$

Thus $a\varphi\in\alpha_{\mathfrak{B}}(p)$.

Suppose now that $a\varphi\in\alpha_{\mathfrak{B}}(p)$ and let $f^{\mathfrak{A}}(a)$ be $(a_1,\ldots,a_m)$. Then $a_1\varphi\in\alpha_{\mathfrak{B}}(p_1),\ldots$ $\ldots,a_m\varphi\in\alpha_{\mathfrak{B}}(p_m)$. This implies that

$$a_1\in\alpha_{\mathfrak{A}}(p_1),\ldots,a_m\in\alpha_{\mathfrak{A}}(p_m).$$

Hence, $a\in\alpha_{\mathfrak{A}}(p)$.

Now $p\in T(\mathfrak{A})$ iff $a_0\in\alpha_{\mathfrak{A}}(p)$
iff $a_0\varphi=b_0\in\alpha_{\mathfrak{B}}(p)$
iff $p\in T(\mathfrak{B})$
which completes the proof.

3. Normalized, connected and reduced recognizers

For any state a of the n-ary ascending F-recognizer $\mathfrak{A}$ we put

$$T(\mathfrak{A},a)=\{p\in T_{F,n}|a\in\alpha_{\mathfrak{A}}(p)\}.$$

The state a is called a *0-state* if $T(\mathfrak{A},a)=\emptyset$.

We say that $\mathfrak{A}$ is *normalized* if, for all $a\in A$, $m>0$ and $f\in F_m$, either all of the components of $f^{\mathfrak{A}}(a)$ are 0-states or none of them is a 0-state.

For any $\mathfrak{A}$ we define an n-ary ascending F-recognizer

$$\mathfrak{A}^*=(A,F,a_0,\mathbf{a})$$

as follows:

(a) if $\mathfrak{A}$ has no 0-state, then $\mathfrak{A}^*=\mathfrak{A}$ and

(b) if $\mathfrak{A}$ has 0-states choose one of them, say d, and define for all $a\in A$, $m>0$ and $f\in F_m$

$$f^{\mathfrak{A}^*}(a)=\begin{cases}(d,\ldots,d)\ (\in A^m), & \text{if } f^{\mathfrak{A}}(a) \text{ contains } a \text{ 0-state}\\ f^{\mathfrak{A}}(a) & \text{otherwise.}\end{cases}$$

Theorem 3. If $\mathfrak{A}$ is any n-ary ascending F-recognizer, then $\mathfrak{A}^*$ is normalized and $T(\mathfrak{A}^*)=T(\mathfrak{A})$.

Proof. We show by tree induction that

$$\alpha_{\mathfrak{A}}(p)=\alpha_{\mathfrak{A}^*}(p), \qquad (*)$$

for all $p\in T_{F,n}$.

(1) If $p=x_i\in X_n$, then $(*)$ holds since

$$\alpha_{\mathfrak{A}}(p)=A_i=\alpha_{\mathfrak{A}^*}(p).$$

(2) Let $p=f(p_1, \ldots, p_m)$, where $(*)$ holds for $p_1, \ldots, p_m$. Consider any $a\in A$. We have two possible cases:

(i) $f^{\mathfrak{A}}(a)$ contains no 0-state. Then $f^{\mathfrak{A}^*}(a)=f^{\mathfrak{A}}(a)$ and, by the inductive assumption,

$$\begin{aligned} a\in\alpha_{\mathfrak{A}}(p) \quad &\text{iff} \quad f^{\mathfrak{A}}(a)\in\alpha_{\mathfrak{A}}(p_1)\times \ldots \times\alpha_{\mathfrak{A}}(p_m) \\ &\text{iff} \quad f^{\mathfrak{A}^*}(a)\in\alpha_{\mathfrak{A}^*}(p_1)\times \ldots \times\alpha_{\mathfrak{A}^*}(p_m) \\ &\text{iff} \quad a\in\alpha_{\mathfrak{A}^*}(p). \end{aligned}$$

(ii) If $f^{\mathfrak{A}}(a)$ contains a 0-state, then it is easily seen that neither $a\in\alpha_{\mathfrak{A}}(p)$ nor $a\in\alpha_{\mathfrak{A}^*}(p)$ is possible.

The claim $T(\mathfrak{A}^*)=T(\mathfrak{A})$ follows immediately from $(*)$. Also, $(*)$ implies that no new 0-states were introduced in the construction of $\mathfrak{A}^*$ and hence that $\mathfrak{A}^*$ is normalized by its definition. $\square$

We call two states a and a' of $\mathfrak{A}$ *equivalent* and write $a\equiv b(\varrho_{\mathfrak{A}})$ if $T(\mathfrak{A}, a)= =T(\mathfrak{A}, a')$.

Clearly, $\varrho_{\mathfrak{A}}$ is an equivalence relation on A, and we call $\mathfrak{A}$ *reduced* if $\varrho_{\mathfrak{A}}$ is the identity relation on A.

Theorem 4. If $\mathfrak{A}$ is normalized then $\varrho_{\mathfrak{A}}$ is a congruence relation and $\mathfrak{A}/\varrho_{\mathfrak{A}}$ is reduced.

Proof. First we show that $\varrho_{\mathfrak{A}}$ is a congruence relation.

(1) Consider any $m>0$, $f\in F_m$ and $a, a'\in A$. Let

$$f^{\mathfrak{A}}(a) = (a_1, \ldots, a_m)$$

and

$$f^{\mathfrak{A}}(a') = (a_1', \ldots, a_m')$$

and suppose that $a\equiv a'$ $(\varrho_{\mathfrak{A}})$. Consider any i $(1\leqq i\leqq m)$ and suppose $p_i\in T(\mathfrak{A}, a_i)$. Then a_i is not a 0-state and therefore none of the states $a_1, \ldots, a_m$ is a 0-state and there exist trees

$$p_1\in T(\mathfrak{A}, a_1), \ldots, p_{i-1}\in T(\mathfrak{A}, a_{i-1}), \quad p_{i+1}\in T(\mathfrak{A}, a_{i+1}), \ldots, p_m\in T(\mathfrak{A}, a_m).$$

Then

$$f(p_1, \ldots, p_i, \ldots, p_m)\in T(\mathfrak{A}, a) = T(\mathfrak{A}, a')$$

implies $p_i\in T(\mathfrak{A}, a_i')$. Similarly, $p_i\in T(\mathfrak{A}, a_i')$ implies $p_i\in T(\mathfrak{A}, a_i)$. Hence $a_i \equiv \equiv a_i'$ $(\varrho_{\mathfrak{A}})$.

(2) If $a\in A_i$ and $a\equiv a'(\varrho_{\mathfrak{A}})$, for some $i=1, \ldots, n$ and $a, a'\in A$, then $x_i\in T(\mathfrak{A}, a)=T(\mathfrak{A}, a')$ implies $a'\in A_i$.

Since $\varrho_{\mathfrak{A}}$ is a congruence the quotient recognizer $\mathfrak{A}/\varrho_{\mathfrak{A}}$ can be defined. It is reduced as

$$a/\varrho_{\mathfrak{A}} \equiv a'/\varrho_{\mathfrak{A}} \ (\varrho_{\mathfrak{A}/\varrho_{\mathfrak{A}}}) \quad (a, a'\in A)$$

implies

$$a/\varrho_{\mathfrak{A}} = a'/\varrho_{\mathfrak{A}}$$

since, by Theorem 2,

$$T(\mathfrak{A}, a) = T(\mathfrak{A}/\varrho_{\mathfrak{A}}, a/\varrho_{\mathfrak{A}}) = T(\mathfrak{A}/\varrho_{\mathfrak{A}}, a'/\varrho_{\mathfrak{A}}) = T(\mathfrak{A}, a'). \quad \square$$

Let $a, a'\in A$. We write $a\Rightarrow_{\mathfrak{A}} a'$ if there exist an $m>0$ and an $f\in F_m$ such that a' appears in $f^{\mathfrak{A}}(a)$. The reflexive, transitive closure of the relation $\Rightarrow_{\mathfrak{A}}$ is denoted by $\Rightarrow_{\mathfrak{A}}^{*}$. If $a \Rightarrow_{\mathfrak{A}}^{*} a'$, then we say that a' is *reachable* from a. The recognizer $\mathfrak{A}$ is said to be *connected* if every state is reachable from the initial state.

The *connected component*

$$\mathfrak{A}^c = (A^c, F, a_0, \mathbf{a}^c)$$

of $\mathfrak{A}$ is the n-ary ascending F-recognizer defined as follows:

(i) $A^c = \{a \in A \mid a_0 \Rightarrow_{\mathfrak{A}}^{*} a\}$,

(ii) $\mathbf{a}^c = (A_1 \cap A^c, \ldots, A_n \cap A^c)$ and

(iii) for all $m > 0$ and $f \in F_m$, $f^{\mathfrak{A}^c}$ is defined as the restriction of $f^{\mathfrak{A}}$ to A^c. Clearly, the operations of $\mathfrak{A}^c$ are completely defined.

Lemma 5. Let $\mathfrak{A}$ be any n-ary ascending F-recognizer. Then
(1) $\mathfrak{A}^c$ is connected,
(2) $\mathfrak{A} = \mathfrak{A}^c$ iff $\mathfrak{A}$ is connected,
(3) $T(\mathfrak{A}^c) = T(\mathfrak{A})$ and
(4) if $\mathfrak{A}$ is normalized, then so is $\mathfrak{A}^c$.

Lemma 6. Let $\mathfrak{A}$ and $\mathfrak{B}$ be normalized, $a \in A$, $b \in B$, $m > 0$, $f \in F_m$, $f^{\mathfrak{A}}(a) = (a_1, \ldots, a_m)$ and $f^{\mathfrak{B}}(b) = (b_1, \ldots, b_m)$. If $T(\mathfrak{A}, a) = T(\mathfrak{B}, b)$, then $T(\mathfrak{A}, a_i) = T(\mathfrak{B}, b_i)$, for all $i = 1, \ldots, m$.

The straightforward proofs of these lemmas are omitted.

Theorem 7. Let $\mathfrak{A}$ and $\mathfrak{B}$ be connected, normalized n-ary ascending F-recognizers. Then $T(\mathfrak{A}) = T(\mathfrak{B})$ iff $\mathfrak{A}/\varrho_{\mathfrak{A}} \cong \mathfrak{B}/\varrho_{\mathfrak{B}}$.

Proof. If $\mathfrak{A}/\varrho_{\mathfrak{A}}$ and $\mathfrak{B}/\varrho_{\mathfrak{B}}$ are isomorphic, then $T(\mathfrak{A}) = T(\mathfrak{A}/\varrho_{\mathfrak{A}}) = T(\mathfrak{B}/\varrho_{\mathfrak{B}}) = T(\mathfrak{B})$ by Theorems 1 and 2.

Assume now that $T(\mathfrak{A}) = T(\mathfrak{B})$. We define a mapping

$$\varphi \colon A/\varrho_{\mathfrak{A}} \to B/\varrho_{\mathfrak{B}}$$

by

$$(a/\varrho_{\mathfrak{A}})\varphi = b/\varrho_{\mathfrak{B}} \quad \text{if} \quad T(\mathfrak{A}, a) = T(\mathfrak{B}, b)$$

$(a \in A, b \in B)$. The following steps (i)—(v) show that φ gives the required isomorphism.

(i) $(a/\varrho_{\mathfrak{A}})\varphi$ is defined for all $a/\varrho_{\mathfrak{A}} \in A/\varrho_{\mathfrak{A}}$. Since $\mathfrak{A}$ is connected there exists for every $a \in A$ an integer $k \geq 0$ and states $a_0, a_1, \ldots, a_k \in A$ such that

$$a_0 \Rightarrow_{\mathfrak{A}} a_1 \Rightarrow_{\mathfrak{A}} \ldots \Rightarrow_{\mathfrak{A}} a_{k-1} \Rightarrow_{\mathfrak{A}} a_k = a.$$

By induction on the length of the shortest such 'derivation' of a it can easily be shown using Lemma 6 that there exists for every $a \in A$ a $b \in B$ such that $T(\mathfrak{A}, a) = T(\mathfrak{B}, b)$.

(ii) φ is well-defined. If $T(\mathfrak{A}, a) = T(\mathfrak{B}, b) = T(\mathfrak{B}, b')$ for some $a \in A$ and $b, b' \in B$, then $b/\varrho_{\mathfrak{B}} = b'/\varrho_{\mathfrak{B}}$.

(iii) φ is injective. Obvious.

(iv) φ is surjective. Repeating the argument used in (i) with the roles of $\mathfrak{A}$ and $\mathfrak{B}$ reversed we see that there exists for every $b \in B$ an $a \in A$ such that $T(\mathfrak{A}, a) = T(\mathfrak{B}, b)$.

(v) φ is a homomorphism. That φ preserves operations follows from Lemma 6. If $a/\varrho_{\mathfrak{A}} \in A_i/\varrho_{\mathfrak{A}}$ $(1 \leq i \leq m)$ and $(a/\varrho_{\mathfrak{A}})\varphi = b/\varrho_{\mathfrak{B}}$, then $x_i \in T(\mathfrak{A}, a) = T(\mathfrak{B}, b)$ implies $b/\varrho_{\mathfrak{B}} \in B_i/\varrho_{\mathfrak{B}}$. Likewise, $(a/\varrho_{\mathfrak{A}})\varphi = b/\varrho_{\mathfrak{B}} \in B_i/\varrho_{\mathfrak{B}}$ implies $a/\varrho_{\mathfrak{A}} \in A_i/\varrho_{\mathfrak{A}}$. □

4. Minimal recognizers and minimization

Suppose $\mathfrak{A}$ is minimal. From Lemma 5 it follows that $\mathfrak{A}$ is connected and from Theorem 3 that we may assume that $\mathfrak{A}$ is normalized. Then $T(\mathfrak{A}/\varrho_{\mathfrak{A}})=T(\mathfrak{A})$ by Theorems 2 and 4. Hence, $\mathfrak{A}$ is reduced.

Conversely, if $\mathfrak{A}$ is connected, normalized and reduced, then it is minimal and every normalized minimal recognizer equivalent to it is also isomorphic to it (Theorem 7).

These facts imply that the following three steps yield for any $\mathfrak{A}$ an equivalent minimal recognizer $\mathfrak{B}$. Moreover, this $\mathfrak{B}$ is normalized and it depends, up to isomorphism, on $T(\mathfrak{A})$ only.

Step 1. Form $\mathfrak{A}^*$.

Step 2. Form $\mathfrak{A}^{*c}$.

Step 3. Form $\varrho_{\mathfrak{A}^{*c}}$ and put $\mathfrak{B}=\mathfrak{A}^{*c}/\varrho_{\mathfrak{A}^{*c}}$.

We shall now verify that these steps are effectively realizable and thus constitute a minimization algorithm for ascending tree recognizers.

Let us define the sets $H_k\subseteq A$, $k=0, 1, \ldots,$ as follows:

(i) $H_0=\{a_0\}$

and, for all $k=0, 1, \ldots,$

(ii) $H_{k+1}=H_k\cup\{a\in A\,|\,a'\Rightarrow_{\mathfrak{A}} a,$ for some $a'\in H_k\}$. Clearly,

$$H_0\subseteq H_1\subseteq H_2\subseteq \ldots$$

and

$$A^c = \bigcup (H_k|k\geqq 0).$$

Since $H_{k+1}=H_k$ implies $H_k=H_{k+j}$, for all $j\geqq 0$, A^c can be obtained as $H_{|A|-1}$.

For any ascending tree recognizer $\mathfrak{A}$ an equivalent (frontier-to-root) tree recognizer can be constructed (cf. [5] or [7], for example). Thus the questions "$T(\mathfrak{A}, a)=\emptyset$?" and "$T(\mathfrak{A}, a)=T(\mathfrak{A}, a')$?" are decidable. (This could easily be shown directly without any reference to frontier-to-root tree automata.) Hence the 0-states can be found and $\varrho_{\mathfrak{A}}$ can be formed. Thus Steps 1 and 3 are also effective.

These results are summed up in the following theorem.

Theorem 8. A normalized ascending tree recognizer is minimal iff it is connected and reduced. For any ascending tree recognizer there exists an equivalent normalized minimal ascending tree recognizer. This is unique up to isomorphism and it can effectively be constructed.

The reduction of an ascending tree automaton can also be done the same way as ordinary finite recognizers (and tree recognizers in [2]) are reduced. Given $\mathfrak{A}$ we define a sequence of equivalence relations $\varrho_0, \varrho_1, \ldots,$ on A as follows: for any $a, a'\in A$

(i) $a\equiv a'(\varrho_0)$ iff $a\in A_i\Leftrightarrow a'\in A_i$, for all $i=1, \ldots, n$, and for $k=0, 1, \ldots,$

(ii) $a\equiv a'(\varrho_{k+1})$ iff $a\equiv a'(\varrho_k)$ and $f^{\mathfrak{A}}(a)/\varrho_k=f^{\mathfrak{A}}(a')/\varrho_k$ for all $m>0, f\in F_m$.

It is easy to see that $\varrho_{\mathfrak{A}}=\varrho_k$, for some $k<|A|$.

UNIVERSITY OF SZEGED
DEPARTMENT OF COMPUTER SCIENCE
6720 SZEGED, HUNGARY

UNIVERSITY OF TURKU
DEPARTMENT OF MATHEMATICS
20500 TURKU 50, FINLAND

References

[1] ARBIB, M. A. and Y. GIVE'ON, Algebra automata 1: Parallel programming as a prolegomena to the categorical approach, *Information and Control,* v. 12, 1968, pp. 331—345.

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

[3] ENGELFRIET, J., Bottom-up and top-down tree transformations, A comparison, *Math. Systems Theory,* v. 9, 1975, pp. 198—231.

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

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

[6] RABIN, M. O. and SCOTT, D., Finite automata and their decision problems, *IBM J. Res. Develop.,* v. 3, No. 2, 1959, pp. 114—125.

[7] THATCHER, J. W., Tree automata: an informal survey, Currents in the theory of computing (A. V. Aho, ed.), Englewood Cliffs, 1973, pp. 143—172.

(Received Jan. 3, 1978)

On the incompleteness of proving partial correctness

By T. GERGELY and M. SZŐTS

To the memory of Professor László Kalmár

1. Introduction

Our paper deals with the question, whether there exists complete calculus to prove partial correctness of programs. The first really important result in program verification was the method of inductive assertions introduced by R. W. Floyd [1]. (Later the method was reformulated by Hoare [2] so we call it Floyd—Hoare method.)

Also nowadays this is the most widespread method used in program verification and it proves partial correctness, so the question of completeness raised by us is not without importance. Z. Manna formalized this method in strict classical logic [3]. Several papers can be found in the relevant literature claiming the Floyd—Hoare method being complete (e.g.: [4] p. 237 Prob. 3—19, [5], [6]). We shall show that in the proofs of completeness some model theoretical questions were neglected. We investigate this method in model theoretical point of view, and prove that there is no complete method for proving partial correctness, and show the causes why the Floyd—Hoare method can be incomplete.

2. General principles

The existing programing languages have two features relevant to proving program properties:

— Only their syntax is formally defined, their semantics are informal.

— Statements about program properties can not be expressed in the programming language itself.

However in program verification one deals with semantic properties of programs in a formal way. In the followings we outline the way how we ensure the ability to do so.

(i) We select a language to express program properties. Since we want to handle these properties by mathematical tools we choose language in the form $\mathbf{L}=\langle L, M_L, \models\rangle$ where L is a formal syntax, M_L is the class of models, $\models$ is the validity relation (see e.g. [7]).

(ii) We interpret the programs in the models of **L**. Let P be the formal syntax of the programming language. Every program $p \in P$ is a static description. We define some mathematical objects on the models of **L** expressing the dynamics and consider it as the formal meaning of the program. Since **L** speaks about programs this meaning has to be describable by formulas of **L**. In favourable cases it can be defined, but weaker specification can be enough for some purposes.

Having defined the formal meaning of programs we can introduce an interpreting function k. To every model $\mathfrak{A} \in M_L$ and program $p \in P$ k renders the meaning of p in $\mathfrak{A}$. So we get a programming language with mathematical semantics: $\mathbf{P} = \langle P, M_L, k \rangle$.

(iii) The intuitive semantics of programming languages speak not only about the way of execution of the commands, but contain also constraits on the systems which the programs can be executed on. In our way of handling programs the models of **L** stand for these systems, so the constraits fix a subclass M_p of M_L as the model class of **P**. M_p is said to be called the class of intended models. So the programming language is: $\mathbf{P} = \langle P, M_p, k \rangle$, and the language speaking about programs: $\mathbf{L}_p = \langle L, M_p, \models \rangle$. Since **L** is the language speaking about programs, it is expedient if M_p can be specified by the expressions of **L**. (It is the case if the constraints can be expressed in **L**.)

(iv) Our aim is not only to express but also to handle formally program properties, that is to prove them. In (ii) we stipulate that the meaning (executions) of a program can be expressed in the formulas of **L**. If we succeed to formalize program properties in **L** and **L** has a calculus, the program properties can be proved by this calculus. So a calculus for program verification consists of two constituents:

a) An algorithm to construct a formula of **L** for the program property in question.

b) The calculus of **L**.

If we have the algorithm of a), the completeness of program verifying calculus depends on the completeness of $\mathbf{L}_p = \langle L, M_p, \models \rangle$.

In this paper we work out these steps for the case when **L** is the language of first order classical logic. Our aim is to examine the provability of partial correctness of programs. We use [8] as standard reference for the logical notions used here.

3. Interpreting programs in relational structures

Let $t = \langle t', t'' \rangle$ be a similarity type. We introduce the notion of "t-type programming language". (The type t determines the function and relation symbols occuring in the language.) According to it **L** will be the t-type first order classical language.

For the definitions of logic see [8].

Definition 1. We define the syntax of the t-type programming language (P_t).
(i) Symbols of the language:
a) Set of the program variable symbols: $Y = \{y_0, y_1, \ldots, y_i, \ldots\}_{i \in \omega}$
b) Function and relation symbols: $Do(t')$ and $Do(t'')$
c) Logical connectives of classical logic: $\{\neg, \wedge\}$

d) Set of labels: $I=\{l_0, \ldots, l_i, \ldots\}_{i\in\omega}$
e) Special symbols: $\{\leftarrow, \text{IF}, \text{THEN}, (,), :, ;, ,\}$
The above mentioned sets are pairwise disjoint.
(ii) Set of commands: $C=C_a\cup C_c$, where
a) C_a is the set of assignement commands:

$$C_a = \{y_i \leftarrow f(y_{i_1}, \ldots, y_{i_n}): y_i, y_{i_1}, \ldots, y_{i_n}\in Y,\ f\in Do(t''),\ t''(f) = n\}$$

b) C_c is the set of control commands:

$$C_c = \{\text{IF } \varrho(y_{i_1}, \ldots, y_{i_n}) \text{ THEN } l_i\text{: } l_i\in I,$$

$$y_{i_1}, \ldots, y_{i_n}\in Y,\ \varrho(y_{i_1}, \ldots, y_{i_n}) \text{ is a quantifier-free formula of } \mathbf{L}_t\}.$$

(iii) The expressions of the programming language are the finite sequences of labelled commands:

$$P_t = \{l_0\colon U_0; l_1\colon U_1; \ldots; l_n\colon U_n;\ l_0, l_1, \ldots, l_n\in I,\quad U_0, U_1, \ldots, U_n\in C,$$

$$\forall i,j < n\ l_i \neq l_j \text{ if } i\neq j\}.$$

These sequences are called t-type programs. □

Let $p\in P_t$. The set of variable symbols occuring in p is designated as Y_p, the set of labels labelling the commands of p as I_p', the set of labels occuring in it's control commands as I_p''. Let l_y the first label not in I_p', then $I_p=I_p'\cup I_p''\cup\{l_y\}$.

Example 1. Let t be the type of arithmetic, $I=\omega$. Then

```
0: y_2 ← 0;
1: y_3 ← 1;
2: IF y_2 = y_1 THEN 6;
3: y_2 ← y_2 + 1;
4: y_3 ← y_3 · y_2;
5: IF y_1 = y_1 THEN 2;
```

is a program. If we designate this program by p, then

$$Y_p = \{y_1, y_2, y_3\},\quad I_p' = \{0, 1, 2, 3, 4, 5\},\quad I_p'' = \{2, 6\},\quad l_y = 6$$

so

$$I_p = \{0, 1, 2, 3, 4, 5, 6\}.\quad □$$

The intuitive meaning of the commands is the usual. We stipulate that the execution of a program starts from the first command $(l_0\colon U_0)$, the variables of the program get their input values before the execution of it. The execution of the program stops when control is given to a label not occuring in I_p' and the values of the program variables at this state will be called the output of the program. All these notions will be soon defined precisely.

The language P is minimal in some respect: some kind of assignment and control commands are needed to build programs. We neglected input-output commands, and do not speak about subroutines. The reason of it is not that if we could not carry on the same investigation having these kind of commands, but that the result would be the same.

Now in accordance with our principles laid down in the preceding section we shall interpret programs in the model class of classical logic, that is in the class of relational structures. Our definition will reflect the intuition that the meaning of a program is it's execution.

Definition 2. Let $\mathfrak{A}\in M_L$ be a model, $p=l_0: U_0;\ l_1: U_1;\ \dots;\ l_m: U_m; \in P_t$ be a program and $k_j: Y_p\to A$ be an assignment function for every j (A is the universe of $\mathfrak{A}$). A *trace* of program p in model $\mathfrak{A}$ is a sequence of pairs of a label and an assignment function, if the following rules (i)—(iii) are satisfied.

(i) $s_0 \overset{d}{=} \langle l_0, k_0\rangle$, that is the sequence starts with a pair having the label of the first command in the program (i.e. the execution of the program starts at the first command). Here k_0 is arbitrary, the values of k_0 are called the *input* values of the program variables.

(ii) Let $s_j=\langle l_i, k_j\rangle$ and $l_i\in L_p'$. Then the next trace element (s_{j+1}) will be constructed by the following way, depending on U_i (the command labelled by l_i).

a) If $U_i\in C_a$, that is $U_i=y_k\leftarrow f(y_{i_1}, \dots, y_{i_n})$, then: $s_{j+1}\overset{d}{=}\langle l, k_{j+1}\rangle$, where

$$l=\begin{cases} l_{i+1} & \text{if } \ i<m \\ l_y & \text{if } \ i=m \end{cases}$$

$$k_{j+1}(y_k)=\begin{cases} k_j(y_h) & \text{if } \ h\neq k \\ f^{\mathfrak{A}}(k_j(y_{i_1}), \dots, k_j(y_{i_n})) & \text{if } \ h=k. \end{cases}$$

(Note that $f^{\mathfrak{A}}(k_j(y_{i_1}), \dots, k_j(y_{i_n}))$ is the value of the term $f(y_{i_1}, \dots, y_{i_n})$ in according to the k_j assignment function.)

b) If $U_i\in C_c$ that is $U_i=\text{IF}\varrho(y_{i_1}, \dots, y_{i_n})$ THEN l_c, then: $s_{j+1}\overset{d}{=}\langle l, k_j\rangle$, where

$$l=\begin{cases} l_{i+1} & \text{if } \ \mathfrak{A}\not\models \varrho(y_{i_1}, \dots, y_{i_n})[k_j] \\ l_c & \text{if } \ \mathfrak{A}\models \varrho(y_{i_1}, \dots, y_{i_n})[k_j]. \end{cases}$$

(iii) Let $s_j=\langle l_i, k_j\rangle$ and $l_i\notin I_p'$. In this case there is no s_{j+1} element, so the length of the sequence is $j+1$. The values of k_j are called the *output* values of the program variables.

So if s is a trace, then $s\in \bigcup\limits_{0<N\leq\omega} {}^{N}(I_p\times {}^{Y_p}A)$, $j+1$ is called the length of the trace, the elements of $I_p\times {}^{Y_p}A$ are called trace elements. □

The rules of the definition will be refered later as rules 2(i), 2(ii), 2(iii) respectively.

It can be seen that Definition 2 formalizes the intuitive meaning we circumscribed after the definition of syntax. Rule 2(ii) determine the correct meaning of the commands, rules 2(i) and 2(iii) the start and stop of execution. Rule 2(iii) determines whether a trace is finite or not. In the first case the execution *terminates*.

We shall use the following notation: instead of the assignment function k we sometimes write the values of k (as a vector: $\bar{b}$). Since the domain of k is ordered, this notation does not give place to misunderstanding: $k(y_i)=b_i$ ($i\in\omega$).

Example 2. Let p be the program shown in Example 1, $\mathfrak{N}$ be the standard model of arithmetic. In this case s is a trace of p in $\mathfrak{N}$:

$$\begin{aligned}s = \langle &\langle 0, [3, 1, 1]\rangle, \langle 1, [3, 0, 1]\rangle, \langle 2, [3, 0, 1]\rangle,\\ &\langle 3, [3, 0, 1]\rangle, \langle 4, [3, 1, 1]\rangle, \langle 5, [3, 1, 1]\rangle, \langle 2, [3, 1, 1]\rangle,\\ &\langle 3, [3, 1, 1]\rangle, \langle 4, [3, 2, 1]\rangle, \langle 5, [3, 2, 2]\rangle, \langle 2, [3, 2, 2]\rangle,\\ &\langle 3, [3, 2, 2]\rangle, \langle 4, [3, 3, 2]\rangle, \langle 5, [3, 3, 6]\rangle, \langle 2, [3, 3, 6]\rangle,\\ &\langle 6, [3, 3, 6]\rangle\rangle.\end{aligned}$$

We say that with the input [3, 1, 1] p terminates in $\mathfrak{N}$ and gives [3, 3, 6] as output. □

For the following investigations we need some auxilary definitions:

Definition 3. Let $p \in P_t$. Then a *partial end-trace* of p is a sequence of trace elements satisfying rules 2(ii), 2(iii). (Intuitively: the execution of p may start at any command in p.)

Let 2(ii)′ be a modified form of 2(ii). In 2(ii)′ the condition of 2(ii) reads: "If $s_j = \langle l_i, k_j\rangle$, and s_j is not the last element in s, ..."

A *partial trace* of p is a sequence of trace elements satisfying 2(ii), and 2(iii). (That is the length of a partial trace in not determined by 2(iii).) □

Having interpreted programs in the models of **L** we can define our programming language:

Definition 4. The programming language is a triple:

$$\mathbf{P}_t = \langle P_t, M_p, k\rangle,$$

where P_t is defined in Definition 1, $M_p \subseteqq M_t$, k is the interpreting function:

$$Do(k) = M_p \times P_t, \quad k(\mathfrak{A}, p) = \{s : s \text{ is a trace of } p \text{ in } \mathfrak{A}\}. \quad \square$$

It is one of the interesting questions how to determine M_p. In the literature two cases are discussed (see e.g. [4] chapter 4). The first is when $M_p = M_t$, that is the programs can be interpreted in any relational structure. In this case they are called program schemes. The second is when $M_p = \{\mathfrak{A}\}$, $\mathfrak{A} \in M_t$, that is the programs are interpreted in one specific model. Intuitively that is what we mean by programs, e.g. if t is the type of arithmetic, the programs are intended to be executed in the standard model of arithmetic. However the question arises, how to characterise the choosen model. It can be done in model theoretic way (using some metalanguage) or by the second order classical language, but usually first order language has no power enough — saved the case of finite models. If we want to use the first order logic as semantic describing language we have to stick to its usage characterizing M_p. So we have to give a first order theory T, and M_p will be the class of models of T: $M_p = \mathrm{Mod}\,(T)$. Then the formulas of T will be the non-logical axioms of the programming language. As we said in the previous section, wanting a complete calculus for proving program properties the language $\langle L, \mathrm{Mod}\,(T), \models\rangle$ has to have a complete one. It is equivalent with the condition that T should be axiomatized, that is there should be a recursive set of formulas (Ax_T) which all the formulas of T can be deduced from. So usually we define M_p as Mod (Ax_T).

According to the principles laid down in the previous section we have to find a first order description of the traces which we used in the interpretation of programs. This will be done in the next section.

4. Description of semantics in first order logic

The power of first order classical logic does not ensure the description of the mathematical object (set of traces) which we have introduced above to handle meaning of programs. This is a natural consequence of the contradiction between the dynamics of programs and the static nature of classical logic. So we have to look for mathematical objects characterizing traces and being describable by classical language.

Definition 5. Let $p \in P_t$, $\mathfrak{A} \in M_P$ and s be a trace of p in $\mathfrak{A}$, $l \in I_p$, then the *l-volume* of s is:

$s|l = \{\bar{b}$: there is trace element in s of the form $\langle l, \bar{b}\rangle\}$ □

It is evident that the l-volume of a trace is a relation defined on the universe of a model, so it can be expressed by the classical language. For the following study let us fix a program p with n program variables. To examine this program we extend L_t with new relational symbols Q_j for every $l_j \in I_p$, $t''(Q_j) = 2 \cdot n$.

(About the extension of a language see [8]). Our intention is that this new symbols should describe the l_j-volume of the traces of the program, where the invidual traces will be denoted by their input values (therefore the $2 \cdot n$ arity).

Formally:

Definition 6. Let $\mathfrak{A}$ be a model, $Q_0^{\mathfrak{A}}$ be a $2 \cdot n$-ary relation on A such that, if $\langle a_0, \ldots, a_{n-1}, b_0, \ldots, b_n, \ldots, b_{n-1}\rangle \in Q_0^{\mathfrak{A}}$ then $a_i = b_i$ for every $0 \leq i < n$. (So Q_0 may be the l_0-volume of a trace of p in $\mathfrak{A}$). Then we define $2 \cdot n$-ary relations on A for every $l_j \in L_p$: $\langle a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}\rangle \in Q_{0,j}^{\mathfrak{A}}$ iff the following conditions are satisfied:

(i) $\langle b_0, \ldots, b_{n-1}, b_0, \ldots, b_{n-1}\rangle \in Q_0^{\mathfrak{A}}$,

(ii) $\langle a_0, \ldots, a_{n-1}\rangle$ is an element of the l_j-volume of the trace with input $\langle b_0, \ldots, b_{n-1}\rangle$.

The $Q_{0,j}^{\mathfrak{A}}$ relations (defined by the executions of the program) are called the *minimal relations for Q_0 in $\mathfrak{A}$*. □

So we want to construct such first order formulas, those whose satisfaction can assure that the relations corresponding to symbols Q_j are the minimal relations corresponding to $Q_0^{\mathfrak{A}}$. Now we define axiom schemes formalizing the rules of traces in Definition 2.

Definition 7. Let us define axiom schemes in the following way:

(i) At the begining of the execution the program variables get their input values:

$$\sigma_0: \ \forall \bar{x} Q_0(\bar{x}, \bar{x}).$$

(ii) The effect of the commands:
a, assignment command: $l_i: y_k \leftarrow f(\bar{y})$

$$\sigma_{i,i+1}: \forall \bar{x}, \bar{y}[Q_i(\bar{y}, \bar{x}) \rightarrow Q_{i+1}(y_1, \dots, y_{k-1}, f(\bar{y}), y_{k+1}, \dots, y_n, \bar{x})]$$

b, control command: l_i: IF $\varrho(\bar{y})$ THEN l_j

$$\sigma_{i,i+1}: \forall \bar{x}, \bar{y}[Q_i(\bar{y}, \bar{x}) \wedge \neg \varrho(\bar{y}) \rightarrow Q_{i+1}(\bar{y}, \bar{x})]$$

$$\sigma_{i,j}: \forall \bar{x}, \bar{y}[Q_i(\bar{y}, \bar{x}) \wedge \varrho(\bar{y}) \rightarrow Q_j(\bar{y}, \bar{x})]. \quad \square$$

Comparing Definition 2 with Definition 7, we can see that:

1. The rule 2(i) is not totally formalized, the axiom $\forall \bar{x} Q_0(\bar{x}, \bar{x})$ ensures only the identity of input.

2. The effect of the statements (rule 2(ii)) is totally formalized.

3. Rule 2(iii) is formalized indirectly by the fact that there is no axiom scheme of the form: $Q_j(\bar{y}, \bar{x}) \rightarrow \dots$ if $l_j \notin I_p'$.

The proposition below says that the axiom scheme for the commands formalize exactly rule 2(ii). It follows immediatly from Definitions 2 and 7:

Proposition 1. For every $i, j \in I_p$, $\mathfrak{A} \in M_p$ and relations $Q_i^{\mathfrak{A}}, Q_j^{\mathfrak{A}}$, *if there exists* the $\sigma_{i,j}$ axiom then the following statements are equivalent:

(i) $\langle \mathfrak{A}, Q_i^{\mathfrak{A}}, Q_j^{\mathfrak{A}} \rangle \models \sigma_{i,j}$

(ii) For every $\langle \bar{a}, \bar{d} \rangle \in Q_i^{\mathfrak{A}}$ if there is a partial trace of the form $\langle \langle i, \bar{a} \rangle, \langle j, \bar{b} \rangle \rangle$, then $\langle \bar{b}, \bar{d} \rangle \in Q_j^{\mathfrak{A}}$. $\square$

Let us apply the relevant axiom scheme for every command of program p. The set of formulas got in this way will be considered the description of the program p, we denote it by Σ_p.

Example 3. Let p be the program shown in Example 1. Then:

$$\begin{aligned}\Sigma_p = \{&\forall \bar{x} Q_0(\bar{x}, \bar{x}),\\
&\forall \bar{x}, \bar{y}[Q_0(\bar{y}, \bar{x}) \rightarrow Q_1(y_1, 0, y_2, \bar{x})],\\
&\forall \bar{x}, \bar{y}[Q_1(\bar{y}, \bar{x}) \rightarrow Q_2(y_1, y_2, 1, \bar{x})],\\
&\forall \bar{x}, \bar{y}[Q_2(\bar{y}, \bar{x}) \wedge y_2 \neq y_1 \rightarrow Q_3(\bar{y}, \bar{x})],\\
&\forall \bar{x}, \bar{y}[Q_3(\bar{y}, \bar{x}) \rightarrow Q_4(y_1, y_2+1, y_3, \bar{x})],\\
&\forall \bar{x}, \bar{y}[Q_4(\bar{y}, \bar{x}) \rightarrow Q_5(y_1, y_2, y_3 \cdot y_2, \bar{x})],\\
&\forall \bar{x}, \bar{y}[Q_5(\bar{y}, \bar{x}) \wedge y_1 = y_1 \rightarrow Q_2(\bar{y}, \bar{x})],\\
&\forall \bar{x}, \bar{y}[Q_2(\bar{y}, \bar{x}) \wedge y_2 = y_1 \rightarrow Q_6(\bar{y}\ \bar{x})],\\
&\forall \bar{x}, \bar{y}[Q_5(\bar{y}, \bar{x}) \wedge y_1 \neq y_1 \rightarrow Q_6(\bar{y}, \bar{x})\}. \quad \square\end{aligned}$$

In the following we analyse what extent Σ_p describes program p to.

Theorem 1. Let $\mathfrak{A} \in M_p$.

(i) For arbitrary minimal relations:

$$\langle \mathfrak{A}, Q_0^{\mathfrak{A}}, \dots, Q_{0,i}^{\mathfrak{A}}, \dots \rangle \models \Sigma_p$$

(ii) If for a given $\langle Q_i^{\mathfrak{A}}\rangle_{i\in I_p}$ $\langle \mathfrak{A}, Q_0^{\mathfrak{A}}, \ldots, Q_i^{\mathfrak{A}}, \ldots\rangle \models \Sigma_p$; then $Q_{0,i}^{\mathfrak{A}} \subseteq Q_i^{\mathfrak{A}}$.
For the proof of the theorem we need the following

Lemma. The following statements are equivalent:
(i) $\langle \mathfrak{A}, Q_0^{\mathfrak{A}}, \ldots, Q_i^{\mathfrak{A}}, \ldots\rangle_{l_i\in I_p} \models \Sigma_p$
(ii) For every partial trace $\langle\langle l_{i_0}, \bar{a}\rangle, \ldots, \langle l_{i_j}, \bar{b}\rangle\rangle$ if $\langle \bar{a}, \bar{d}\rangle \in Q_{i_0}^{\mathfrak{A}}$ then for every trace element $\langle i_k, \bar{c}\rangle$ occuring in the partial trace in question we have $\langle \bar{c}, \bar{d}\rangle \in Q_{i_k}^{\mathfrak{A}}$

Proof of lemma. (i) Let us suppose that $\langle \mathfrak{A}, Q_0^{\mathfrak{A}}, \ldots, Q_i^{\mathfrak{A}}, \ldots\rangle \models \Sigma_p$. We shall prove the lemma by induction on the length of the partial traces.

a. For two element traces lemma says the same as Proposition 1.

b. Let us suppose that the proposition of the lemma stands for every partial trace with length shorter then n. Let $s = \langle s', \langle l_j, \bar{b}\rangle\rangle$, where the length of s' is $n-1$. If $\langle l_k, \bar{c}\rangle$ is a trace element from s', the proposition stands for it because of the inductive hypothesis. Let $\langle l_m, \bar{c}\rangle$ the last element of s', so $\langle \bar{c}, \bar{d}\rangle \in Q_m^{\mathfrak{A}}$. Let us apply Proposition 1 to the partial trace $\langle\langle l_m, \bar{c}\rangle, \langle l_j, \bar{b}\rangle\rangle$ and we get that $\langle \bar{b}, \bar{d}\rangle \in Q_j^{\mathfrak{A}}$.

(ii) It is enough to consider the two element partial traces and then Proposition 1 is got. □

Proof of theorem. (i) (ii) of the lemma stands also for the minimal relations. Thus, by the lemma, $\langle \mathfrak{A}, Q_0^{\mathfrak{A}}, \ldots, Q_{0,i}^{\mathfrak{A}}, \ldots\rangle_{l_i\in I_p} \models \Sigma_p$.

(ii) Let us suppose that $\langle \mathfrak{A}, Q_0^{\mathfrak{A}}, \ldots, Q_i^{\mathfrak{A}}, \ldots\rangle \models \Sigma_p$. So also (ii) of the lemma stands for every trace having $\bar{d}$ as input if $\langle \bar{d}, \bar{d}\rangle \in Q_0^{\mathfrak{A}}$. For this case (ii) of theorem is equivalent to (ii) of lemma. □

If we could have proved that a family of relations $\langle Q_i^{\mathfrak{A}}\rangle_{l_i\in I_p}$ satisfies Σ_p iff it consists of volumes of traces, we could say that Σ_p describes totally the program p. This theorem shows that it is not the case. The next proposition shows the power of Σ_p.

(The proposition is an immediate consequence of the above lemma.)

Proposition 2. If a family of relations $\langle Q_i^{\mathfrak{A}}\rangle_{l_i\in I_p}$ satisfies Σ_p, all the relations are volumes of partial end-traces. □

This proposition shows clearly that our failure describing programs totally in first order logic comes from the fact that we could not formulize rule 2(i). Intuitively Proposition 2 says, that Σ_p allows to start the execution of a program at a command different from the first one. This failure is not due to our inadequency, later we prove that the volumes of traces (the minimal relations) can not be defined by first order formulas.

However the power of Σ_p is enough to prove properties of programs. The key of complete proof procedures is our ability to express the programs properties in our semantic description language, that is in the first order classical language. So to make a program property provable we have to find first order formula which describes this property. Succeeding with this we can prove the program property in question by proving this formula from $Ax_T \cup \Sigma_p$ by a calculus of first order logic. We show an example in the following.

Definition 8. A program p is totally correct in a model $\mathfrak{A}$ with respect to (w.r.t.) the input condition $\varphi(x)$ and output condition $\psi(\bar{y}, \bar{x})$ iff for every input $(\bar{a})$ satisfying $\varphi(\bar{x})$ the appropriate trace in $\mathfrak{A}$ terminates and the output $(\bar{b})$ satisfies $\psi(\bar{y}, \bar{x})$. □

Theorem 2. A program p is totally correct in every modell of Ax_T w.r.t. $\varphi(\bar{x})$ and $\psi(\bar{y}, \bar{x})$ iff

$$Ax_T \cup \Sigma_p \vdash \forall \bar{x}[Q_0(\bar{x}, \bar{x}) \wedge \varphi(\bar{x}) \to \exists \bar{y}(\psi(\bar{y}, \bar{x}) \wedge (\bigvee_{l_i \in I_p - I'_p} Q_i(\bar{y}, \bar{x})))].$$

The proof of this theorem is not difficult using Theorem 1. For soundness use (i) of the theorem, for completeness (ii). □

5. Provability of partial correctness

Definition 9. A program p is partially correct in a model $\mathfrak{A}$ w.r.t. $\varphi(\bar{x})$ and $\psi(\bar{y}, \bar{x})$ iff for every input $(\bar{a})$ satisfying $\varphi(\bar{x})$ the output of the appropriate trace satisfies $\psi(\bar{y}, \bar{x})$. (So we do not demand the trace to terminate for every input statifying $\varphi(\bar{x})$, but if it does for some of them the output must be correct.) We shall use a shorthand for partial correctness: (φ, p, ψ). □

Let us substitute $\varphi(\bar{x}) \wedge \bar{y} = \bar{x}$ for $Q_0(\bar{y}, \bar{x})$ and $\psi(\bar{y}, \bar{x})$ for every $Q_j(\bar{y}, \bar{x})$ when $l_j \in I_p \setminus I'_p$, in Σ_p. The obtained set of formulas is denoted by $\Sigma_p(\varphi, \psi)$.

Theorem 3. A program is partially correct in every model of Ax_T w.r.t. $\varphi(\bar{x})$ and $\psi(\bar{y}, \bar{x})$ iff $Ax_T \models \exists Q_1, \ldots, Q_i, \ldots \Sigma_p(\varphi, \psi)$.

Proof. In the proof we use the following equivalence: $Ax_T \models \exists Q_1, \ldots, Q_i, \ldots$ $\ldots, \Sigma_p(\varphi, \psi)$ iff every model of Ax_T can be extend so that $\langle \mathfrak{A}, [\varphi(\bar{x}) \wedge \bar{y} = \bar{x}]^{\mathfrak{A}},$ $Q_1^{\mathfrak{A}}, \ldots, Q_i^{\mathfrak{A}}, \ldots, [\psi(\bar{y}, \bar{x})]^{\mathfrak{A}} \rangle \models \Sigma_p$. (Here if $\chi(\bar{y})$ is a formula, then $[\chi(\bar{y})]^{\mathfrak{A}}$ is the relation on A of all vectors $\bar{a}$ satisfying $\chi(\bar{y})$.)

(i) Let us suppose that the program is partially correct in every model of Ax_T. By (i) of Theorem 1:

$$\langle \mathfrak{A}, [\varphi(\bar{x}) \wedge \bar{y} = \bar{x}]^{\mathfrak{A}}, Q_{0,1}^{\mathfrak{A}}, \ldots, Q_{0,i}^{\mathfrak{A}}, \ldots \rangle \models \Sigma_p$$

Since the program is partially correct w.r.t. φ and ψ: $Q_{0,j}^{\mathfrak{A}} \subseteq [\psi(\bar{y}, \bar{x})]^{\mathfrak{A}}$ for every $j \in I_p \setminus I'_p$. By Proposition 2:

$$\langle \mathfrak{A}, [\varphi(\bar{x}) \wedge \bar{y} = \bar{x}]^{\mathfrak{A}}, Q_{0,1}^{\mathfrak{A}}, \ldots, Q_{0,i}^{\mathfrak{A}}, \ldots, [\psi(\bar{y}, \bar{x})]^{\mathfrak{A}} \rangle \models \Sigma_p$$

So we have found appropriate family of relations to extend any model of Ax_T.

(ii) Let us suppose that there are $Q_1^{\mathfrak{A}}, \ldots, Q_i^{\mathfrak{A}}, \ldots$ satisfying Σ_p:

$$\langle \mathfrak{A}, [\varphi(\bar{x}) \wedge \bar{y} = \bar{x}]^{\mathfrak{A}}, Q_1^{\mathfrak{A}}, \ldots, Q_i^{\mathfrak{A}}, \ldots, [\psi(\bar{y}, \bar{x})]^{\mathfrak{A}} \rangle \models \Sigma_p$$

By (ii) of Theorem 1 for every $j \in I_p$, $Q_{0,j}^{\mathfrak{A}} \subseteq Q_j^{\mathfrak{A}}$. Thus for all $j \in I_p \setminus I'_p$, $Q_{0,j}^{\mathfrak{A}} \subseteq$ $\subseteq [\psi(\bar{y}, \bar{x})]^{\mathfrak{A}}$, that is the program is partially correct. □

Notice that the theorem could formalize the notion of partial correctness only by a second order formula ($Q_1, \ldots, Q_i, \ldots$ stay here for relational variable symbols). So this theorem failed to give a complete calculus to prove partial correctness. The question has arised whether it is possible to give any. Before giving an answer let us analyse the question itself, that is the notion of completeness. For a given type t and class of intended models M_p we say that for the programming language $\mathbf{P}_t = \langle P_t, M_p, k\rangle$ there is a complete calculus to prove partial correctnes, if we have a calculus which proves (φ, p, ψ) iff p is partially correct w.r.t. φ, ψ in every model $\mathfrak{A} \in M_p$. So the question can be raised only with respect to the similarity type and the model class of the language. We give some propositions and theorems dealing with different cases.

It is evident that if t and M_p are such that the second order language $\langle L_t^2, M_p, \models\rangle$ has complete calculus, then for $\langle P_t, M_p, k\rangle$ there is complete calculus to prove (φ, p, ψ).

In the following we give some negative results. The first of them is concerned with program schemes, and is based on the well known theorem that there are no complete calculus to prove that a program does not terminate for any input in any model (see e.g. [4] p. 264 theorem 4—2). Since non-terminating can be expressed by partial correctness using unsatisfiable formula as output condition, the following proposition stands:

Proposition 3. Let t be a type containing denumerable infinitely many function and relation symbols for every arity, and $M_p = M_L$, then there are no complete calculus for $\langle P_t, M_L, k\rangle$ to prove (φ, p, ψ). □

The following two theorems are our main ones. We select the type of arithmetic as the type of the programming language. The negative result for this case shows that in the practically important cases we have no complete calculus.

Theorem 4. Let t be the type of arithmetic, and $\mathfrak{N}$ is the standard model of arithmetic. There are no complete calculus for $\langle P_t, \{\mathfrak{N}\}, k\rangle$ to prove (φ, p, ψ).

Proof. We use the result that the problem of existence of solution for Diophantine equations is undecidable (see e.g. [9]). For the solution of each Diophantine equation $\tau_1(\bar{x}) = \tau_2(\bar{x})$ we write a program $P_{\langle\tau_1, \tau_2\rangle}$:

```
0: $y_1 \leftarrow 0$;
1: $y_2 \leftarrow 0$;
   ⋮
$n-1$: $y_n \leftarrow 0$;
$n$: IF $\tau_1(\bar{y}) = \tau_2(\bar{y})$ THEN $m+1$;
   ⋮
$m$: IF $y_1 = y_1$ THEN $n$;
```

where the command between the ones labeled by n and m compute the lexicographical successor of $\bar{y}$.

The execution of these programs gives a complete calculus for the problem whether a Diophantine equation has solution. If we had a complete calculus to prove partial correctness, it would give an algorithm to enumerate the Diophantine equations having no solution. So the problem of Diophantine equation would be decidable. Therefore there are no complete calculus to prove partial correctness of programs interpreted in the standard model of arithmetic. □

Theorem 5. Let t be the type of arithmetic, and PA be the Peano axiom system. There are no complete calculus for $\langle P_t, \text{Mod}(PA), k\rangle$ to prove (φ, p, ψ).

Proof. Let us notice that the traces of program $P_{\langle\tau_1,\tau_2\rangle}$ defined above, in any model of PA will be the same as the one in the standard model. (It is due to the fact that input values does not effect the execution.) So the same argument is applicable as in the proof of Theorem 4. □

Notice that the negative result is not due to the choice of first order language for **L**. Theorem 4 shows the impossibility of complete calculus for any language having the model class $\{\mathfrak{N}\}$.

Similar proofs can be created using any undecidable problem.

In the following part of our paper we analyse the Floyd—Hoare method. First we define a calculus equivalent to this method.

Definition 10. Let Ax_T a decidable axiom system, $p = l_0\colon U_0; \dots; l_n\colon U_n; \in P_t$, $\varphi, \psi \in L_t^1$

A *Floyd—Hoare derivation* of (φ, p, ψ) consist of:

a. A mapping $\Phi\colon I_p \to L_t^1$ such that

(i) $\Phi(l_0) = \varphi(\bar{x}) \wedge \bar{y} = \bar{x}$,

(ii) $\Phi(l_j) = \psi(\bar{y}, \bar{x})$ if $l_j \in I_p \setminus I_p'$.

b. First order derivations listed below:

(i) To each labelled command $l_m\colon y_k \leftarrow f(\bar{y})$ occuring in p a derivation of the form

$$Ax_T \vdash \Phi(l_m) \to \Phi(l_{m+1})[y_k/f(\bar{y})]$$

is assigned ($[y_k/f(\bar{y})]$ means that each free occurence of y_k is substituted by $f(\bar{y})$ in a collapsion free way).

(ii) To each labelled command l_m: IF $\varrho(\bar{y})$ THEN l_n occuring in p two derivations of the form

$$Ax_T \vdash \Phi(l_m) \wedge \varrho(\bar{y}) \to \Phi(l_n),$$

$$Ax_T \vdash \Phi(l_m) \wedge \neg\varrho(\bar{y}) \to \Phi(l_{m+1})$$

are corresponded.

Our notation for Floyd—Hoare derivability is: $Ax_T \underset{\text{F.H.}}{\vdash} (\varphi, p, \psi)$. □

Note that the definition of the calculus is in accordance with (iv) of Section 2.

Theorem 6. The Floyd—Hoare calculus is sound, that is if $Ax_T \underset{\text{F.H.}}{\vdash} (\varphi, p, \psi)$ then the p program is partially correct w.r.t. φ, ψ in every model of Ax_T.

Proof. Let us notice that the first order formulas whose derivation is required in Definition 10 are the axioms for the appropriate command as defined in Definition 2 substituting $\Phi(l_j)$ for Q_j. Having a Floyd—Hoare derivation, we have relations $[\Phi(l_j)]^{\mathfrak{A}}$ for every $\mathfrak{A} \in \text{Mod}(Ax_T)$ so that:

$$\langle \mathfrak{A}, \langle [\Phi(l_j)]^{\mathfrak{A}}_{l_j \in I_p} \rangle \models \Sigma_p$$

Therefore by Theorem 3 the program p is partially correct w.r.t. φ, ψ in every model of Mod (Ax_T). □

By Theorem 5 the following proposition is evident:

Proposition 4. If the similarity type includes the type of arithmetic and Ax_T is a recursive expansion of Peano axioms, the Floyd—Hoare calculus is not complete. □

We emphasize this last proposition because claim can be found in the literature that the Floyd—Hoare method is complete (see e.g. [4] p. 237 Prob. 3—19, [5], [6]). Now we analyse what causes its incompleteness and which points are neglected by those who claims completeness.

Ax_T is recursive, so $\langle L_t^1, \mathrm{Mod}\,(Ax_T), \models\rangle$ has complete calculus. This fact shows that if we have Φ so that relations $[\Phi(l_j)]^{\mathfrak{A}}$ satisfy Σ_p, then (φ, p, ψ) can be proved.

So the Floyd—Hoare method would be complete if for every $l_j \in I_p' \setminus \{l_0\}$ some of the relations $Q_j^{\mathfrak{A}}$ satisfying Σ_p could have been defined by first order formulas. Since the minimal relations satisfy Σ_p, the following stands:

Proposition 5. If the programming language is in the form $\langle P_t, \mathrm{Mod}\,(Ax_T), k\rangle$ where t includes arithmetic and Ax_T is a recursive expansion of Peano axioms, then the volumes of traces (the minimal relations) can not be defined. □

This is the point, where the refered publications fail to prove completeness in spite of their claim. They prove the existence of relations statisfying Σ_p, refering to the minimal relations. (So they prove theorems equivalent to our Theorem 3.) Some of them (e.g. [6]) neglect the question whether these relations can be expressed by first order formulas. J. W. de'Bakker in [5] introduces a language speaking about relations and using this language constructs the minimal relations for any given program.

However his construction can not be transformed to first order language, only to infinitary one permitting infinite "or". So he proved that the minimal relations can be defined by infinitary logic, but such logic has no complete calculus.

Independently from us M. Wand proved Proposition 5 in [10] for a type not including arithmetic.

Finally we investigate the traditional case — interpreting programs in one specific model. We discuss the case $M_p = \{\mathfrak{N}\}$, $\mathfrak{N}$ is the standard model of arithmetic. From Theorem 4 we known that Floyd—Hoare calculus can not be complete neither for this case. It is well know that in the standard model of arithmetic every recursive function can be represented (see e.g. [11] chapter 6), so:

Proposition 6. In the standard model of arithmetic the minimal relations can be defined. □

Proposition 6 is important because it can show the nature of incompleteness of Floyd—Hoare system to prove partial correctness — this is the same as the incompleteness of any first order calculus to prove theorems of arithmetic. Indeed, Theorem 4 can be veiwed as a version of the Gödel incompleteness theorem for first order logic extended with formulas (φ, p, ψ). Theorem 4 and Proposition 6 jointly say that for programming language $\langle P_t, \{\mathfrak{N}\}, k\rangle$ there are first order formula expressing (φ, p, ψ), but we have no universal algorithm to enumerate the axioms usable in its proof. This fact shows that the incompleteness theorems for proving partial correctness does not prevent us from proving partial correctness as the Gödel incompleteness theorem does not prevent mathematicians proving theorems of arithmetic. It is true that fully automatized algorithm to prove partial correctness in every case can not exist, but with human intuition every program can be proved. Speaking about the mechanisation of program verification this argument underlies the necessity of interactive systems.

Abstract

First the paper shows generally the way how languages of mathematical logic can be used to describe semantics of programming languages and to prove theorem about programs. It is worked out for the case of first order classical logic, emphasis is laid on the model theoretical point of view. Provability of partial correctness is investigated. We show that if the programming language includes arithmetic, there are no complete calculus to prove partial correctness. The method of inductive assertions is discussed, and we analyse why several publication claimed its completeness.

RESEARCH INSTITUT FOR APPLIED
COMPUTER SCIENCE
H—1536 BUDAPEST, HUNGARY
P. O. BOX 227.

References

[1] FLOYD, R. W., Assigning meanings to programs, *Proceeding of Symposium on Applied Mathemathics,* 19, 1967.
[2] HOARE, C. A. R., An axiomatic basis for computer programming, *Comm. ACM,* v. 12, No. 10, 1969.
[3] MANNA, Z., The correctness of programs, *J. Comput. System Sci.,* v. 3, No. 2, 1969.
[4] MANNA, Z., *Mathematical theory of computation,* McGraw-Hill, 1974.
[5] BAKKER, J. W. de and L. G. L. T. MEERTENS, On the completeness of the inductive assertions method, *J. Comput. System Sci.* ,v. 11, No. 3, 1975.
[6] EMDEN, M. H. VAN, Verification conditions as programs, *Automata, Languages and Programming,* Third International Colloquium at the University of Edinburg. ed. by S. Michaelson and R. Miner Edinburgh University Press, 1976.
[7] ANDRÉKA, H., T. GERGELY and I. NÉMETI, Easily comprehensible mathematical logic and it's model theory, KFKI, No. 24, 1975.
[8] CHANG, C. C. and H. J. KEISLER, *Model theory,* North Holland Publishing Co., 1973.
[9] DAVIS, M., Hilbert's tenth problem is unsolvable, *Amer. Math. Monthly,* March 1973.
[10] WAND, M., A new incompleteness result for Hoare's system, *J. Assoc. Comput. Mach.,* v. 25, No. 1, 1978.
[11] SCHOENFIELD, J. R., *Mathematical logic,* Reading, Addison—Wesley, 1967.

(Received May 29, 1978)

A simple shading for computer displayed surfaces

By G. T. HERMAN and H. K. LIU

To the memory of Professor László Kalmár

Introduction

In order to achieve a visually realistic representation of the surface of a three-dimensional object on a cathode-ray tube, one has to solve the problems of removing the hidden parts of the surface and shading its visible parts. A commonly used approach which allows efficient solutions to both these problems is to approximate the surface so that it is composed of planar polygons [2]. Lack of continuity in the shading across polygonal boundaries causes undesirable artifacts which can be removed by continuous shading procedures such as the one suggested by Gouraud [2].

In this note we discuss an extremely simple and efficient alternative method for removing the artifact mentioned above. In the application area which is our main concern our method is not only two orders of magnitude faster than that of Gouraud's, but it also produces superior displays. In this application area the surface is approximated by a composition of squares, each one of which is parallel to one of three mutually perpendicular planes. In the next section we describe the nature of our application area.

Three-dimensional reconstruction from projections

The problem of reconstructing a three-dimensional object from a set of its two-dimensional projected images has arisen in fields ranging from electron microscopy to holographic interferometry. For example, in medicine we use an x-ray source to project the body onto the two-dimensional surface of a film. From a subset of all possible projections of a body, reconstruction algorithms [1] produce a three-dimensional array of numbers in which each number is an estimate of the average density of the body in one of a set of equal, non-overlapping cubes, called voxels (volume elements).

In order to see the shape of a particular organ, a three-dimensional boundary detection algorithm has been devised [3]. This detects the organ's boundary surface and generates the description of the surface in terms of the square faces of the voxels.

An example

For a demonstration of our ideas we use a plastic cast of an isolated canine left ventricle with some beads inserted on the surface. Figure 1 shows a television image of this cast, blurred so as to make the resolution similar to that obtained in the reconstruction process.

A variant of the Algebraic Reconstruction Technique [1] has produced from 34 x-ray projections of our cast a $64\times64\times28$ density array, with the edge of each voxel being $1.04\times1.04\times1.68$ mm. (Based on such a reconstruction variations in the surface smaller than a voxel could not possibly be displayed. For a fair evaluation of the efficacy of our display procedure its output should be compared to an image of the original which has been blurred as in Figure 1 to remove features

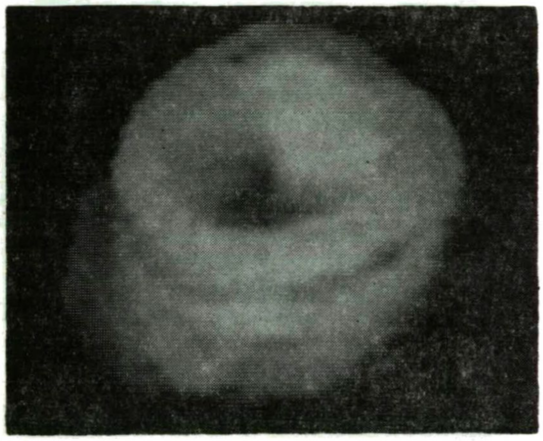


Fig. 1
A television image of the plastic cast of a canine left ventricle

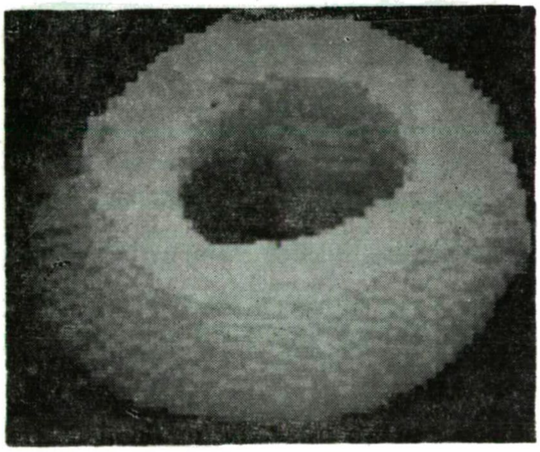


Fig. 2
Computer-generated display of the cast which was determined from 28 reconstructed cross-sectional levels of the cast

smaller than the resolution of the reconstruction procedure.) A surface detection algorithm [3] has been applied to the reconstructed array and it produced an estimated surface of the cast consisting of a connected collection of faces of voxels. This collection is displayed in Figure 2 using a shading which is uniform within each face and whose value depends on the angle the face makes with an assumed direction of light and on the distance of the face from an assumed light source (Warnock's shading rule as given on page 624 of [2]). As can be seen, approximating the curved surfaces with small square surfaces generates an undesirable visual effect, caused mainly by having only three directions in which surface elements lie. Similar, though less disturbing artifacts are observable whenever a curved surface is displayed as a collection of planar polygons.

Gouraud's shading procedure

Gouraud [2] suggested a way to get rid of the artifact apparent in Figure 2. This approach is to modify the computation of the shading on each surface so that continuity exists across surface boundaries. This continuity can be achieved by assigning as normal at a vertex the average of the normals to each surface associated with this particular vertex. Each surface has a different shading for each of its vertices and the shading at any particular point inside the surface has to be computed as a continuous function of the shading at the vertices of the surface.

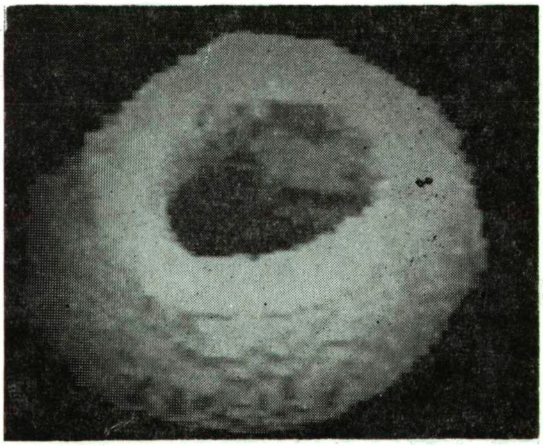


Fig. 3
Computer-generated display of the cast by use of Gouraud's shading procedure

Due to the large number of the surfaces in the objects we are interested in (typically 10,000), this approach is prohibitively time consuming. Also, the results are far from acceptable, because the surfaces are either parallel or perpendicular to each other. Figure 3 shows a display produced by Gouraud's method applied to the surface of the cast shown in Figure 2.

A simple smoothing procedure

In this section, we describe an alternative approach to this problem. Instead of smoothing the surfaces in object-space, we smooth the intensities in image-space.

Let $G_1, \ldots, G_9$ denote the shading intensities in a screen dot and its eight neighbors before smoothing, as indicated by the diagram below.

G_6	G_2	G_7
G_3	G_1	G_4
G_8	G_5	G_9

After smoothing, the new intensity of the center dot will be

$$G_1^{\text{new}} = \frac{G_1 + W * \sum_{i=2}^{5} f_i * G_i + W^2 * \sum_{i=6}^{9} f_i * G_i}{1 + W * \sum_{i=2}^{5} f_i + W^2 * \sum_{i=6}^{9} f_i}.$$

If the dot labelled by 1 is on an edge and does not have a neighbor labelled by i, then f_i is assumed to be zero, otherwise f_i equals one. W is a user adjustable weighting factor.

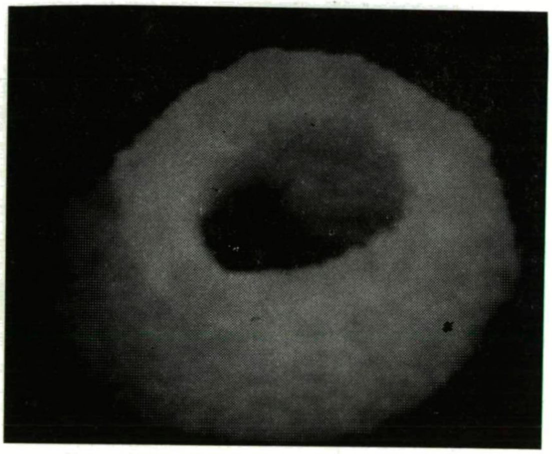


Fig. 4
Computer-generated display of the cast by the use of the proposed shading procedure

This simple smoothing method gives surprisingly good results. Figure 4 shows the display shown in Figure 2 after our smoothing procedure with $w=0.8$. The time required for this smoothing procedure is under eight seconds (CDC 3500) while our implementation of Gouraud's method needed 985 seconds to produce the smoothing shown in Figure 3. (Note that the time required by our procedure is dependent only on the number of picture elements in the display, while the time required by Gouraud's method depends on the number of polygons from which the surface is made up.) Also, in our opinion, the true image (Figure 1) is better approximated by the display produced by our method (Figure 4) then it is by the display produced by Gouraud's method (Figure 3).

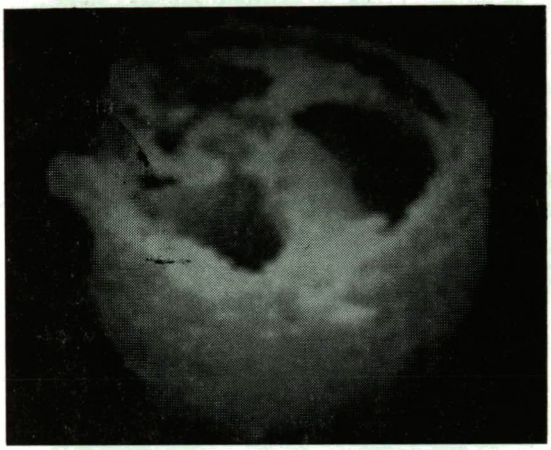


Fig. 5
Computer-generated display of a canine heart which was determined from 30 reconstructed cross-sectional levels of the heart

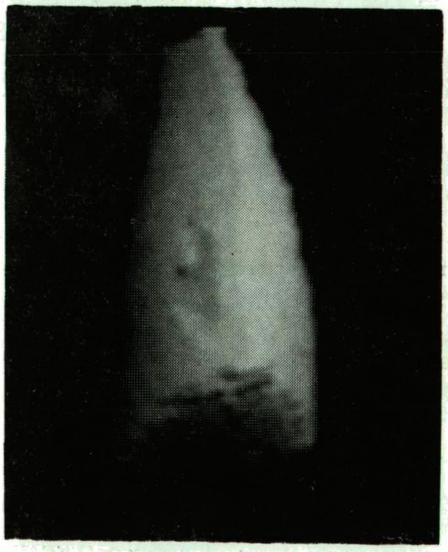


Fig. 6
Computer-generated display of a part of a canine left lung which was determined from 64 reconstructed cross-sectional levels of the intact thorax of a dead dog

Further examples

We demonstrate our display method on reconstructions of two further biological objects.

In Figure 5, we display the surface of an isolated canine heart whose $64 \times 64 \times 30$ reconstruction has been produced from 50 x-ray projections.

In Figure 6, we display the surface of a part of the left lung of a dog. Thirty-five x-ray projections of an intact dead dog have been taken and the thorax and contents of the dog were reconstructed as a $64 \times 64 \times 64$ array. The input to the surface detection algorithm was a $28 \times 64 \times 64$ subarray containing most but not the whole of the left lung. The surface detection algorithm [3] was applied to detect the surface of the lung in the thorax and the result is displayed in Figure 6. The imprints in the lung of the heart and of the major airway above it are clearly visible.

Conclusion

We have proposed a display smoothing algorithm which is very efficient if the surface to be displayed consists of many polygonal surface elements. Efficiency is due to the smoothing being done on the image and not on the surface. In the display of organ surfaces obtained by three-dimensional reconstruction, our algorithm produces results visually superior to Gouraud's smoothing procedure in a small fraction of the time required by that procedure.

Acknowledgements. The authors wish to thank Dr. James F. Greenleaf for valuable discussions and Drs. Lowell D. Harris, Erik L. Ritman and Richard A. Robb for providing test data. The authors are indebted to Ms. Jean Frank and her coworkers for secretarial and typing assistance.

The research of the authors is supported by the United States Public Health Service, under grants HL 18968, HL 4664 and RR7.

Abstract

This note proposeses a simple way of removing the artifact caused by approximating curved surfaces with polygons in computer-generated three-dimensional display. The method is compared with Gouraud's continuous shading method.

DEPT. OF COMPUTER SCIENCE
STATE UNIVERSITY OF NEW YORK
AT BUFFALO
4226 RIDGE LEA ROAD
AMHERST, NEW YORK 14226

References

[1] Gordon, R., G. T. Herman and S. A. Johnson, Image reconstruction from projections, *Scientific American,* v. 233, 1975, pp. 56—68.
[2] Gouraud, H., Continuous shading of curved surfaces, *IEEE Trans. Computers,* v. C—20, 1971, pp. 623—629.
[3] Liu, H. K., Two- and three-dimensional boundary detection, *Computer Graphics and Image Processing,* v. 6, 1977, pp. 123—134.

(Received May 29, 1978)

Normal-form transformations of context-free grammars

By G. Hotz

To the memory of Professor László Kalmár

Introduction

Each context-free grammar G can be transformed into a Chomsky-normalform (CNF) and into a Greibach-normalform (GNF) without changing the languages generated by the grammars. Our interest does not concern the invariance of the languages under such transformations but the ambiguity of the grammars, the multiplicity of words relative to the grammars and relations between pairs of grammars. Syntactical transformations of languages are induced by the grammars. Therefore, it should be of interest, if certain syntactical transformations between languages transform in a natural manner with the normal form transformations. The role of monoid homomorphisms in connection with rational transformation is played by functors between the syntactical categories of grammars in connection with tree transformations.

In this paper we define three different transformations τ_1, τ_2 and τ_3 of grammars in CNF into GNF. τ_1 produces productions with one terminal and at most two non-terminals in the range of the productions. τ_2 and τ_3 generate productions p with maximally two resp. three non-terminals and one terminal on each side of the range (p).

τ_1 has been considered for the first time in a technical report 1967 by S. Greibach. One finds it again in [GR] (1975). Implicitely the construction is contained in [Ho 2] (1974) too. τ_2 and τ_3 seem to be studied here the first time.

Geller, Harrison and Havel showed in [GE—HA], that for each $LR(k)$ language there exist a $LR(k')$ grammar in GNF with $k'=k$ for $k\geqq 1$ and that there exist $LR(0)$ languages for which one has always $k'\geqq 1$. But they did not use the simple transformation τ_1.

We show that τ_1, τ_2 and τ_3 preserve unambiguity and do not increase multiplicities. But there exist grammars for which the multiplicity decreases. Non $LR(k)$ grammars may be transformed into $LR(k)$ grammars.

We show that functors between the syntactical categories of the grammars G_1 and G_2 are transformed into functors between the syntactical categories between the grammars $\tau_1(G_1)$ and $\tau_1(G_2)$.

With the same methods we show in a following paper, that τ_1 preserves $LL(k)$ for all k and $LR(k)$ for $k \geqq 1$ and that $LR(0)$ is transformed into $LR(1)$. The proofs for both properties are nearly identical. From this paper we use the unambiguity lemma for the existence of well formed decompositions of morphisms (classes of derivations) in products of $(t, 1)$-prime derivations. τ_2 and τ_3 may destroy the LR and LL properties. This means that transformations inverse to τ_2 and τ_3 may eventually transform non $LR(k)$ grammars into such grammars.

Because until now we do not know much about transformations which transform certain grammars of $LR(k)$ languages into $LR(k)$ grammars the relations τ_2^{-1}, τ_3^{-1} may be of interest.

For certain transformations from general context-free grammars into CNF the LR-invariance has been showed by [Be] (1976) and [Sch] (1973).

We use the notation of x-categories or syntactical categories as defined in [Ho—Cl]. An introduction in related questions the reader may find in [A—Ull] or [Sa].

Definitions and preliminaries

In the following T is the terminal and Z the variable alphabet, and S is the axiom of the context-free grammar G. We assume that the set P of productions of G is in Chomsky normal form. This means that for $f \in P$ we have

$$f = (z, z_1 z_2) \quad \text{or} \quad f = (z, t),$$

where, as always in this paper, z, z_1, z_2 are in Z and t in T. We assign to G the free x-category $F(G)$; that means that we wish to calculate with derivations of G, or — more precisely formulated — we wish to calculate with the classes of inessentially different derivations of G. We write

$$w \xrightarrow{f} u \quad \text{and} \quad D(f) = w, \quad C(f) = u$$

if f is a derivation class from w to u. w is the domain and u the codomain of f.

From

$$w \xrightarrow{f} u \quad \text{and} \quad w' \xrightarrow{g} u'$$

we from

$$ww' \xrightarrow{f \times g} uu',$$

the class of derivations we get from f and g by doing the derivations f and g in parallel.

We form

$$\begin{tikzcd} u \arrow[r, "f"] \arrow[dr, dashed, "h"'] & v \arrow[d, "g"] \\ & w \end{tikzcd}$$

$$h = g \circ f$$

by executing first f and then g if $C(f)=D(g)$.

For $F(G)$ we also write $F(P)$ where P is the production set of G, and we write

$$w \xrightarrow[P]{} v$$

if there exists $f \in F(P)$ such that

$$w \xrightarrow{f} v$$

holds.

Now we study as in [Ho 2] special derivations which are related to canonical derivations of words uw with $u \in T^*$ and $w \in Z^*$. These special derivations will be used to construct the productions in our normal form grammars.

Definition. A derivation f in Chomsky normalform

$$z \xrightarrow{f} uwv,\ u \in T^*,\ v \in T^*,\ w \in Z^*$$

is called (u, v)*-prime* if from

$$f = (1_u \times g \times 1_v) \circ h$$

it follows that $g = 1_w$.

This means that f is (u, v)-prime if f is a shortest derivation which generates from z a word which begins with the terminal symbols u and ends with the terminal symbols v and has only nonterminal symbols (possibly none) between.

As we will see later, of special interest are the cases

1. $u=1,\ v \in T$,
2. $u \in T,\ v=1$,
3. $u \in T,\ v \in T$.

Let

$$B(z, u, v) = \{w \in Z^* \mid \text{ there exists } z \xrightarrow{f} uwv,\ f\ (u, v)\text{-prime}\}.$$

In [Ho 2] we showed that $B(z, u, 1)$ is a regular set for all $u \in T^*$. By symmetry arguments it follows that $B(z, 1, v)$, too, is a regular set for $v \in T^*$.

For f (u, v)-prime $u, v \in T$ we have a decomposition

$$f = (1_u \times 1_w \times g) \circ h, \quad w \in Z^*$$

such that h is $(u, 1)$-prime and g is $(1, v)$-prime.

On the other hand f is (u, v)-prime for all $(u, 1)$-prime h and $(1, v)$-prime g.

We define for $L \subset Z^*$ and $x \in Z^*$

$$L_x = \{w \in Z^* \mid wx \in L\}$$

and

$$_xL = \{w \in Z^* \mid xw \in L\}.$$

With this notation we have

$$B(z, t, r) = \bigcup_{y \in Z} [B(z, t, 1)]_y B(y, 1, r) \quad \text{for} \quad t, r \in T.$$

Now, the relation $w_1 \equiv w_2 \Leftrightarrow L_{w_1} = L_{w_2}$ is the well known syntactical congruence (i.e., left invariant equivalence relation). For regular sets L there are only a finite number of these congruence classes where each class is also a regular set. From this we conclude that $[B(z, t, 1)]_y$ is a regular set and thus that $B(z, t, r)$ is regular for all $t, r \in T \cup \{1\}$.

Lemma 1. The set

$$B = \{{}_x[B(z, t, r)]_y | z \in Z,\ x \in Z^*,\ y \in Z^*\}$$

is a finite set of regular sets for all $t, r \in T \cup \{1\}$.

Proof. We know from the above discussion that $B(z, t, r)$ is regular. We know also that

$$A = \{[B(z, t, r)]_y | y \in Z^*,\ z \in Z\}$$

is a finite set of regular sets.

Now as one sees immediately

$$[{}_pL_q]_s = {}_pL_{sq},\quad {}_s[{}_pL_q] = {}_{ps}L_q.$$

Therefore we conclude from the finiteness of the set A, that B is also finite and from the regularity of the elements of A, that the elements of B are regular. This finishes our proof.

Lemma 2. For

$$u, v \in T^*,\quad z \in Z \quad \text{and} \quad t, r \in T$$

it follows that

$$B(z, ut, rv) = \bigcup_{x, y \in Z} B(x, t, 1)_x[B(z, u, v)]_y B(y, 1, r)$$

$$\cup \bigcup_{x \in B(z, u, v) \cap Z} B(x, t, r).$$

$B(z, ut, rv)$, then, is regular.

Proof. Let f be a (ut, rv)-prime derivation with $D(f) = z$. Then f can be decomposed into

$$f = (1_u \times g \times 1_v) \circ h$$

such that h is (u, v)-prime.

Now we discuss the two cases corresponding to $D(g) \in Z^m$ for $m \geqq 2$ and $D(g) \in Z$; these are the only two possibilities, since G is in Chomsky normal form.

1. For this case g can be decomposed into

$$g = g_1 \times 1_w \times g_2.$$

Here g_1 is $(t, 1)$-prime and g_2 is $(1, r)$-prime. Otherwise f would not be (ut, rv)-prime. Let $D(g_1) = x$, $D(g_2) = y$. Then we have

$$C(h) = uxwyv, \quad \text{where} \quad xwy \in B(z, u, v).$$

Further, let $C(g_1) = tw_1$ and $C(g_2) = w_2 r$. Then we have $w_1 \in B(x, t, 1)$, $w_2 \in B(y, 1, r)$ and for $C(f) = ut\tilde{w}rv$ the following holds:

$$\tilde{w} = w_1 w w_2 \in B(x, t, 1) \cdot {}_x[B(z, u, v)]_y \cdot B(y, 1, r).$$

2. For this case g can be decomposed into

$$g = (1_t \times 1_{w_1} \times g_2) \circ g_1$$

with $g_1(t, 1)$-prime and $g_2(1, r)$-prime. Then for $C(g_1)=tw_1y$, $C(g_2)=w_2r$, $C(f)=$ $=utwrv$ and $C(h)=uxv$ we have for $y \in Z$

$$w = w_1 \cdot w_2 \in [B(x, t, 1)]_y \cdot B(y, 1, r) \subset B(x, t, r).$$

From case 1 and case 2 we conclude that the left part of the equation in our lemma is contained in the right part. This completes the proof in one direction.

The inclusion in the other direction follows directly from the following facts. For $g_1(t, 1)$-prime and $g_2(1, r)$-prime and $h(u, v)$-prime, then if the product

$$f = (1_u \times g_1 \times 1_w \times g_2 \times 1_v) \circ h$$

is defined where $w \in Z^*$, f is (ut, rv)-prime. This means that each $B(x, t, 1)$ $[B(z, u, v)]_y \cdot$ $B(y, 1, r)$ is contained in $B(z, ut, rv)$. For $x \in B(z, u, v)$ it is clear that $B(x, t, r) \subset$ $\subset B(z, ut, rv)$. This completes the proof of the lemma.

This lemma nearly gives us a recursive equation for calculating the sets $B(z, u, v)$. The importance of these sets follows from the obvious

Theorem 1.

$$w = u \cdot v \in L \Leftrightarrow 1 \in B(S, u, v).$$

This means that the word problem $w \in L$ can be reduced to the problem of whether 1 is in a regular set. We are here not interested in developing this direction further, however. For our purposes of constructing a normal form grammar we do not need a complete recursive definition of the $B(z, u, v)$.

To construct the productions of our normal form grammars we will use the (u, v)-prime derivations of the free x-category $F(G)$ for the special case that $u, v \in T$ rather than T^*. If, for example, $S \xrightarrow{f} uwv$ is such a (u, v)-prime derivation, we could include a production of the form $S \to uRv$ in one of our normal form production systems, $\tilde{P}$, where $R = B(S, u, v)$. Then for any such new variable R we would also have to introduce productions of the form

$$R \to t \cdot B(R, t, r) \cdot r$$

into $\tilde{P}$ representing the class of all (t, r)-prime derivations from the set $R \subset Z^+$ in P. Here $B(R, t, r)$ is a simple extension of the definition of $B(z, t, r)$:

$$B(R, t, r) = \{\tilde{w} | w \xrightarrow{f} t\tilde{w}r \text{ is } (t, r)\text{-prime and } w \in R\}.$$

To see that this process of constructing productions for $\tilde{P}$ can be continued with the $B(R, t, r)$ sets we give the following lemma which one can easily prove.

Lemma 3. For $t, r \in T$ and $R \subset Z^+$

$$B(R, t, r) = \underbrace{}_{x, y \in Z} B(x, t, 1)_x [R]_y B(y, 1, r)$$

$$\cup \underbrace{}_{x \in R \cap Z} B(x, t, r).$$

This lemma gives us a way to factor the set $B(R, t, r)$ into the regular sets we introduced earlier. Thus we are now able to generate all codomain sets of derivations $f \in F(G)$, where each such f is a product in "$\circ$" and "$\times$" of prime derivations, from the domain sets

$$_p[B(z, t, r)]_q, \quad t, r \in T \cup \{1\}, \quad t \cdot r \neq 1$$

where length (p) = length (q) for $p, q \in Z^*$. From Lemma 1 it follows that $_p[B(z, t, r)]_q$ is a finite set of regular sets. More precisely formulated, from Lemma 3 we can select the following classes of derivations from $F(P)$ for all $p, q \in Z^*$, length (p) = length (q) and $x, y \in Z$, $u, v \in T$, and $t, r \in T \cup \{1\}$, $t \cdot r \neq 1$:

$$S \to t \cdot B(S, t, r) \cdot r, \tag{P'1}$$

$$_p[B(z, t, r)]_q \to u \cdot B(x, u, 1) \cdot {}_{px}[B(z, t, r)]_{yq} \cdot B(y, 1, v) \cdot v, \tag{P'2}$$

$$_p[B(z, t, r)]_q \to u \cdot B(x, u, v) \cdot v, \quad x \in {}_p[B(z, t, r)]_q \cap Z. \tag{P'3}$$

Each of the classes $(P'1)$, $(P'2)$ and $(P'3)$ represents an entire set of derivations generated from the choices of p, q, r, t, u, v, w, x, and y. Clearly, many of these choices will lead to empty sets. However, it is evident that each of these classes is finite (since $B(z, t, r)$ is a regular set, the congruence relation established by $_p[B(z, t, r)]_q$ for all $p, q \in Z^*$ is of finite index). Therefore, we can use these derivations as the basis for constructing the productions $\tilde{P}$ of our normal form grammars. Before constructing such a normal form production system, however, we must convince ourselves that *every* derivation class in $F(P)$ can be decomposed as above.

Well formed decompositions of derivations

Now we consider normal forms of derivations for any context-free grammar G in Chomsky normal form using the x-categorical expressions which define derivations. We show that each class f of derivations has exactly one normal form derivation which we will call *well formed* (w.f.).

Definition. A decomposition $f = f_n \circ \ldots \circ f_1$ with $D(f) \in Z$ and

$$f_i = f_{i,1} \times \ldots \times f_{i,m_i} \quad \text{for} \quad i = 1, \ldots, n$$

is *well formed* if conditions $(W1)$, $(W2)$ and $(W3)$ hold.

See Fig. 1 for $(W1)$ and Fig. 2 for $(W2)$.

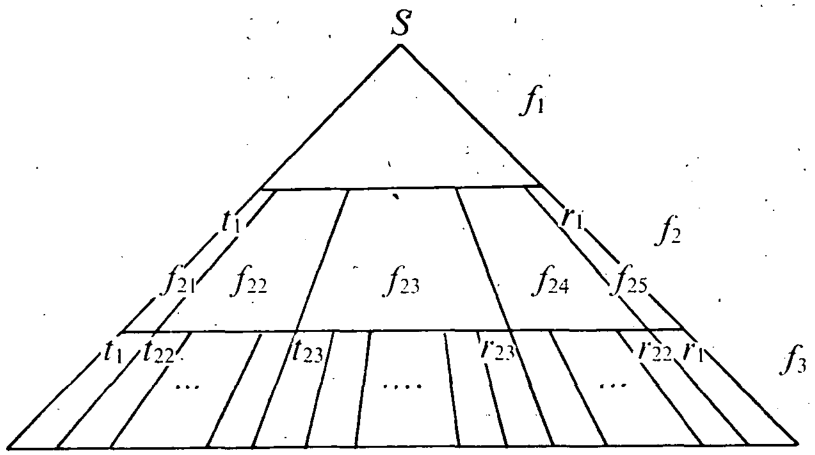
$(W1)$ $D(f_{i,l}) \in Z \cup T$.
If $D(f_{i,l}) = t \in T$ then $f_{i,l} = 1_t$.
If $D(f_{i,l}) \in Z$ then $f_{i,l}$ is (t, r)-prime with $t, r \in T \cup \{1\}$ and $t \cdot r \neq 1$.

$(W2)$ Let

$$f_{i+1} \circ f_i = F_1 \times \ldots \times F_{m_i}$$

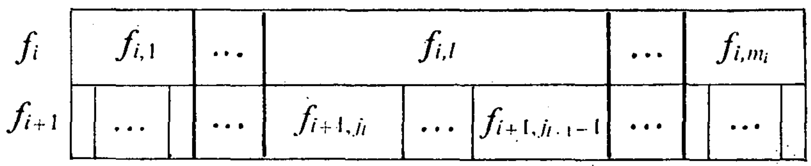
be the uniquely determined decomposition with $D(F_i) \in Z \cup T$ for $i = 1, \ldots, m_i$, and

$$F_l = (H_1 \times \ldots \times H_m \times H \times G_m \times \ldots \times G_1) \circ f_{i,l}$$

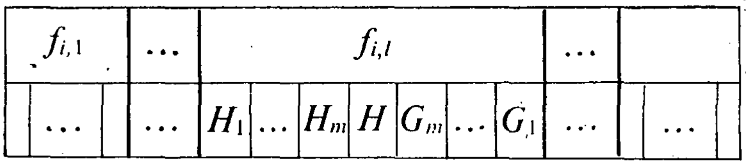


$f_{21} = 1_{t_1}$, $f_{25} = 1_{r_1}$,
f_{22} $(t_{22}, 1)$-prime, f_{24} $(1, r_{22})$-prime, f_{23} (t_{23}, r_{23})-prime

Fig. 1



to be short in indices we write for this



with $H=1$ for $j_{l+1}-j_l=o(2)$

Fig. 2

be the uniquely defined decomposition with $D(H_i)\in Z\cup T$, $D(G_i)\in Z\cup T$, $D(H)\in Z\cup \cup\{1\}$ and $f_{i,l}(t_1, r_1)$-prime. Then it follows that $H_1=1_{t_1}$, $G_1=1_{r_1}$ for $t_1, r_1\in T\cup\{1\}$, $t_1\cdot r_1\neq 1$ and

$$H_i \text{ is } (t_i, 1)\text{-prime}, \quad t_i\in T,$$

$$G_i \text{ is } (1, r_i)\text{-prime}, \quad r_i\in T$$

for $i=2, \ldots, m$, and $H=1$ for length $(C(f_{i,l}))$ even; for length $(C(f_{i,l}))$ odd $H\in P$ (the set of productions of the underlying grammar) with $C(H)\in T^+$, or H is (t_{m+1}, r_{m+1})-prime with $t_{m+1}, r_{m+1}\in T$.

$(W3)$ $f_1\in P$ and f_1 is terminal, or f_1 is (t, r)-prime with $t, r\in T$.

Lemma 4. Let $F(P)$ be the free x-category generated by the context-free production system P in Chomsky normal form. For each $f\in F(P)$ there exists exactly one (w.f.)-decomposition of f if $D(f)\in Z$ and $C(f)\in T^+$.

Proof. Let

$$z\xrightarrow{f} w, \quad z\in Z, \quad w\in T^+.$$

If $f\in P$, then f is terminal and f is a unique (w.f.)-decomposition of itself.

Now assume $f\notin P$. Then we can write $w=tw'r$ with $t, r\in T$ and w' uniquely determined by t and r. Therefore there exists a unique decomposition

$$f = (1_t\times h\times 1_r)\circ f_1$$

such that f_1 is (t, r)-prime. We decompose

$$h = H_2\times\ldots\times H_k\times H\times G_k\times\ldots\times G_2$$

such that $D(H_i)\in Z$, $D(G_i)\in Z$ and $H=1$ or $D(H)\in Z$ for $k\geqq 2$.

Again this decomposition is unique, if it is possible. If it is not possible to decompose h in this way then $h=1$ and $f=f_1$; that is, one has

$$f = f_1 = (g_1\times g_2)\circ g_3$$

with

$$z\xrightarrow{g_3} z_1z_2, \quad z_1\xrightarrow{g_1} t, \quad z_2\xrightarrow{g_2} r$$

productions in P, and our lemma holds in this case.

Now with h decomposed as shown,

$$z_i\xrightarrow{H_i} w_i, \quad y_i\xrightarrow{G_i} v_i, \quad w_i, v_i\in T^+$$

for $i=2, \ldots, k$ and

$$x\xrightarrow{H} u, \quad u\in T^+$$

for $H\neq 1$.

For $i=2, \ldots, k$ we can decompose the derivations uniquely as

$$H_i = (1_{t_i}\times h_i)\circ f_{2,i},$$

$$G_{k+2-i} = (g_i\times 1_{r_i})\circ f_{2,k+i}$$

in the case that $H=1$. Here $f_{2,i}$ is $(t_i, 1)$-prime and $f_{2,k+i}$ is $(1, r_i)$-prime. In the case $H\neq 1$ we have the same decomposition for $H_2, \dots, H_k$ but

$$H = (1_t \times h' \times 1_{r'}) \circ f_{2,k+1}, \quad t', r' \in T$$

and

$$G_{k+2-i} = (g_i \times 1_{r_i}) \circ f_{2,k+1+i}$$

for $i=2, \dots, k$, where $f_{2,k+1}$ is (t', r')-prime and $f_{2,k+1+i}$ is $(1, r_i)$-prime.

We now iterate this construction by applying it to the h_i, g_i and h' in the same way as we did to h, and so on.

After a finite number of steps we get the uniquely determined (w.f.)-decomposition of f.

The first normal form transformation

Using the result of lemma 4 we now derive a production system from the relations $(P'1)$, $(P'2)$ and $(P'3)$.

We write

$$B \to B'$$

for $B, B' \subset (Z \cup T)^*$ iff for each $w \in B'$ there exists a $u \in B$ such that $u \to w$ in the usual sense holds. It follows directly that

$$B \to \{u\}$$

for $u \in B$. For simplicity we identify u with $\{u\}$. Using this relation and the transitive closure property of derivations one has

$${}_p[B(z, t, r)]_q \to s$$

for $x \in {}_p[B(z, t, r)]_q$ and $x \to s$.

Let V' be an alphabet and v a mapping into V' which is defined on

$$U = \{(z, t, r, p, q) | z \in Z;\ t, r \in T \cup \{1\},\ t \cdot r \neq 1,\ p, q \in Z^*,\ \text{length}(p) = \text{length}(q)\}$$

such that

$$v(z, t, r, p, q) = v(z', t'\ r', p', q')$$

iff

$${}_p[B(z, t, r)]_q = {}_{p'}[B(z', t', r')]_{q'}.$$

From lemma 1 we know that such an alphabet V' and such a mapping v can be constructed effectively and that V' is finite. Let $\tilde{V} = V' \cup \{S\}$ be the nonterminal alphabet of our normal form. For $v(z, t, r, 1, 1)$ we write simply $v(z, t, r)$.

Using $(P', 1)$ we construct the productions $(\tilde{P}, 1)$ as

$(\tilde{P}, 1)$ $S \to u$ for $u \in T \cup T^2$ and $S \overset{*}{\to} u$.

$$S \to t \cdot v(S, t, r) \cdot r \quad \text{for} \quad B(S, t, r) \neq \emptyset.$$

We define $(\tilde{P}, 2)$ as follows.

$(\tilde{P}, 21)$ $\quad v(z, t, r, p\ q) \to u \cdot v(x, u, 1) \cdot v(z, t, r, px, yq) \cdot v(y, 1, w) \cdot w$

for $\quad B(x, u, 1) \cap Z^+ \neq \emptyset, \quad B(y, 1, w) \cap Z^+ \neq \emptyset, \quad {}_{px}[B(z, t, r)]_{yv} \cap Z^+ \neq \emptyset,$

$(\tilde{P}, 22)$ $v(z, t, r, p, q) \to u \cdot v(x, u, 1) \cdot v(y, 1, w) \cdot w$

for $1 \in {}_{px}[B(z, t, r)]_{yq}$, $B(x, u, 1) \cap Z^+ \neq \emptyset$, $B(y, 1, w) \cap Z^+ \neq \emptyset$,

$(\tilde{P}, 23)$ $v(z, t, r, p, q) \to u \cdot v(z, t, r, px, yq) \cdot v(y, 1, w) \cdot w$

for $1 \in B(x, u, 1)$, ${}_{px}[B(z, t, r)]_{yq} \cap Z^+ \neq \emptyset$, $B(y, 1, w) \cap Z^+ \neq \emptyset$,

$(\tilde{P}, 24)$ $v(z, t, r, p, q) \to u \cdot v(x, u, 1) \cdot v(z, t, r, px, yq) \cdot w$

for $1 \in B(y, 1, w)$, $B(x, u, 1) \cap Z^+ \neq \emptyset$, ${}_{px}[B(z, t\ r)]_{yq} \cap Z^+ \neq \emptyset$,

$(\tilde{P}, 25)$ $v(z, t, r, p, q) \to u \cdot v(x, u, 1) \cdot w$

for $1 \in {}_{px}[B(z, t, r)]_{yq} \cdot B(y, 1, w)$, $B(x, u, 1) \cap Z^+ \neq \emptyset$,

$(\tilde{P}, 26)$ $v(z, t, r, p, q) \to u \cdot v(z, t, r, px, yq) \cdot w$

for $1 \in B(x, u, 1) \cdot B(y, 1, w)$, ${}_{px}[B(z, t, r)]_{yq} \cap Z^+ \neq \emptyset$,

$(\tilde{P}, 27)$ $v(z, t, r, u, v) \to u \cdot v(y, 1, w) \cdot w$

for $1 \in B(x, u, 1) \cdot {}_{px}[B(z, t, r)]_{yq}$, $B(y, 1, w) \cap Z^+ \neq \emptyset$,

$(\tilde{P}, 28)$ $v(z, t, r, p, q) \to u \cdot w$

for $1 \in B(x, u, 1) \cdot {}_{px}[B(z, t, r)]_{yq} \cdot B(y, 1, w)$.

We set

$$(\tilde{P}, 2) = \bigcup_{i=1}^{8} (\tilde{P}, 2i).$$

We now define the productions $(\tilde{P}, 3)$.

$(\tilde{P}, 3)$ $v(z, t, r, p, q) \to u \cdot v(x, u, w) \cdot w$

for $x \in {}_p[B(z, t, r)]_q \cap Z$ and $B(x, u, w) \cap Z^+ \neq \emptyset$,

$v(z, t, r, p, q) \to u \cdot v$

for $x \in {}_p[B(z, t, r)]_q \cap Z$, $1 \in B(x, u, w)$,

$v(z, t, r, p, q) \to u$

for $x \in {}_p[B(z, t, r)]_q \cap Z$, $(x, u) \in P$.

We define

$$\tilde{P} = (\tilde{P}, 1) \cup (\tilde{P}, 2) \cup (\tilde{P}, 3)$$

and

$$\tilde{G} = (\tilde{V} \cup T,\ T,\ \tilde{P},\ S).$$

We write

$$\tilde{G} = \tau_3(G).$$

τ_3 is our first normal form transformation.

Let

$$L = L(G), \quad \tilde{L} = L(\tilde{G})$$

be the languages generated by the grammars G and $\tilde{G}$, respectively.

Lemma 5.

$$\tilde{L} \subset L.$$

Proof. We construct a functor from the free x-category $F(\tilde{P})$ into the monoidal category of the relations

$$B \to B' \quad \text{for} \quad B, B' \subset (Z \cup T)^*$$

which are induced by the production set P.

Let $\mathfrak{A}$ be the power set of $(Z \cup T)^*$; then we define the monoid homomorphism

$$\varphi_1\colon (\tilde{V} \cup T)^* \to \mathfrak{A}$$

by setting

$$\varphi_1(t) = \{t\} \quad \text{for} \quad t \in T,$$

$$\varphi_1(S) = \{S\},$$

$$\varphi_1(v(z, t, r, p, q)) = {}_p[B(z, t, r)]_q$$

$$\text{for} \quad v(z, t, r, p, q) \in \tilde{V}.$$

We will write t for $\{t\}$ and S for $\{S\}$. For each $f \in \tilde{P}$ we define

$$\varphi_2'(f) = (\varphi_1(D(f)), \varphi_1(C(f))).$$

One can easily check that for $f \in \tilde{P}$

$$\varphi_1(D(f)) \to \varphi_1(C(f)).$$

We extend (φ_1, φ_2') to the functor $\varphi = (\varphi_1, \varphi_2)$ which is determined uniquely by (φ_1, φ_2'). We have then for $S \xrightarrow{f} w$, $w \in T^*$

$$S = \varphi_1(S) \xrightarrow{\varphi_2(f)} \varphi_1(w) = w,$$

and therefore from the definition of $B \to B'$ for sets we have

$$S \to w$$

in the usual sense.

This means that $w \in L$ for all $w \in \tilde{L}$, and thus $\tilde{L} \subset L$.

Lemma 6.

$$L \subset \tilde{L}.$$

This lemma will be proved in two parts.

Part 1. A derivation step $f_s\colon Z^+ \to (Z \cup T)^+$ is called a *w.f. derivation step* iff $f_s = H_2 \times \ldots \times H_m \times H \times G_m \times \ldots \times G_2$ is a decomposition of f_s into prime derivations in the usual sense (e.g. see ($W2$)). How, let $w_1 \ldots w_n \in B(p, t, r)$ for $w_i, p \in Z$ and $t, r \in T \cup \{1\}$ such that $t \cdot r \neq 1$, and let f_s be a w.f. derivation step with $D(f_s) = w_1 \ldots w_n$. Then we can construct $\tilde{f}_s$ with $D(\tilde{f}_s) = v(p, t, r)$ such that

$$\tilde{\varphi}(C(\tilde{f}_s)) = \varphi(C(f_s))$$

where $\tilde{\varphi}$ and φ are two homomorphisms which forget the nonterminals in a string and are constant on terminals.

To show this we must examine the two cases for n even and n odd. For n even we have

$$w_1 \dots w_n = x_1 \dots x_m y_m \dots y_1$$

where $m=n/2$, $x_i=w_i$, $y_i=w_{n-i+1}$, and $n \geqq 2$. For n odd we have similarly for $m \geqq 1$

$$w_1 \dots w_n = x_1 \dots x_m z y_m \dots y_1.$$

Since the proof for these two cases is similar, we will show the result only for the case that n is odd.

Here we have for $w_1 \dots w_n \in B(p, t, r)$

$$w_1 \dots w_n = x_1 \dots x_m z y_m \dots y_1 \xrightarrow{f_s = H_1 \times \dots \times H_m \times H \times G_m \times \dots \times G_1}$$

$$t_1 x_{1_1} \dots x_{1_{n_1}} \dots t_m x_{m_1} \dots x_{m_{n_m}} t_{m+1} z_1 \dots z_j r_{m+1} y_{m_1} \dots y_{m_{l_m}} r_m \dots y_{1_1} \dots y_{1_{l_1}} r_1.$$

We then construct $\tilde{f}_s$ as shown below for $v(p, t, r)$, the variable corresponding to $B(p, t, r)$, using the rules $\tilde{P}$.

$$v(p, t, r) \xrightarrow{\tilde{f}_{s,1}}$$

$$t_1 v(x_1, t_1, 1) v(p, t_1, r_1, x_1, y_1) v(y_1, 1, r_1) r_1 \xrightarrow{\tilde{f}_{s,2}} \dots \xrightarrow{\tilde{f}_{s,m}}$$

$$t_1 v(x_1, t_1, 1) \dots t_m v(x_m, t_m, 1) v(p, t_1, r_1, x_1 \dots x_m, y_m \dots y_1)$$

$$v(y_m, 1, r_m) r_m \dots v(y_1, 1, r_1) r_1 \xrightarrow{f_{s,m+1}}$$

$$t_1 v(x_1, t_1, 1) \dots t_m v(x_m, t_m, 1) t_{m+1} v(z, t_{m+1}, r_{m+1}) r_{m+1}$$

$$v(y_m, 1, r_m) r_m \dots v(y_1, 1, r_1) r_1.$$

Now, set $\tilde{f}_s = \tilde{f}_{s,m+1} \circ \tilde{f}_{s,m} \circ \dots \circ \tilde{f}_{s,1}$.

Clearly $\tilde{\varphi}(C(\tilde{f}_s)) = \varphi(C(f))$. Further, for each string of "isolated" nonterminals in $C(f_s)$ (a string of nonterminals with a terminal on both ends) there is a corresponding v variable in $C(\tilde{f}_s)$ in the same location. For example, for $t_k x_{k,1} \dots \dots x_{k,n_k}$ in $C(f_s)$, where $x_{k,1} \dots x_{k,n_k} \in B(x_k, t_k, 1)$, we have $t_k v(x_k, t_k, 1)$ in $C(\tilde{f}_s)$ in the same location in terms of the terminal symbols in $C(f_s)$ and $C(\tilde{f}_s)$.

If one of the x_i, z, or y_i — for example x_i — is rewritten to a single terminal followed by no nonterminal string, this corresponds to the fact the $1 \in B(x_i, t_i, 1)$ and thus that the corresponding v variable also does not appear in $C(\tilde{f}_s)$. Therefore the isolated nonterminal strings and the v variables correspond exactly. Using these results the lemma can now be easily proved.

Part 2. For $w \in L$ and $S \xrightarrow{f} w$ with $f \in F(P)$, let

$$f = f_n \circ \dots \circ f_1$$

be the unique (w.f.)-decomposition of f. Then we can construct $\tilde{f} \in F(\tilde{P})$ such that $S \xrightarrow{\tilde{f}} w$.

We will prove this inductively by showing that for each $f_1, f_2, \ldots, f_n$ we can find $\tilde{f}_1, \tilde{f}_2, \ldots, \tilde{f}_n$ such that $\varphi(C(f_i))=\tilde{\varphi}(C(\tilde{f}_i))$ for all i, where $\tilde{f}=\tilde{f}_n \circ \ldots \circ \tilde{f}_1$, and such that the isolated variable strings in $C(f_i)$ correspond exactly to the v variables in $C(\tilde{f}_i)$.

For f_1 we have

$$S \xrightarrow{f_1} tx_1 \ldots x_m r$$

and

$$S \xrightarrow{\tilde{f}_1} tv(S, t, r)r,$$

which clearly satisfies our conditions (if $x_1 \ldots x_m$ is empty in $C(f_1)$, $v(S, t, r)$ does not appear in $C(\tilde{f}_1)$).

Assume that this is true for $f_k \circ \ldots \circ f_1$ and $\tilde{f}_k \circ \ldots \circ \tilde{f}_1$ for $n > k > 1$ to show for the case $k+1$, we look at the partial derivation $f_{k+1} \circ f_k \circ \ldots \circ f_1$. We know from the induction hypothesis that $\tilde{\varphi}(C(\tilde{f}_k))=\varphi(C(f_k))$ and further that the isolated nonterminal strings in $C(f_k)$ correspond exactly with the v variables in $C(\tilde{f}_k)$.

From what we proved in Part 1, then, the result should be clear. To each terminal in $C(f_k)$ we apply an identity derivation; to each string of isolated nonterminals in $C(f_k)$ we apply a w.f. derivation step f_s. The "$\times$" product of these identity derivations and w.f. derivation steps forms f_{k+1} as one can easily see from the proof of Lemma 4 and figure 1.

Let

$$f_{k+1} = g_1 \times \ldots \times g_m$$

be this product. Then we construct

$$\tilde{f}_{k+1} = \tilde{g}_1 \times \ldots \times \tilde{g}_m$$

where $\tilde{g}_j$ is the identity derivation if g_j is the identity, and $\tilde{g}_j$ is the corresponding $\tilde{f}_s$ derivation if g_j is a "type f_s" derivation. Clearly, then, the conditions of our assumptions hold, and we have that $C(f)=C(\tilde{f})$ and thus that $L \subset \tilde{L}$.

Our proof gives a sharper result than stated in the lemma. $f \to \tilde{f}$ is a mapping. If this mapping is surjective, then the multiplicity of each element $w \in L$ relative to G will not be greater relative to $\tilde{G}$.

Analysing the proof of Lemma 5, one also sees that given $\tilde{f} \in F(\tilde{P})$ one can find a $g \in F(P)$ such that $\tilde{g}=\tilde{f}$.

From Lemma 5 and Lemma 6 and the above remark we have

Theorem 2. For each language L generated by a context-free grammar G our transformation τ_3 produces a context-free grammar $\tilde{G}=\tau_3(G)$ which generates L and in which the productions are of the form

$$z \to tpr \quad \text{or} \quad z \to v$$

where $p \in Z \cup Z^2 \cup Z^3$, $v \in T \cup T^2$ and $t, r \in T$, $z \in Z$. τ_3 does not increase the multiplicity of words.

Corollary. τ_3 transforms unambiguous grammars into unambiguous grammars.

We now define two more transformations τ_1 and τ_2 for which the same theorems hold.

τ_1 is the transformation into Greibach normal form from Chomsky normal form with at most two nonterminals in the right hand side of each production.

τ_2 transforms each G in Chomsky normal form into a grammar $\tilde{G}$ in which the productions are of the form

$$z \to tqr \quad \text{or} \quad z \to v$$

with $q \in Z \cup Z^2$, $v \in T \cup T^2$, $z \in Z$, $t, r \in T$.

We will only give the relations corresponding to $(P', 1)$, $(P', 2)$ and $(P', 3)$. From this the definitions of τ_1 and τ_2 and the proofs of the corresponding theorems will be obvious to the reader.

The transformation τ_1

Now we apply the methods which led us to the just proved normal form theorem to the recursive equation of theorem 2 in [Ho 2].

$$B(s, ut, 1) = \bigcup_{z \in Z} B(z, t, 1)_z[B(s, u, 1)].$$

Again, we can try to construct productions from these B sets of the form

$$B(s, v, 1) \to tB(s, vt, 1)$$

for $s \in Z$, $v \in T^+$, $t \in T$. Factoring the right hand side, we have

$$B(s, v, 1) \to t \cdot B(z, t, 1) \cdot {}_z[B(s, v, 1)]$$

for $t \in T$, $z \in Z$, $v \in T$, $s \in Z$.

We introduce as before variables $x(z, t, p)$ which we assign to ${}_p[B(z, t, 1)]$. Here we get a production system

$$x(s, v, p) \to t \cdot x(z, t, 1) \cdot x(s, v, pz)$$

for $B(z, t, 1) \neq \emptyset$, ${}_{pz}[B(s, v, 1)] \neq \emptyset$, and $B(z, t, 1)$ and ${}_{pz}[B(s, 0, 1)] \neq \{1\}$ and

$$x(s, v, p) \to t \cdot x(z, t, 1),$$
$$x(s, v, p) \to t \cdot x(s, v, pz)$$

for $1 \in_{pz}[B(s, v, 1)]$ or $1 \in B(z, t, 1)$, respectively.

We define the first and the terminal productions as follows:

$$S \to t \cdot x(S, t, 1) \quad \text{for} \quad B(S, t, 1) \neq \emptyset;$$
$$S \to t \quad \text{for} \quad (S, t) \in P;$$
$$x(z, t, p) \to r \quad \text{for} \quad y \in_p [B(z, t, 1)] \quad \text{and} \quad (y, r) \in P.$$

This grammar we call $\tilde{G}$ and the transformation from G to $\tilde{G}$ is the desired transformation τ_1.

As in the case of τ_3 one proves

Theorem 3. The transformation τ_1 transforms context-free grammars G in Chomsky normal form into grammars $\tilde{G} = \tau_1(G)$ which are in Greibach normal form. The productions of $\tilde{G}$ contain on the right hand side not more than two variables. The transformation τ_1 does not increase the multiplicity of words.

The transformation τ_2

Let

$$\mathfrak{R} = \{{}_u[B(z, t, r)]_v | z \in Z,\ t \in T \cup \{1\},\ r \in T \cup \{1\},\ t \cdot r \neq 1,\ u \in Z^*,\ v \in Z^*\}.$$

As we have seen $\mathfrak{R}$ is a finite set. We derived from relations of the form

$$R \to tR_1 R_2 R_3 r$$

and

$$S \to tRr$$

a cubic normal form for the context-free languages.

Now from relations of a similar form

$$[R_1 R_2] \to tB(z, t, 1) \cdot {}_z[R_1 R_2]_y \cdot B(y, 1, r) \cdot r$$

we can derive a quadratic normal form from the fact that

$${}_z[R_1 R_2]_y = {}_z[R_1] \cdot [R_2]_y \cup \sigma(R_1) \cdot {}_z[R_2]_y \cup \sigma(R_2) \cdot {}_z[R_1]_y$$

where

$$\sigma(R) = \begin{cases} \emptyset & \text{for } 1 \in R \\ \{1\} & \text{for } 1 \in R. \end{cases}$$

If we now write

$$R_0 = B(z, t, 1),\quad R_1' = {}_z[R_1],\quad R_2' = [R_2]_y,\quad R_3 = B(y, 1, r),$$
$$\tilde{R}_1 = \sigma(R_2) \cdot {}_z[R_1]_y,\quad \tilde{R}_2 = \sigma(R_1) \cdot {}_z[R_2]_y$$

then we have the relations

$$[R_1 R_2] \to t \cdot [R_0 R_1'] \cdot [R_2' R_3] \cdot r,$$
$$[R_1 R_2] \to t \cdot [R_0 \tilde{R}_1] \cdot [R_3] \cdot r,$$
$$[R_1 R_2] \to t \cdot [R_0] \cdot [\tilde{R}_2 R_3] \cdot r.$$

$\mathfrak{R}$, the set of all valid R sets, is closed under left and right divisions by construction, and from $\mathfrak{R}$ finite it follows that $\mathfrak{R} \cup \mathfrak{R} \cdot \mathfrak{R}$ is also finite.

If we now choose variables $v(R_1)$ and $v(R_2, R_2)$ for $R_1, R_2 \in \mathfrak{R}$ just as we did in developing our cubic normal form we get a production system $\tilde{P}$ of the type

$$y \to txzr,$$
$$y \to txr,$$
$$y \to tr,$$
$$y \to t,$$

or y, x, z nonterminals and t, r terminals.

This is the transformation τ_2.

As in the cubic case we have the following

Theorem 4. The normal form transformation

$$G \xrightarrow{\tau_2} \tilde{G}$$

defined for grammars in Chomsky normal form has the property $L(G) = L(\tilde{G})$. The transformation does not increase the multiplicity of the words $w \in L$.

The proof is completely analogous to the proof of theorem 2 and is therefore left to the reader.

Functorial properties of the normal form transformations

Let $F(P_i) = ((Z_i \cup T)^*, \mathfrak{M}_i, D, C)$ for $i = 1, 2$ be two x-categories generated by the context-free production systems P_1 and P_2 in Chomsky normal form. Further let $\varphi = (\varphi_1, \varphi_2)$ be an x-functor from $F(P_1)$ to $F(P_2)$.

This means that

$$\varphi_1 \colon (Z_1 \cup T)^* \to (Z_2 \cup T)^*$$

is a monoid homomorphism and that

$$\varphi_2 \colon \mathfrak{M}_1 \to \mathfrak{M}_2$$

fulfills

$$\varphi_2(f \circ g) = \varphi_2(f) \circ \varphi_2(g)$$

if $f \circ g$ is defined, and

$$\varphi_2(f \times g) = \varphi_2(f) \times \varphi_2(g).$$

Also for identities 1_w we have

$$\varphi_2(1_w) = 1_{\varphi_1(w)}.$$

We restrict ourselves to the case $\varphi_1(T) \subset T$ and $\varphi_1(Z_1^*) \subset Z_2^*$. From this follows

$$\text{length}(w) = \text{length}(\varphi_1(w)) \quad \text{for} \quad w \in T^*.$$

We have no derivation

$$1 \to u$$

for $u \neq 1$. Because we are in Chomsky normal form we have no superfluous variables. This means for each $z \in Z$ there exists

$$z \xrightarrow{f} w, \ w \in T^+;$$

therefore $\varphi_1(z) = 1$ would be a contradiction. From this and the fact that φ_1 is length preserving on T^* it also follows that $\varphi_1(Z_1) \subset Z_2$. Thus, since we are using Chomsky normal form, we have

$$\varphi_1(P_1) \subset P_2.$$

Now let

$$z \xrightarrow{f} tur$$

be a (t, r)-prime derivation. Then

$$\varphi_1(z) \xrightarrow{\varphi_2(f)} \varphi_1(t)\varphi_1(u)\varphi_1(r)$$

is $(\varphi_1(t), \varphi_1(r))$-prime. Therefore

$$\varphi_1(B(z, t, r)) \subset B(\varphi_1(z), \varphi_1(t), \varphi_1(r)).$$

For $R \subset Z^*$, $x, y \in Z$ the identity

$$\varphi_1({}_x[R]_y) = {}_{\varphi_1(x)}[\varphi_1(R)]_{\varphi_1(y)}$$

holds since $\varphi_1(Z_1) \subset Z_2$.

Let $\mathfrak{R}_i$ be the set of our sets ${}_p[B(z, t, r)]_q \neq \emptyset$ belonging to G_1 and G_2 respectively. Then for the variables $v(z, t, r, p, q)$ we can write $v(R)$ for certain $R \in \mathfrak{R}_i$. Then we have for the set $\tilde{Z}_i$ of variables of $\tilde{G}_i$

$$\tilde{Z}_i = \{v(R) | R \in \mathfrak{R}_i\}, \quad i = 1, 2.$$

Now φ_1 induces a mapping

$$\varphi_1 \colon \mathfrak{R}_1 \to \mathfrak{R}_2.$$

Using this we can define the monoid homomorphism

$$\tilde{\varphi}_1 \colon (\tilde{Z}_1 \cup T)^* \to (\tilde{Z}_2 \cup T)^*$$

by setting

$$\tilde{\varphi}_1(t) = t \quad \text{for} \quad t \in T$$

and

$$\tilde{\varphi}_1(v(R)) = v(\varphi_1(R)) \quad \text{for} \quad R \in \mathfrak{R}_1.$$

It is clear, then, that the following diagram commutes

$$\begin{tikzcd} O(F(P_1)) \arrow[r, "\varphi_1"] \arrow[d, "\tau"'] & O(F(P_2)) \arrow[d, "\tau"] \\ O(F(\tilde{P}_1)) \arrow[r, "\tilde{\varphi}_1"] & O(F(\tilde{P}_2)) \end{tikzcd}$$

for $\tau = \tau_1, \tau_2, \tau_3$, where 0 is the object set of the given categories.

We can now define the function φ_2' which maps the productions of $\tilde{P}_1$ to productions in $\tilde{P}_2$ by setting

$$\varphi_2'(z, q) := (\tilde{\varphi}_1(z), \tilde{\varphi}_1(q))$$

for $(z, q) \in \tilde{P}_1$.

Extending $(\tilde{\varphi}_1, \varphi_2')$ to the x-functor $(\tilde{\varphi}_1, \tilde{\varphi}_2)$ we have proved the following

Theorem 5. Let τ be one of our normal form transformations τ_1, τ_2, τ_3 and let $\varphi = (\varphi_1, \varphi_2)$ be a functor from $F(P_1)$ to $F(P_2)$, where P_1 and P_2 are in Chomsky normal form with $\varphi_1(T) \subset T$ and $\varphi_1(Z_1^*) \subset Z_2^*$. Then there exists a natural transformation of φ to a functor $\tilde{\varphi}$ from $F(\tau(P_1))$ to $F(\tau(P_2))$.

The theorem states in other words that the diagramm

$$\begin{tikzcd} F(P_1) \arrow[r, "\varphi"] \arrow[d, "\tau"'] & F(P_2) \arrow[d, "\tau"] \\ F(\tilde{P}_1) \arrow[r, dashed] & F(\tilde{P}_2) \end{tikzcd}$$

has a solution $\tilde{\varphi}$. This means that the τ_i induce a functor between the functor categories of the x-categories $F(P)$, P in Chomsky normal form, and the functor categories $F(\tilde{P})$ with $\tilde{P}$ in one of the three normal forms 1, 2 or 3.

Transformations of linear languages

We have seen that the transformations τ_i do not increase the multiplicity of words. Therefore the question arises whether an $LR(k)$-grammar G is transformed into $LR(k')$-grammar $\tilde{G}$ by our transformations τ_i. We are not able to solve this problem here, but we show that τ_1 transforms one sided linear grammars into minimal linear grammars. This means that in this case τ_1 transforms non-$LR(k)$-grammars into $LR(0)$-grammars. τ_1 here corresponds to the reduction of finite automata.

Let P be a left-linear grammar where productions are of the type

$$z \to z' \cdot t, \quad z \to t$$

for $z, z' \in Z$ and $t \in T$, where Z is the variable alphabet, and T is the terminal alphabet. We transform these productions into Chomsky normal form by introducing the variable alphabet $X = \{(x, t) | t \in T\}$ where x is a fixed symbol.

We define

$$P_C = \{(z, z' \cdot (x, t)) | (z, z' \cdot t) \in P\}$$
$$\cup \{((x, t), t) | t \in T\} \cup \{(z, t) | (z, t) \in P\}.$$

P_C, then, is in Chomsky normal form and the grammars $G = (Z \cup T, T, P, S)$ and $G' = (Z \cup X \cup T, T, P_C, S)$ generate the same language L.

Now we apply our transformation τ_1 to P_C. We have for $z \in Z$ and $(x, t) \in X$

$$B(z, t, 1) \subset X^*$$

and

$$B((x, t), u, 1) \subset \{1\}.$$

From this follows

$$_z[B(y, t, 1)] = \emptyset$$

for $z \in Z$ and $y \in Z \cup X$.

Therefore our relations which define $\tilde{P}_C$ have the form

$$_p[B(y', t, 1)] \to u \cdot {}_{py}[B(y', t, 1)]$$

for $p \in X^*$, $y = (x, u) \in X$, and $y' \in Z$.

Now let

$$\varphi : X^* \to T^*$$

be the monoid isomorphism defined by

$$\varphi(x, t) = t.$$

Then

$$\varphi({}_p[B(y, t, 1)]), \quad y \in Z, \quad t \in T, \quad p \in X^*$$

defines the syntactical congruence classes of L (i.e. the left invariant equivalence relations). This means that τ_1 transforms P_C into a minimal grammar for L.

We therefore have the following

Theorem 6. τ_1 transforms left linear grammars — represented in Chomsky normal form as shown — into minimal right linear grammars.

Corollary. τ_1 transforms certain non-$LR(k)$-grammars into $LR(0)$-grammars. There exist grammars such that under the transformation τ_1 the multiplicity of words decreases properly.

One can easily prove similar results for the transformations τ_2 and τ_3.

From our theorem about the multiplicity of words it follows that the transformations τ_i transform an $LR(k)$-grammar G into an unambiguous grammar G. τ_2 and τ_3 do not preserve the $LL(k)$ and $LR(k)$ property of grammars, but τ_1 does preserve it as we can show [Ho 3].

A normal form for the Chomsky—Schützenberger theorem

Using our normal form transformations τ_2 and τ_3 one easily derives a normal form for the theorem of Chomsky—Schützenberger.

Let

$$X_k = \{x_1, \dots, x_k, x_1^{-1}, \dots, x_k^{-1}\}$$

where x_i, x_i^{-1} are bracket pairs and D_k the corresponding Dyck language over X_k. The well known theorem states that for each context-free language $L \subset T^*$ there exists an alphabet X_k, a standard regular event R, and a homomorphism $\varphi\colon X_k^* \to T^*$ with $\varphi(X_k) \subset T \cup \{1\}$ such that

$$L = \varphi(D_k \cap R).$$

Using our normal forms and following the well known proof of this theorem one finds the normal form of

Theorem 7. For each context-free language $L \subset T^*$ one can find X_k, φ, and R such that $L = \varphi(D_k \cap R)$ and from $\varphi(w) \in T$ and the existence of u, v such that $uwv \in R$ it follows length $(w) \leqq 3$.

From this theorem we arrive at the theorem of S. Greibach [Gr] about a hardest context-free language as it was proved in [Ho 1].

Abstract

We discuss three normal form transformations τ_1, τ_2 and τ_3 of grammars G which are in Chomsky normal form into grammars G_1, G_2 and G_3 respectively. G_1 is in Greibach normal form with nonterminal productions restricted to $z \to tp$ such that $t \in T$ and $p \in Z^+$ and length $(p) \leqq 2$. The nonterminal productions of G_2 and G_3 are of the form $z \to tpr$ such that $t, r \in T$ and $p \in Z^+$, length $(p) \leqq 2$ or length $(p) \leqq 3$, respectively. It is shown that these transformations do not increase the multiplicity of words in the generated languages. Furthermore we show that certain functorial relations between languages are preserved under these transformations. The restriction of τ_1 to one sided linear grammars produces the minimal grammars. τ_2 and τ_3 do not preserve the $LR(k)$ property of grammars. τ_1 preserves $LL(k)$ for $k \geqq 0$ and $LR(k)$ for $k > 1$, $LR(0)$ may be transformed into $LR(1)$ as we show in the following paper.

UNIVERSITÄT DES SAARLANDES
D-66 SAARBRÜCKEN

References

[A—ULL] AHO, V. and J. D. ULLMAN, *The theory of parsing, translation, compiling,* vol. 1.

[BE] BENSON, D. B., *Some properties of normal form grammars,* Comp. Sc. Dept. Washington State Univ., TR CS 75 24, July 1976.

[CH—SCH] CHOMSKY, N. and M. P. SCHÜTZENBERGER, *The algebraic theory of context-free languages, computer programming and formal systems,* North-Holland Publ., 1970, pp. 116—161.

[GE—HA] GELLER, M. M., M. A. HARRISON, I. M. HAVEL, *Normal forms of deterministic grammars,* Discrete Math.

[GR] GREIBACH, S. A., The hardest context-free language, *SIAM J. Computing,* v. 2, 1973, pp. 304—310.

— Erasable context-free languages, *Information and Control,* v. 4, 1975, pp. 301—326.

[Ho 1] HOTZ, G., The theorem of Chomsky—Schützenberger and the hardest context-free language of S. Greibach, *Astérisque,* v. 38—39, 1976, pp. 105—115.

[Ho 2] HOTZ, G., Sequentielle Analyse kontextfreier Sprachen, *Acta Informatica,* v. 4, 1974, pp. 55—75.

[Ho 3] HOTZ, G., *LL(k)*- und *LR(k)*-Invarianz von kontextfreien Grammatiken unter einer Transformation auf Greibach-Normalform, *EIK,* No. 1—2, 1979.

[Ho—CL] HOTZ, G., V. CLAUS, Automatentheorie und formale Sprachen III, *BI-Informatik,* Hochschulskripten, Mannheim, 1972.

[KN] KNUTH, D. E., On the translation of languages from left to right, *Information and Control,* v. 8, 1965, pp. 607—639.

[MAU] MAURER, H. A., Theoretische Grundlagen der Programmiersprachen, *BI-Informatik,* Hochschulskripten, Mannheim, 1969.

[SA] SALOMAA, A., *Formal languages,* Academic Press, N. Y and London, 1973.

[SCH] SCHAUERTE, R., *Transformationen von LR(k)-Grammatiken,* Diplomarbeit, Göttingen, 1973.

(Received March 22, 1978)

Processing of random sequences with priority

By A. IVÁNYI and I. KÁTAI

To the memory of Professor László Kalmár

Introduction

This paper is devoted to study of processing random sequences with priority. At first we formulate the general problem (§ 1.), later we show: the state sequence characterizing the course of the processing — as processing of independent homogeneous Markov-chains — is also a homogeneous Markov-chain (§ 2.).

We deal with characterizing the processing speed (§ 3.). Since the stationary initial distribution plays a main role, therefore we give a simple algorithm to determine it: when the transition probability matrix is the simplest (§ 4.) and for two sequences (§ 5.).

Finally we investigate the asymptotic behaviour of the speed (§ 6.).

Our work has practical importance e.g. in computer performance analysis, more precisely in modelling of multiprogrammed computers with one processor and interleaved memory [1]. In this case the programs are modelled by sequences (the program with the greatest priority by the first sequence etc.), the chosen measure of the speed corresponds to the average number of the executed operations in a time unit, the transition probability matrix with the same elements corresponds to the random program behaviour model and the asymptotic problem is connected with the great number of memory moduls.

§ 1. Formulation of the problem

Let $\mathcal{A}_N$ denote the set $\{1, 2, \dots, N\}$, and

$$\begin{matrix} f_1^{(1)}, f_2^{(1)}, \dots \\ \vdots \\ f_1^{(r)}, f_2^{(r)}, \dots \end{matrix} \tag{1.1}$$

r infinite sequences consisting of the elements of $\mathcal{A}_N$. We process the elements in the sequences according to the following rules:

1. Processing proceeds in the points of time $1, 2, \dots$; let i be equal to 1.

2. Let k_1 denote the greatest positive integer for which the elements $f_1^{(1)}, \dots, f_{k_1}^{(1)}$ are mutually distinct. If $k_1, \dots, k_{t-1}$ have been defined, then let k_t $(t=2, \dots, r)$ denote the greatest nonnegative integer for which

$$\bigcup_{j=1}^{t-1} \{f_1^{(j)}, \dots, f_{k_j}^{(j)}\} \cap \{f_1^{(t)}, \dots, f_{k_t}^{(t)}\} = \emptyset \tag{1.2}$$

holds.

3. In the i-th point of time we process the first k_t elements of the t-th $(t=1, \dots, r)$ sequence. We omit the processed elements from the sequences, and reduce the lower index of the remaining elements by k_t in the t-th $(t=1, \dots, r)$ sequence.

4. We add 1 to i and continue the processing from the rule 2.

For a more precise characterizing of the processing we register the first and last processed and the first nonprocessed elements for every point of time. Therefore the processing in the first point of time is characterized by the array

$$\begin{array}{l} f_1^{(1)}, |\dots, f_{k_1}^{(1)}, \| f_{k_1+1}^{(1)} \\ \vdots \\ f_1^{(t)}, |\dots, f_{k_t}^{(t)}, \| f_{k_t+1}^{(t)} \\ \vdots \\ f_1^{(r)}, |\dots, f_{k_r}^{(r)}, \| f_{k_r+1}^{(r)}. \end{array} \tag{1.3}$$

If $k_t=0$ holds for a given t, then we have $*, \| f_1^{(t)}$ in the t-th line of (1.3). The star shows that none of the elements has been processed. For the sake of brevity let

$$A_t = \langle f_1^{(t)}, |\dots, f_{k_t}^{(t)}, \| f_{k_t+1}^{(t)} \rangle \quad \text{or} \quad A_t = \langle *, \| f_{k_t+1}^{(t)} \rangle, \tag{1.4}$$

resp. By using this notation, the processing in the first point of time is characterized by

$$\vartheta = (A_1, \dots, A_r). \tag{1.5}$$

Let $\mathcal{D}_r$ denote the set of all possible ϑ's. In other words $\mathcal{D}_r$ is the set of all ϑ's that are representable in form (1.3) giving suitable values to the elements $f_i^{(t)}$. It is clear, that $(A_1, \dots, A_r)$ belongs to $\mathcal{D}_r$ if and only if the following conditions hold:

1. $A_1 = \langle i_1, |i_2, \dots, i_k, \| j \rangle$; $i_1, \dots, i_k, j \in \mathcal{A}_N$; $i_1, \dots, i_k$ are mutually distinct, $j \in \{i_1, \dots, i_k\}$.

2. Let $A_1, \dots, A_{t-1}$ be defined, then

$$A_t = \langle s_1, |s_2, \dots, s_m \| l \rangle, \quad s_1, \dots, s_m, \ l \in \mathcal{A}_N,$$

and

a) $\{s_1, \dots, s_m\} \cap \bigcup_{n=1}^{t-1} A_n = \emptyset$,

b) $s_1, s_2, \dots, s_m$ are mutually distinct,

c) $l \in \{s_1, \dots, s_m\} \cup \left(\bigcup_{n=1}^{t-1} A_n \right)$

or

$$A_t = \langle *, \| l \rangle, \quad \text{and} \quad l \in \bigcup_{n=1}^{t-1} A_n.$$

After this the processing of a given array of type (1.1) can be described by the state sequence

$$\vartheta^{(1)}, \vartheta^{(2)}, \ldots, \vartheta^{(s)}, \ldots$$
$$\vartheta^{(s)} \in D_r \quad (s = 1, 2, \ldots). \tag{1.6}$$

It is obvious that there are pairs $C_1, C_2 \in \mathcal{D}_r$ that cannot occur as consecutive states, i.e. for which $\vartheta^{(s)} = C_1$, $\vartheta^{(s+1)} = C_2$.

Let

$$\vartheta^{(s)} = (A_1^{(s)}, \ldots, A_r^{(s)}), \tag{1.7}$$

where

$$A_t^{(s)} = \langle i_{1,t}^{(s)}, | \ldots, i_{k_t^{(s)},t}^{(s)}, \| j_t^{(s)} \rangle \tag{1.8}$$

or

$$A_t^{(s)} = \langle *, \| j_t^{(s)} \rangle. \tag{1.9}$$

Let *in* $\vartheta^{(s)}$ and *fin* $\vartheta^{(s)}$ denote the initial and final elements of $\vartheta^{(s)}$, i.e.

$$in\, \vartheta^{(s)} = (i_{1,1}^{(s)}, \ldots, i_{1,r}^{(s)}); \; fin\, \vartheta^{(s)} = (j_1^{(s)}, \ldots, j_r^{(s)}), \tag{1.10}$$

remarking that if $A_t^{(s)} = \langle *, \| j_t^{(s)} \rangle$, then in *in* $\vartheta^{(s)}$ we put $*j_t^{(s)}$ instead of $i_{1,t}^{(s)}$. It is clear, that the transition $\vartheta^{(s)} \to \vartheta^{(s+1)}$ is realisable if and only if *fin* $\vartheta^{(s)} =$ *in* $\vartheta^{(s+1)}$ holds. Deciding about this equality we do not take into account whether the components of *in* $\vartheta^{(s+1)}$ contain stars or not.

§ 2. Processing of independent Markov-chains

Let $\xi_i^{(l)}$ $(l=1, \ldots, r;\ i=1, 2, \ldots)$ be random variables with values from $\mathcal{A}_N$ for which the following conditions hold:

1. The sequences $\xi_i^{(l)}$ $(i=1, 2, \ldots)$ for every l form a homogeneous Markov-chain with an initial distribution π_l and transition probability matrix Π_l, i.e.

$$\pi_l(p(1, l), \ldots, p(N, l)), \quad \text{where} \quad p(k, l) = P(\xi_1^{(l)} = k)$$

and

$$\Pi_l = [p(x, y, l)], \quad \text{where} \quad p(x, y, l) = P(\xi_{\nu+1}^{(l)} = x | \xi_\nu^{(l)} = y).$$

2. The sequences $\xi_i^{(l)}$ $(i=1, 2, \ldots)$ are mutually independent.
3. The elements of the matrices Π_l are positive.

Our job is to process the array of random variables

$$\begin{matrix} \xi_1^{(1)}, \xi_2^{(1)}, \ldots \\ \vdots \\ \xi_1^{(r)}, \xi_2^{(r)}, \ldots \end{matrix} \tag{2.1}$$

by using the algorithm defined in § 1.

Let

$$\mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(s)}, \ldots$$
$$\mathcal{B}^{(s)} \in \mathcal{D}_r \quad (s = 1, 2, \ldots) \tag{2.2}$$

denote the state sequence of type (1.5).

We prove the following

Theorem 1. Under the previous conditions the sequence (2.2) represents a homogeneous Markov-chain.

Proof. Let us compute the probabilities

$$P(\mathcal{B}^{(1)}=\vartheta^{(1)})=q(\vartheta^{(1)}),$$

$$P(\mathcal{B}^{(s+1)}=\vartheta^{(s+1)}|\mathcal{B}^{(1)}=\vartheta^{(1)},\dots,\mathcal{B}^{(s)}=\vartheta^{(s)}).$$

We shall use the notations (1.8) and (1.9).

Let

$$\tau(A_t^{(1)})=p(i_{1,t}^{(1)};t)\cdot p(i_{2,t}^{(1)},i_{1,t}^{(1)};t)\dots p(j_t^{(1)},i_{k_t^{(1)},t}^{(1)};t), \tag{2.3}$$

if $A_t^{(1)}$ has the form (1.8) and

$$\tau(A_t^{(1)})=p(j_t^{(1)};t), \tag{2.4}$$

if $A_t^{(1)}$ has the form (1.9).

It is clear, that

$$q(\vartheta^{(1)})=\prod_{t=1}^{r}\tau_t(A_t^{(1)}).$$

Let

$$\lambda_t(A_t^{(s)})=p(i_{2,t}^{(s)};i_{1,t}^{(s)};t)\dots p(j_t^{(s)};i_{k_t,t}^{(s)};t), \tag{2.5}$$

if $A_t^{(s)}$ has the form (1.8) and let

$$\lambda_t(A_t^{(s)})=1, \tag{2.6}$$

if $A_t^{(s)}$ has the form (1.9). Further let

$$Q(\vartheta^{(s)})=\prod_{t=1}^{r}\lambda_t(A_t^{(s)}). \tag{2.7}$$

Since the sequences $\xi_i^{(l)}$ form homogeneous Markov-chains, therefore

$$P(\mathcal{B}^{(1)}=\vartheta^{(1)},\dots,\mathcal{B}^{(s+1)}=\vartheta^{(s+1)})=q(\vartheta^{(1)})Q(\vartheta^{(2)})\dots Q(\vartheta^{(s+1)}),$$

if $\vartheta^{(1)},\dots,\vartheta^{(s+1)}$ is a realisable sequence. It is clear, that for a nonrealisable sequence

$$P(\mathcal{B}^{(1)}=\vartheta^{(1)},\dots,\mathcal{B}^{(s+1)}=\vartheta^{(s+1)})=0.$$

So we have proved that (2.2) is a homogeneous Markov-chain with initial distribution (2.4) and with the following transition probabilities:

$$P(\mathcal{B}^{(s+1)}=\vartheta^{(s+1)}|\mathcal{B}^{(s)}=\vartheta^{(s)},\dots,\mathcal{B}^{(1)}=\vartheta^{(1)})=$$

$$=\begin{cases}Q(\vartheta^{(s+1)}), & \text{if } \ in\,\vartheta^{(s+1)}=fin\,\vartheta^{(s)}\\ 0, & \text{if } \ in\,\vartheta^{(s+1)}\neq fin\,\vartheta^{(s)}.\end{cases} \tag{2.8}$$

Now we shall prove, that under a suitable positive k all of the conditional probabilities

$$P(\mathcal{B}^{(s+k)}=C_2|\mathcal{B}^{(s)}=C_1)$$

are positive for every $C_1, C_2\in\mathcal{D}_r$.

Since $Q(C)$ are positive for every $C\in\mathcal{D}_r$, thus we have to show that there exists a realisable sequence

$$C_1 = \vartheta_1, \vartheta_2, \dots, \vartheta_{k+1} = C_2.$$

Let $\mathit{fin}\, C_1=(j_1, \dots, j_r)=\beta_1$, $\mathit{in}\, C_2=(i_1, i_2, \dots, i_r)=\alpha_2$, where α_2 may have stars. It is clear that there is a realisable sequence starting with C_1 and ending with ϑ_{u-1}, where $\mathit{fin}\, \vartheta_{u-1}=\alpha_2$. Since the number of possible states is finite, we can find a bound d with $u\leqq d$. Let $k>d$, and

$$\vartheta^{(s)} = \begin{bmatrix} i_1, & |i_2, \dots, i_r, & \| i_1 \\ *, & & \| i_2 \\ \vdots & & \\ *, & & \| i_r \end{bmatrix} \quad (s = u, \dots, k).$$

In this case the subsequence $\vartheta^{(k)}$, $\vartheta^{(k+1)}$ is realisable.

Hence immediately follows the following

Theorem 2. Under the conditions of Theorem 1 the sequence (2.2) is an ergodic Markov-chain.

§ 3. Determination of the processing speed

Let $l(\vartheta)$ (interpreted for every $\vartheta\in\mathcal{D}_r$) be an arbitrary function having complex values.

Since any given array (1.1) determines uniquely the sequence (1.5), therefore the sequence

$$l(\vartheta^{(1)}), l(\vartheta^{(2)}), \dots \tag{3.1}$$

is determined too. We are interested in such functions l that characterize the speed of the processing. Assuming that the conditions stated for $\xi_i^{(l)}$ in § 2. are satisfied, we shall show that the mean values and other moments of the random variables

$$\eta_t(l) = \sum_{j=1}^{t} l(\mathcal{B}^{(j)}) \tag{3.2}$$

can be computed by using known theorems.

Let $(\Omega, \mathcal{A}, \mathcal{P})$ be a probability space, $\varrho_1, \varrho_2, \dots$ a homogeneous Markov-chain with a finite set of possible states $\{1, 2, \dots, n\}$. Let

$$\pi = (p_1, \dots, p_n) \tag{3.3}$$

denote the initial distribution and

$$\Pi = [p_{ij}]_{i,j=1,\dots,n} \tag{3.4}$$

the matrix of transition probabilities.

Let

$$p_{ij}^{(k)} = P(\varrho_{t+k}=j\,|\,\varrho_t = i) \quad (i = 1, 2, \dots).$$

The following wellknown assertion is due to Markov.

Lemma 1. Let us suppose that there exist j and k such that $p_{ij}^{(k)}>0$ for $i=1, \ldots, n$. Then

$$\lim_{r\to\infty} p_{ij}^{(r)} = x_j, \quad \sum_{j=1}^{n} x_j = 1, \tag{3.5}$$

further

$$|p_{ij}^{(r)}-x_j| \leqq C\cdot\varphi^r, \tag{3.6}$$

where $C>0$ and φ $(0<\varphi<1)$ are suitable constants.

Let f be a function having complex values defined on the set $\{1, \ldots, n\}$. Let $M_\pi f(\varrho_t)$ denote the mean value of $f(\varrho_t)$ supposing that ϱ_t has an initial distribution π. Let $\theta_1, \theta_2, \ldots$ be a stationary Markov-chain on the set $\{1, \ldots, n\}$ with a transition probability matrix (3.4). Therefore the Markov-chain $\theta_1, \theta_2, \ldots$ has an initial distribution $x=(x_1, \ldots, x_n)$. As an immediate consequence of Lemma 1 we get

$$|M_\pi f(\varrho_t)-M_x f(\theta_t)| \leqq C_1\varphi^t, \tag{3.7}$$

where $C_1>0$, $0<\varphi<1$ are constants. Since $\theta_1, \theta_2, \ldots$ is stationary, therefore

$$M_x f(\theta_t) = M_x f(\theta_1), \tag{3.8}$$

and from (3.7) it follows that

$$M_\pi\left(\sum_{j=1}^{t} f(\varrho_j)\right) = t\big(M_x f(\theta_1)\big)+O(1). \tag{3.9}$$

Theorem 2 guarantees the fulfilment of Lemma 1 for the sequence (2.2). The approximate determination of $M\eta_t(l)$ is simple, if the stationary initial distribution belonging to the chain (2.2) is known.

The explicite calculation of the stationary values is in general a cumbersome matter, since the number of elements in $\mathcal{D}_r$ is about n^3 even for $r=1$.

Now we give a simple algorithm to compute it in a special case.

§ 4. Algorithm for the computation of the stationary distribution

Let the random variable sequences (2.1) be mutually independent with the distribution

$$P(\xi_i^{(l)} = k) = \frac{1}{N} \quad (l = 1, \ldots, r;\ k = 1, \ldots, N;\ i = 1, 2, \ldots). \tag{4.1}$$

Let $\vartheta=(A_1, \ldots, A_r)$ denote the processing in the first point of time, and $b(A_j)$ denote the number of processed elements of the chain $\xi_i^{(j)}$ (at this time), and $b(\vartheta)$ denote

$$b(\vartheta) \doteq \big(b(A_1), \ldots, b(A_r)\big). \tag{4.2}$$

For given integers $k_1\geqq 1$, $k_i\geqq 0$ $(i=2, \ldots, r)$ let $p(k_1, \ldots, k_r)$ denote the probability of the event $b(\vartheta)=(k_1, \ldots, k_r)$, i.e.

$$p(k_1, \ldots, k_r) = P\big(b(\vartheta) = (k_1, \ldots, k_r)\big). \tag{4.3}$$

Let $s_0=0$, $s_t=k_1+\dots+k_t$ $(t=1,\dots,r)$. It is clear that $p(k_1,\dots,k_r)=0$ unless

$$1 \leqq k_1 \leqq N, \quad 0 \leqq k_t \leqq N-1, \quad s_t \leqq N \quad (t=2,\dots,r). \tag{4.4}$$

Let

$$\gamma(t, N) = \prod_{\nu=1}^{t}\left(1-\frac{\nu}{N}\right) \tag{4.5}$$

and let V_m^k denote the number of k-variations of m elements.

It follows from simple combinatorial considerations that in the cases (4.4)

$$\begin{aligned} p(k_1,\dots,k_r) &= \frac{1}{N^{s_r+r}} V_N^{k_1} V_{N-s_1}^{k_2} \dots V_{N-s_{r-1}}^{k_r} \cdot s_1 s_2 \dots s_r = \\ &= \frac{N!}{(N-s_r)!}\cdot\frac{s_1 s_2 \dots s_r}{N^{s_r+r}} = \gamma(s_r-1, N)\frac{s_1}{N}\cdot\frac{s_2}{N}\dots\frac{s_r}{N}. \end{aligned} \tag{4.6}$$

From this representation we can easily get the limit distributions of s_t's as $N\to\infty$ for a fixed r. We are going to devote an other paper to compute the distribution and moments of k_r's under various conditions.

§ 5. Processing of two sequences

Let $r=2$. Suppose that the conditions stated for $\xi_i^{(j)}$ in the previous paragraph are fulfilled. We wish to determine the mean speed of the processing. Using the notations (2.2) the speed is determined by the sequence of random vectors

$$l(\mathscr{B}^{(1)}), l(\mathscr{B}^{(2)}), \dots, l(\mathscr{B}^{(s)}), \dots .$$

Due to the independence of $\xi_i^{(j)}$'s

$$P(\xi_{i+1}^{(j)} = u | \xi_i^{(j)} = v) = \frac{1}{N}.$$

By using notations (2.5), (2.6) and (2.7) we get

$$Q(\vartheta^{(s)}) = N^{-(k_1^{(s)}+\dots+k_r^{(s)})}. \tag{5.1}$$

So $Q(\vartheta^{(s)})$ depends only on $l(\vartheta^{(s)})$. It is clear, that the condition for the realisability of $\vartheta^{(s)}$, $\vartheta^{(s+1)}$ is $\mathit{fin}(\vartheta^{(s)})=\mathit{in}(\vartheta^{(s+1)})$.

For given $i=(i_1, i_2)$ [or $*i_2$ instead of i_2], $j=(j_1, j_2)$, $k=(k_1, k_2)$ let

$$\mathscr{B}(i,j,k) = \bigcup_{\substack{\text{in}\,\vartheta=i\\ \text{fin}\,\vartheta=j\\ l(\vartheta)=k}} \vartheta. \tag{5.2}$$

Let $\mathscr{E}$ be the set of all elements $\mathscr{B}(i,j,k)$.

The sequences (2.1) and (2.2) determine the sequence

$$\alpha_1, \alpha_2, \dots, \quad \alpha_j \in \mathscr{E} \quad (j = 1, 2, \dots) \tag{5.3}$$

uniquely, where α_j $(j=1, 2, \dots)$ denotes that element of $\mathscr{E}$ for which $\mathscr{B}^{(j)} \in \alpha_j$ $(j=1, 2, \dots)$.

It is clear that the sequence (5.3) is a homogeneous Markov-chain with an initial distribution

$$P(\alpha_1 = \mathscr{B}(i, j, k)) = \frac{1}{N^{k_1+k_2+2}} \nu(i, j, k), \tag{5.4}$$

where $\nu(\alpha)$ or $\nu(i, j, k)$ denotes the number of elements of $\mathscr{D}_r$ belonging to α.

It is clear that

$$P(\alpha_{s+1} = \mathscr{B}(i, j, k) | \alpha_s = C) = \begin{cases} \dfrac{1}{N^{k_1+k_2}} \nu(i, j, k), \\ \text{if} \quad C, \mathscr{B} \quad \text{is realisable} \\ 0, \quad \text{otherwise.} \end{cases} \tag{5.5}$$

For the computation of $\nu(i, j, k)$ we have to distinguish the following cases:

$\langle 1\rangle: k_2 \neq 0,$ then $i_2 \neq *j_2,\ i_2 \neq i_1, j_1$

$$\langle 1.1\rangle: j_1 \neq i_1 - \left[\begin{array}{l} -\langle 1.1.1\rangle: j_2 \neq i_1, j_1, i_2 \\ -\langle 1.1.2\rangle: j_2 = i_1 \\ -\langle 1.1.3\rangle: j_2 = j_1 \\ -\langle 1.1.4\rangle: j_2 = i_2 \end{array}\right.$$

$$\langle 1.2\rangle: j_1 = i_1 - \left[\begin{array}{l} -\langle 1.2.1\rangle: j_2 \neq i_2, i_1 \\ -\langle 1.2.2\rangle: j_2 = i_1 \\ -\langle 1.2.3\rangle: j_2 = i_2 \end{array}\right.$$

$\langle 2\rangle: k_2 = 0,$ then $i_2 = *j_2$

$$\langle 2.1\rangle: j_1 \neq i_1 - \left[\begin{array}{l} -\langle 2.1.1\rangle: j_2 \neq j_1, i_1 \\ -\langle 2.1.2\rangle: j_2 = j_1 \\ -\langle 2.1.3\rangle: j_2 = i_2 \end{array}\right.$$

$$\langle 2.2\rangle: j_1 = i_1 - \left[\begin{array}{l} -\langle 2.2.1\rangle: j_2 \neq i_1 \\ -\langle 2.2.2\rangle: j_2 = i_1. \end{array}\right.$$

We summarize the types, the number of possible different pairs of i's and j's, the corresponding $\nu(i, j, k)$ and $G(\text{type}\langle .,.,.\rangle)$ values in the following table, where

$$G(\text{type}\langle ., .,.\rangle) = \sum_{\mathscr{B}(i,j,k)} \frac{1}{N^{k_1+k_2}} \nu(i, j, k),$$

and we summarize for the $\mathscr{B}$'s of given type.

Type	$v(i, j, k)$	The number of possible pairs i, j	G(type $\langle .,.,. \rangle$)
$\langle 1.1.1 \rangle$	$\frac{(N-4)!}{(N-k_1-k_2)!}(k_1-1)(k_1+k_2-3)$	$N(N-1)(N-2)(N-3)$	$\gamma(k_1+k_2, N)(k_1-1)\cdot \cdot(k_1+k_2-3)$
$\langle 1.1.2 \rangle$	$\frac{(N-3)!}{(N-k_1-k_2)!}(k_1-1)$	$N(N-1)(N-2)$	$\gamma(k_1+k_2, N)(k_1-1)$
$\langle 1.1.3 \rangle$	s. $\langle 1.1.2 \rangle$	s. $\langle 1.1.2 \rangle$	s. $\langle 1.1.2 \rangle$
$\langle 1.1.4 \rangle$	s. $\langle 1.1.2 \rangle$	s. $\langle 1.1.2 \rangle$	s. $\langle 1.1.2 \rangle$
$\langle 1.2.1 \rangle$	$\frac{(N-3)!}{(N-k_1-k_2)!}(k_1+k_2-2)$	s. $\langle 1.1.2 \rangle$	$\gamma(k_1+k_2, N)(k_1+k_2-2)$
$\langle 1.2.2 \rangle$	$\frac{(N-2)!}{(N-k_1-k_2)!}$	$N(N-1)$	$\gamma(k_1+k_2, N)$
$\langle 1.2.3 \rangle$	s. $\langle 1.2.2 \rangle$	s. $\langle 1.2.2 \rangle$	s. $\langle 1.2.2 \rangle$
$\langle 2.1.1 \rangle$	$\frac{(N-3)!}{(N-k_1)!}(k_1-1)(k_1-2)$	$N(N-1)(N-2)$	$\gamma(k_1, N)(k_1-1)(k_1-2)$
$\langle 2.1.2 \rangle$	$\frac{(N-2)!}{(N-k_1)!}(k_1-1)$	$N(N-1)$	$\gamma(k_1, N)(k_1-1)$
$\langle 2.1.3 \rangle$	s. $\langle 2.1.2 \rangle$	s. $\langle 2.1.2 \rangle$	s. $\langle 2.1.2 \rangle$
$\langle 2.2.1 \rangle$	s. $\langle 2.1.2 \rangle$	s. $\langle 2.1.2 \rangle$	s. $\langle 2.1.2 \rangle$
$\langle 2.2.2 \rangle$	$\frac{(N-1)!}{(N-k_1)!}$	N	$\gamma(k_1, N)$

We have got the values in this table by using simple combinatorial considerations. Let us consider e.g. the case $\langle 1.1.1 \rangle$. Let $k_1, k_2, i_1, i_2, j_1, j_2$ be fixed, i_1, i_2, j_1, j_2 be mutually distinct.

We need to enumerate the number of arrays

$$\begin{bmatrix} i_1, & |u_2, & \dots, & u_{k_1}, & \| \\ i_2, & |v_2, & \dots, & v_{k_2}, & \| \end{bmatrix}.$$

It is clear, that $j_1 \in \{u_2, \dots, u_{k_1}\}$. On the other hand $j_2 \in \{u_2, \dots, u_{k_1}\} \cup \{v_2, \dots, v_{k_2}\}$. Let us consider the subcase $j_2 \in \{u_2, \dots, u_{k_1}\}$. We can arrange the elements j_1 and j_2 among $u_2, \dots, u_{k_1}$ in $(k_1-1)(k_1-2)$ different ways. Then we choose the remaining (k_1-3) u's in $\frac{(N-4)!}{(N-4-(k_1-3))!} = \frac{(N-4)!}{(N-k_1-1)!}$ ways. The sequences $v_1, \dots, v_{k_2}$ and $i_1, u_2, \dots, u_{k_1}, i_2$ have no common elements, otherwise they are arbitrary. Therefore we can choose the sequence $v_2, \dots, v_k$ in $\frac{(N-k_1-1)!}{(N-k_1-1-(k_2-1))!} = \frac{(N-k_1-1)!}{(N-k_1-k_2)!}$ ways. Therefore $\frac{(N-4)!}{(N-k_1-k_2)!}$ different processing states belong to this subcase.

Let now $j_2 \in \{v_2, \ldots, v_{k_2}\}$. The set $\{u_2, \ldots, u_{k_1}\}$ contains j_1, but it does not contain i_1, i_2, j_2. Therefore we can choose this set in

$$(k_1-1)\frac{(N-4)!}{(N-4-(k_1-2))!} = (k_1-1)\frac{(N-4)!}{(N-k_1-2)!}$$

ways. The set $\{v_2, \ldots, v_{k_2}\}$ contains j_2, but it doesn't contain the different elements $i_2, i_1, u_2, \ldots, u_{k_1}$, therefore the number of such sets is

$$(k_2-1)\frac{(N-k_1-2)!}{(N-k_1-2-(k_2-2))!} = (k_2-1)\frac{(N-k_1-2)!}{(N-k_1-k_2)!}.$$

Therefore

$$(k_1-1)(k_2-1)\frac{(N-4)!}{(N-k_1-k_2)!}$$

different processing states belong to the second subcase. From here

$$v(i, j, k) = (k_1-1)(k_2-1)\frac{(N-4)!}{(N-k_1-k_2)!}.$$

The proof of the remaining cases is a bit easier.

Due to Lemma 1 the stationary distribution is constructable for the Markov-chain (5.3). Let $u(\alpha)$ denote it.

Let us introduce the notation

$$F(x, y) = \sum_{\substack{\mathrm{fin}\,\alpha=(x,y)\\ \alpha\in\mathscr{E}}} u(\alpha). \tag{5.6}$$

Due to the symmetry

$$F(x, y) = \begin{cases} F(1, 2), & \text{if} \quad x \neq y \\ F(1, 1), & \text{if} \quad x = y. \end{cases} \tag{5.7}$$

Let

$$G(x, y) = \sum_{\substack{\mathrm{in}\,\alpha=(x,y)\\ \alpha=\mathscr{E}}} u(\alpha). \tag{5.8}$$

For the stationary distribution we have

$$\sum_{\alpha_1\in\mathscr{E}} P(\alpha_2|\alpha_1)u(\alpha_1) = u(\alpha_2), \tag{5.9}$$

where

$$\sum_{\alpha\in\mathscr{E}} u(\alpha) = 1. \tag{5.10}$$

Introducing the notation

$$Q(\alpha) = \sum_{\mathscr{B}\in\alpha} Q(\mathscr{B}) \quad (\alpha\in\mathscr{E})$$

we get

$$P(\alpha_2|\alpha_1) = \begin{cases} Q(\alpha_2), & \text{if} \quad in\,\alpha_2 = fin\,\alpha_1, \\ 0, & \text{otherwise.} \end{cases}$$

Then due to (5.9)

$$u(\alpha_2) = Q(\alpha_2) \sum_{\text{fin}\,\alpha_1=\text{in}\,\alpha_2} u(\alpha_1) = Q(\alpha_2)F(in\,\alpha_2). \tag{5.11}$$

Let x and y be arbitrary integers $(1 \leqq x, y \leqq N)$, and $fin\,\alpha_1 = (x, y)$. Then

$$1 = \sum_{\alpha_2 \in \mathscr{E}} P(\alpha_2|\alpha_1) = \sum_{\text{in}\,\alpha_2=(x,y)} Q(\alpha_2). \tag{5.12}$$

Hence, by using (5.11) and (5.12) we get

$$G(x, y) = \sum_{\text{in}\,\alpha=(x,y)} u(\alpha) = F(x, y) \sum_{\text{in}\,\alpha=(x,y)} Q(\alpha) = F(x, y). \tag{5.13}$$

Let

$$\lambda = F(1, 2), \quad \mu = F(1, 1). \tag{5.14}$$

If λ and μ are known, then the stationary distribution can be computed easily by using (5.11).

From (5.10) we have

$$N(N-1)\lambda + N\mu = 1. \tag{5.15}$$

To determine λ and μ we shall give another relation between them.

Let

$$\mathscr{E}_2 = \{\alpha | in\,\alpha = (h, h),\ (h = 1, 2, \ldots, N)\}, \tag{5.16}$$

$$\mathscr{E}_1 = \mathscr{E} \setminus \mathscr{E}_2. \tag{5.17}$$

It is clear that

$$\mu = F(1, 1) = \sum_{\text{fin}\,\alpha=(1,1)} u(\alpha) = \lambda \sum_{\substack{\text{fin}\,\alpha=(1,1)\\ \alpha \in \mathscr{E}_1}} Q(\alpha) + \mu \sum_{\substack{\text{fin}\,\alpha=(1,1)\\ \alpha \in \mathscr{E}_2}} Q(\alpha) = \lambda\alpha + \mu\beta.$$

Let us observe that the α's having the form

$$\begin{bmatrix} h, & |(\ldots)\|1 \\ *, & \|h = 1 \end{bmatrix} = \begin{bmatrix} 1, & |(\ldots)\| \\ & \| \end{bmatrix}$$

are belonging to $\mathscr{E}_2$. These α's are belonging to $\langle 2.2.2 \rangle$. Therefore

$$\beta = \frac{1}{N} \sum_{k=1}^{N} \gamma(k-1, N). \tag{5.18}$$

Let us consider the sum α. We classify the α's in $\mathscr{E}_1$ according to $in\,\alpha = (x, y)$, where $x \neq y$:

Class 1: $y = 1$, then $x \neq 1$. This is the case $\langle 2.1.2 \rangle$.

Class 2: $x = 1$, then $y \neq 1$. This is the case $\langle 1.2.2 \rangle$.

Class 3: $x \neq 1$, $y \neq 1$. This is the case $\langle 1.1.3 \rangle$. Let α_1, α_2, α_3 denote the corresponding sums and let

$$\alpha = \alpha_1 + \alpha_2 + \alpha_3.$$

From the table we can see easily that

$$\alpha_1 = \frac{1}{N}\sum_{t=1}^{N-1} \gamma(t, N)t, \quad \alpha_2 = \alpha_1, \quad \alpha_3 = \frac{1}{N}\sum_{t=1}^{N-1} \gamma(t, N)\frac{t^2-3t+2}{2},$$
$$\alpha = \frac{1}{N}\sum_{t=1}^{N-1} \gamma(t, N)\frac{t^2+t+2}{2}. \tag{5.19}$$

From the system of linear equations

$$\begin{cases} N(N-1)\lambda + N\mu = 1 \\ \lambda\alpha + \mu(\beta-1) = 0 \end{cases} \tag{5.20}$$

we can compute λ and μ.

Let now $f(\alpha)$ be a function depending only on the length of processing (number of processed elements). Let us compute $M_u f(\alpha)$, i.e. supposing the stationarity of (5.3).

Then we get
$$M_u f(\alpha) = \lambda \sum_{\alpha\in\mathcal{E}_1} f(\alpha)Q(\alpha) + \mu \sum_{\alpha\in\mathcal{E}_2} f(\alpha)Q(\alpha). \tag{5.21}$$

Those processing states
$$\begin{bmatrix} i_1, | \dots \| j_1 \\ i_2, | \dots \| j_2 \end{bmatrix}$$

belong to the elements $\alpha\in\mathcal{E}_2$, for which $i_1 = j_2$, $i_2 = *i_1$, i.e. the cases (2.1.3), (2.2.2).

From here

$$\sum_{\alpha\in\mathcal{E}_2} f(\alpha)Q(\alpha) = \sum_{k_1=1}^{N} f(k_1, 0)\cdot k_1 \cdot \gamma(k_1, N), \tag{5.22}$$

$$\sum_{\alpha\in\mathcal{E}_1} f(\alpha)Q(\alpha) = \sum_{k_1=1}^{N} f(k_1, 0)k_1(k_1-1)\gamma(k_1, N) +$$
$$+ \sum_{k_1=1}^{N-1}\sum_{k_2=1}^{N-k_1} f(k_1, k_2)k_1(k_1+k_2)\gamma(k_1+k_2, N). \tag{5.23}$$

Substituting (5.22) and (5.23) into (5.21) we get

$$M_u f(\alpha) = \lambda \sum_{k_1=1}^{N-1}\sum_{k_2=1}^{N-k_1} f(k_1, k_2)k_1(k_1+k_2)\gamma(k_1+k_2, N) +$$
$$+ \sum_{k_1=1}^{N} f(k_1, 0)\gamma(k_1, N)[\lambda k_1(k_1-1) + \mu k_1]. \tag{5.24}$$

§ 6. Asymptotic behaviour of the speed

We compute the asymptotic value of the expression (5.24) as $N\to\infty$ for $f(k_1, k_2)=s_2$. Let M denote the left hand side of (5.24). Then

$$M = \lambda \sum_{1\le s_1<s_2\le N} s_1 s_2^2 \gamma(s_2, N)+\lambda \sum_{s_1=1}^{N} s_1^2(s_1-1)\gamma(s_1, N)+\mu \sum_{s_1=1}^{N} s_1^2\gamma(s_1, N), \quad (6.1)$$

where λ, μ is the solution of

$$\begin{cases} N(N-1)\lambda+N\mu = 1 \\ \alpha\lambda+(\beta-1)\mu = 0 \end{cases} \quad (6.2)$$

and α and β are defined by

$$\alpha = \frac{1}{N}\sum_{t=0}^{N-1} \frac{t^2+t+2}{2}\gamma(t, N), \quad (6.3)$$

$$\beta = \frac{1}{N}\sum_{t=0}^{N-1} \gamma(t, N). \quad (6.4)$$

(6.1) is easily computable approximately from the original expression. We shall give M as a simple function of N. Let

$$\tau_k = \sum_{t=0}^{N-1} t^k\gamma(t, N), \quad (k = 1, 2, 3, 4) \quad (6.5)$$

and

$$\varrho_j = \sum_{t=j}^{N-1} \gamma(t, N) \quad (j = 0, 1, \ldots, 4). \quad (6.6)$$

It is clear that

$$\begin{aligned} \varrho_1 &= \varrho_0-1, \\ \varrho_2 &= \varrho_0-1-\left(1-\frac{1}{N}\right) = \varrho_0-2-\frac{1}{N}, \\ \varrho_3 &= \varrho_2-\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right) = \varrho_0-3-\frac{4}{N}-\frac{2}{N^2}; \\ \varrho_4 &= \varrho_3-\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\left(1-\frac{3}{N}\right) = \varrho_0-4-\frac{10}{N}-\frac{13}{N^2}-\frac{6}{N^3}. \end{aligned} \quad (6.7)$$

Now we compute τ_k's as functions of $\varrho_0, \ldots, \varrho_k$. Because of the definition of $\gamma(t, N)$ we have

$$\gamma(t+1, N) = \gamma(t, N)\left[1-\frac{t+1}{N}\right] \quad (t = 0, 1, \ldots),$$

i.e.

$$\gamma(t+1, N)N = \gamma(t, N)[N-(t+1)], \quad (t = 0, 1, \ldots). \quad (6.8)$$

Hence

$$\gamma(t, N)t = (N-1)\gamma(t, N)-N\gamma(t+1, N) \quad (6.9)$$

and therefore, by using $\gamma(k, N)=0$ (if $k \geqq N$), we get

$$\tau_1 = (N-1)\varrho_0 - N\varrho_1.$$

Let us compute now the polynomial t^k as the sum of the basic functions

$$p_0(t, N) = 1, \quad p_j(t, N) = \prod_{h=1}^{j} (N-(t+h)) \quad (j = 1, \ldots, 4).$$

By simple operations we get

$$t^2 = p_2(t, N) - (2N-3)p_1(t, N) + (N-1)^2 p_0(t, N), \tag{6.10}$$

$$t^3 = -p_3(t, N) + (3N-6)p_2(t, N) - (3N^2-9N+7)p_1(t, N) + (N-1)^3 p_0(t, N), \tag{6.11}$$

$$t^4 = p_4(t, N) + Ep_3(t, N) + Fp_2(t, N) + Gp_1(t, N) + Hp_0(t, N), \tag{6.12}$$

where

$$E = -4N+10, \quad F = 6N^2-24N+25,$$

$$G = -4N^3+18N^2-28N+15, \quad H = (N-1)^4. \tag{6.13}$$

On the other hand because of (6.8)

$$\gamma(t, N)p_k(t, N) = N^k \gamma(t+k, N), \quad (k = 0, \ldots, 4). \tag{6.14}$$

So we have

$$\gamma(t, N) \cdot t^2 = N^2 \gamma(t+2, N) - (2N-3)N\gamma(t+1, N) + (N-1)^2 \gamma(t, N),$$

$$\gamma(t, N) \cdot t^3 = -N^3 \gamma(t+3, N) + (3N-6)\gamma(t+2, N) - (3N^2-9N+7)N\gamma(t+1, N) +$$
$$+ (N-1)^3 \gamma(t, N),$$

$$\gamma(t, N) \cdot t^4 = N^4 \gamma(t+4, N) + E \cdot N^3 \gamma(t+3, N) + FN^2 \gamma(t+2, N) +$$
$$+ GN\gamma(t+1, N) + H\gamma(t, N),$$

and hence

$$\begin{aligned} \tau_2 &= N^2 \varrho_2 - (2N-3)N\varrho_1 + (N-1)^2 \varrho_0 = (N+1)\varrho_0 - 2N, \\ \tau_3 &= -N^3 \varrho_3 + (3N-6)N^2 \varrho_2 - (3N^2-9N+7)\varrho_1 + (N-1)^3 \varrho_0 = \\ &= (2N^2+3N) - (4N+1)\varrho_0, \\ \tau_4 &= N^4 \varrho_4 + EN^3 \varrho_3 + FN^2 \varrho_2 + GN_1 \varrho_1 + H\varrho_0 = (3N^2+11N)\varrho_0 - (11N^2+4N). \end{aligned} \tag{6.15}$$

Now it follows from (6.3) and (6.4)

$$\beta = \frac{1}{N}\varrho_0, \quad \alpha = \frac{1}{2N}(\tau_2+\tau_1+2\varrho_0) = \left(\frac{1}{2}+\frac{1}{N}\right)\varrho_0 - \frac{1}{2}. \tag{6.16}$$

By substituting (6.16) into (6.2) we get

$$\mu = \frac{\alpha}{N\alpha + N(N-1)(1-\beta)} = \frac{(N+2)\varrho_0 - N}{(2N^3-3N^2)+(4N-N^2)\varrho_0} \tag{6.17}$$

and

$$\lambda = \frac{1-\beta}{N\alpha + N(N-1)(1-\beta)} = \frac{2N-2\varrho_0}{(2N^3-3N^2)+(4N-N^2)\varrho_0}. \tag{6.18}$$

Let us observe that

$$M = \lambda \sum_{s_2=2}^{N} \frac{s_2(s_2-1)s_2^2}{2}\gamma(s_2, N) + \lambda \sum_{s_1=1}^{N} s_1^2(s_1-1)\gamma(s_1, N) +$$

$$+\mu \sum_{s_1=1}^{N} s_1^2\gamma(s, N) = \lambda\left[\frac{1}{2}(\tau_4-\tau_3)+(\tau_3-\tau_2)\right]+\mu\tau_2 = \lambda\left[\frac{1}{2}(\tau_4+\tau_3)-\tau_2\right]+\mu\tau_2. \tag{6.19}$$

Substituting (6.15), (6.17) and (6.18) into (6.19) we get M as a function of N and ϱ_0:

$$M = \frac{N^3(3\varrho_0-9)-N^2(2\varrho_0^2-11\varrho_0+9)-N(2\varrho_0^2+7\varrho_0)+5\varrho_0^2}{2N^3-N^2(\varrho_0+3)+4N\varrho_0}. \tag{6.20}$$

To estimate this expression we need the following

Lemma 2.

$$\varrho_0 = \sum_{t=0}^{N-1}\gamma(t, N) = \sqrt{\frac{\pi N}{2}} + O(1) \quad (N\to\infty). \tag{6.21}$$

Proof. Since $1-x \leqq e^{-x}$, we get

$$\gamma(t, N) = \prod_{\nu=1}^{t}\left(1-\frac{\nu}{N}\right) \leqq e^{-\frac{1}{N}\sum_{\nu=1}^{t}\nu} < e^{-\frac{t^2}{2N}}. \tag{6.22}$$

Therefore

$$\sum_{H\leqq t<N}\gamma(t, N) < \sum_{t\geqq H} e^{-\frac{t^2}{2N}} < \int_{H-1}^{N} e^{-\frac{t^2}{2N}}dt = \frac{1}{2}\sqrt{2N}\int_{\frac{(H-1)^2}{2N}}^{\frac{N}{2}} e^{-\lambda}\lambda^{1/2}d\lambda <$$

$$< \frac{1}{2}\sqrt{2N}\frac{\sqrt{2N}}{H-1}\int_{\frac{(H-1)^2}{2N}}^{\infty} e^{-\lambda}d\lambda = \frac{N}{H-1}e^{-\frac{(H-1)^2}{2N}}. \tag{6.23}$$

On the other hand

$$\sum_{t\leqq H} e^{-\frac{t^2}{2N}} \leqq \int_0^H e^{-\frac{t^2}{2N}}dt < \sqrt{2N}\frac{1}{2}\int_0^{\infty} e^{-\lambda}\lambda^{1/2}d\lambda \leqq C_1\sqrt{N}. \tag{6.24}$$

It is clear that

$$\varrho_0 = \sum_{t\leqq N^{0,6}}\gamma(t, N) + \sum_{N^{0,6}<t\leqq N}\gamma(t, N) = \Sigma_A + \Sigma_B. \tag{6.25}$$

Because of (6.23)

$$\Sigma_B \leqq C_2 N^{0,4} \cdot e^{-\frac{1}{3}N^{0,2}} = o(1). \tag{6.26}$$

On the other hand by using Stirling-formula for $\gamma(t, N)$ in the interval $1 \leqq t \leqq \leqq N^{0,6}$ we get

$$\log \gamma(t, N) = \left(N - \frac{1}{2} - t\right) \log \frac{N}{N-t} - t + O\left(\frac{1}{N}\right). \tag{6.27}$$

Since

$$\log \frac{N}{N-t} = -\log\left(1 - \frac{N}{t}\right) = \frac{t}{N} + \frac{t^2}{2N^2} + O\left(\left(\frac{t}{N}\right)^3\right), \tag{6.28}$$

therefore

$$\log \gamma(t, N) = -\frac{t^2}{2N} + O\left(\frac{t}{N} + \frac{t^3}{N^2}\right), \tag{6.29}$$

and so

$$\Sigma_A = \sum_{t \leqq N^{0,6}} e^{-\frac{t^2}{2N}}\left(1 + O\left(\frac{t}{N} + \frac{t^3}{N^2}\right)\right) = \sum_{t \leqq N^{0,6}} e^{-\frac{t^2}{2N}} + \frac{1}{N} O(\Sigma_C) + \frac{1}{N^2} O(\Sigma_D), \tag{6.30}$$

where

$$\Sigma_C = \sum_{t \leqq N^{0,6}} t \cdot e^{-\frac{t^2}{2N}}, \tag{6.31}$$

$$\Sigma_D = \sum_{t \leqq N^{0,6}} t^3 e^{-\frac{t^2}{2N}}. \tag{6.32}$$

Since

$$\sum_{t=1}^{\infty} t^\alpha e^{-\frac{t^2}{2N}} < \int_0^\infty t^\alpha e^{-\frac{t^2}{2N}} dt = (2N)^{\frac{\alpha+1}{2}} \frac{1}{2} \int_0^\infty \lambda^{\frac{\alpha-1}{2}} e^{-\lambda} d\lambda = = (2N)^{\frac{\alpha+1}{2}} \frac{1}{2} \Gamma\left(\frac{\alpha+1}{2}\right), \tag{6.33}$$

therefore

$$\Sigma_C = O(N), \quad \Sigma_D = O(N^2), \tag{6.34}$$

and so

$$\varrho_0 = \sum_{1 \leqq t \leqq N^{0,6}} e^{-\frac{t^2}{2N}} + O(1) = \sum_{t=1}^{\infty} e^{-\frac{t^2}{2N}} + O(1). \tag{6.35}$$

Since $e^{-\frac{t^2}{2N}}$ is a monoton decreasing function of t, therefore

$$\int_0^\infty e^{-\frac{t^2}{2N}} dt < \sum_{t=1}^{\infty} e^{-\frac{t^2}{2N}} < \int_1^\infty e^{-\frac{t^2}{2N}} dt, \tag{6.36}$$

and so

$$\varrho_0 = \int_0^\infty e^{-\frac{t^2}{2N}} dt + O(1). \tag{6.37}$$

Since

$$\int_0^\infty e^{-\frac{t^2}{2N}}\,dt = \frac{1}{2}\sqrt{2N}\int_0^\infty e^{-\lambda}\lambda^{1/2}\,d\lambda = \frac{\sqrt{2}}{2}\Gamma\left(\frac{1}{2}\right)\sqrt{N} = \sqrt{\frac{\pi}{2}}\sqrt{N}, \tag{6.38}$$

therefore

$$\varrho_0 = \sqrt{\frac{\pi}{2}}\sqrt{N} + O(1). \tag{6.39}$$

By substituting (6.39) into (6.20) we get the following

Teorem 3. Let $f(k_1, k_2) = k_1 + k_2$. Then under the assumptions of § 5. we have

$$\lim_{N\to\infty} N^{-0,5} M_u f(\alpha) = \frac{3}{2}\sqrt{\frac{\pi}{2}}. \tag{6.40}$$

In a previous paper [2] we have proved that the similar limit is $\sqrt{\frac{\pi}{2}}$ for the processing speed of one sequence. Comparing the results we get that the processing speed of the second sequence is half of the speed of the first one.

DEPT. OF COMPUTER SCIENCE
OF L. EÖTVÖS UNIVERSITY
MÚZEUM KRT. 6—8.
BUDAPEST, HUNGARY
H—1088

References

[1] BURNETT, G. J. and E. G. COFFMAN, Combinatorial problem related to interleaved memory systems, *J. Assoc. Comput. Mach.*, v. 20, 1973, pp. 39—46.

[2] IVÁNYI, A. and I. KÁTAI, Estimates for speed of computers with interleaved memory systems, *Ann. Univ. Sci. Budapest. Eötvös Sect. Mat.*, v. 19, 1976, pp. 159—164.

(Received August 7, 1978)

Über das Rechnen mit den Elementen abstrakt präsentierter Halbgruppen

Von H. JÜRGENSEN

Herrn Professor László Kalmár zum Gedächtnis

$X \neq \emptyset$ sei eine endliche Menge, X^+ die von X erzeugte freie Halbgruppe. R sei eine endliche Menge von Gleichungen (Relationen) über X^+. $\varrho_{(X,R)}$ sei die von R erzeugte Kongruenz auf X^+. Dann ist $S_{(X,R)} = X^+/\varrho_{(X,R)}$ die durch (X, R) präsentierte Halbgruppe. In der vorliegenden Arbeit geht es darum, eine Klasse $\mathfrak{N}$ von endlichen „normierten" Präsentationen anzugeben, für die gilt:

(1) $\mathfrak{N}$ wird „modulo Gruppen" formal beschrieben.

(2) Jede endliche Halbgruppe besitzt in $\mathfrak{N}$ eine Präsentation.

(3) Für $(X, R) \in \mathfrak{N}$ ist die durch (X, R) präsentierte Halbgruppe endlich.

(4) Aus der Beschreibung von $\mathfrak{N}$ läßt sich „modulo Gruppenelementen" eine formale Beschreibung „normierter Wörter" angeben, so daß für jede Präsentation $(X, R) \in \mathfrak{N}$ jede $\varrho_{(X,R)}$-Klasse ein normiertes Wort enthält.

(5) Zu jeder endlichen Halbgruppe gibt es eine Präsentation $(X, R) \in \mathfrak{N}$, so daß jede $\varrho_{(X,R)}$-Klasse genau ein normiertes Wort enthält.

(6) Aus der Beschreibung von $\mathfrak{N}$ und der normierten Wörter läßt sich ein „modulo Gruppenelementen" leicht programmierbarer, recht effizienter Algorithmus zum Normieren von Wörtern und damit zum Rechnen mit den normierten Wörtern angeben.

Wie schon in diesen Forderungen formulieren wir die meisten Aussagen und den Algorithmus zunächst nur „modulo Gruppen", d. h. unter Verwendung von Orakeln für das Rechnen mit Gruppen. Im Abschnitt 3 geben wir dann einige Hinweise auf Realisierungsmöglichkeiten unter der Voraussetzung spezieller Gruppenpräsentationen.

1. Normierte Präsentationen und normierte Wörter

1.1. Definition. (X, R) sei eine Halbgruppenpräsentation. (X, R) ist eine *normierte Präsentation*, wenn gilt:

(1) (X, R) ist endlich.

(2) X besitzt eine Partition in Mengen $X_0, X_1, \ldots, X_n$ mit $X_i = \{c^i\}$ oder $X_i = \{a_0^i, \ldots, a_{k_i}^i, b_0^i, \ldots, b_{m_i}^i\} \cup E_i$ für $i = 1(1)n$ und $X_0 = \{a_0^0, \ldots, a_{k_0}^0, b_0^0, \ldots, b_{m_0}^0\} \cup E_0$ mit $k_0, m_0, k_i, m_i \in \mathbf{N} \cup \{0\} = \mathbf{N}_0$ und $E_0, E_i \neq \emptyset$. Sei $X^i = \bigcup_{j=0}^{i} X_j$.

(3) R enthält genau die folgenden Relationen (jeweils für alle Indizes, für die die Symbole erklärt sind):
(a) Mengen S_i von Relationen über E_i^+, so daß (E_i, S_i) Halbgruppenpräsentation einer endlichen Gruppe G_i ist. $e^i \in E_i^+$ sei das Einselement von G_i.
(b) $a_0^i = e^i = b_0^i$.
(c) $a_j^i e^i = a_j^i$.
(d) $e^i b_j^i = b_j^i$.
(e) $b_j^i a_l^i = p_{jl}^i \in E_i^+ \cup (X^{i-1})^+$ für $(j, l) \neq (0, 0)$.
(f) $xy = q_{x,y} \in (X^{\min(i,j)})^+$ für $x \in X_i$, $y \in X_j$, $i \neq j$.
(g) $c^i c^i = r^i \in (X^{i-1})^+$.

Mit $\mathfrak{N}$ sei die Klasse aller normierten Präsentationen bezeichnet. Die offenbar überflüssigen a_0^i, b_0^i dienen nur zur Vermeidung von Fallunterscheidungen.

1.2. Lemma. Sei $(X, R) \in \mathfrak{N}$, $\varrho = \varrho_{(X,R)}$. Jede ϱ-Klasse von X^+ enthält ein Wort der Form c^i oder $a_j^i g^i b_k^i$ mit $g^i \in E_i^+$.

Beweis. $w = x_1 \ldots x_v$ sei ein Wort mit $x_1, \ldots, x_v \in X$. Sei zunächst $v = 1$. Falls $w \neq c^i$ ist, gilt
$$w = a_j^i \varrho a_j^i e^i b_0^i$$
oder
$$w = b_j^i \varrho a_0^i e^i b_j^i$$
oder
$$w = g^i \varrho e^i g^i e^i \varrho a_0^i g^i b_0^i \quad \text{mit} \quad g^i \in E_i.$$
Sei also jetzt $v > 1$. $\alpha(w)$ sei das Maximum der oberen Indizes der Symbole in w, $\beta(w)$ das Minimum. $\gamma(w)$ sei die Anzahl der Symbole in w mit oberem Index $\alpha(w)$. Wir unterscheiden zwei Fälle:

$\alpha(w) \neq \beta(w)$: Dann gibt es in w x_j, x_{j+1} mit $x_j \in X_{\alpha(w)}$, $x_{j+1} \notin X_{\alpha(w)}$ oder umgekehrt. Anwendung von (f), d. h. Ersetzen von $x_j x_{j+1}$ durch $q_{x_j, x_{j+1}}$ ergibt ein zu w ϱ-äquivalentes Wort w' mit $\alpha(w) = \alpha(w')$, $\gamma(w) > \gamma(w') \geqq 1$ oder $\alpha(w) > \alpha(w')$. Mit endlich vielen Anwendungen von (f) erhält man daher so ein zu w ϱ-äquivalentes Wort w'' mit $\alpha(w'') = \beta(w'') \leqq \beta(w)$.

$\alpha(w) = \beta(w)$: Falls $X_{\alpha(w)} = \{c^{\alpha(w)}\}$ ist, ist w wegen (g) zu
$$w' = r^{\alpha(w)} \underbrace{c^{\alpha(w)} \ldots c^{\alpha(w)}}_{v-2 \text{ mal}}$$
ϱ-äquivalent; dabei ist für $v > 2$ $\alpha(w) = \alpha(w') \neq \beta(w')$ und für $v = 2$ $\alpha(w') < \alpha(w)$. Durch endlich viele Anwendungen von (f) und (g) erhält man also ein zu w ϱ-äquivalentes Wort w' mit $\alpha(w') = \beta(w')$ und $X_{\alpha(w')} \neq \{c^{\alpha(w')}\}$ oder $|w'| = 1$. Wir können dies also schon für w voraussetzen. Durch endlich viele Anwendungen von (b) bis (e), nämlich durch Ersetzen
von $b_j^i a_l^i$ für $(j, l) \neq (0, 0)$ durch p_{jl}^i,
von $b_0^i a_0^i$ durch e^i,
von $h^i a_j^i$ durch $h^i p_{0j}^i$, von $b_j^i h^i$ durch $p_{j0}^i h^i$ für $h^i \in E_i$, $j \neq 0$,
von $a_j^i a_l^i$ durch $a_j^i p_{0l}^i$ für $l \neq 0$ und durch a_j^i für $l = 0$,
von $b_j^i b_l^i$ durch $p_{j0}^i b_l^i$ für $j \neq 0$ und durch b_l^i für $j = 0$,

erhält man ein zu w ϱ-äquivalentes Wort w' mit $\beta(w')<\alpha(w')$ oder $w'\in E_i^+$ oder $w'\in a_j^i E_i^+$ oder $w'\in E_i^+ b_1^i$ oder $w'\in a_j^i E_i^+ b_1^i$. Im ersten Falle schließt man für w' statt w wie oben weiter. In den übrigen Fällen hat $a_0^i w' b_0^i$, $w' b_0^i$, $a_0^i w'$, bzw. w' die gewünschte Form und ist zu w ϱ-äquivalent wegen (b). □

1.3. Lemma. Für $(X, R)\in\mathfrak{N}$ hat $S_{(X,R)}$ höchstens die Ordnung

$$\delta(X, R) = c+\Sigma\,|G_i|(k_i+1)(m_i+1),$$

wobei die Summation über die i mit $|X_i|\neq 1$ durchgeführt wird und c die Anzahl der i mit $|X_i|=1$ ist.

Beweis. Es gibt höchstens c Wörter der Form c^i, und wegen 1.1. (3a) gibt es zu festen j, l höchstens $|G_i|$ paarweise nicht-äquivalente Wörter der Form $a_j^i g^i b_l^i$ mit $g^i\in E_i^+$. Damit gibt es höchstens $\delta(X, R)$ paarweise nicht-äquivalente Wörter dieser Formen, und aus 1.2 folgt die Behauptung. □

1.4. Definition. Sei $(X, R)\in\mathfrak{N}$. Zu jedem vorkommenden Paar (E_i, S_i) sei ein Repräsentantensystem $\{s^i\}$ der $\varrho_{(E_i,S_i)}$-Klassen beliebig, aber fest gewählt. Die Wörter der Form c^i und $a_j^i s^i b_l^i$ heißen *normiert.*

Zusammenfassend erhält man:

1.5. Satz. Die durch $(X, R)\in\mathfrak{N}$ präsentierte Halbgruppe ist endlich. Sie hat genau dann die Ordnung $\delta(X, R)$, wenn jede $\varrho_{(X,R)}$-Klasse genau ein normiertes Wort enthält.

Aus dem Beweis von 1.2 folgt weiter, indem man ein Orakel für das Rechnen mit Gruppenelementen voraussetzt:

1.6. Satz. Sei $(X, R)\in\mathfrak{N}$ und $\delta(X, R)=|S_{(X,R)}|$. Es gibt „modulo Gruppen" einen Algorithmus, der zu jedem Wort das $\varrho_{(X,R)}$-äquivalente normierte Wort berechnet.

Einen allgemeinen Beweis der Entscheidbarkeit des Wortproblems „modulo Gruppen" für beliebige normierte Halbgruppenpräsentationen (d. h. ohne die Forderung $\delta(X, R)=|S_{(X,R)}|$) erhält man wegen 1.5 aus [5], wo wir das Todd—Coxeter-Verfahren auf Halbgruppen übertragen haben. Der daraus resultierende Algorithmus für das Wortproblem ist jedoch sehr aufwendig.

Die Umkehrung von 1.5 erhält man mit Hilfe bekannter Struktursätze für endliche Halbgruppen [z. B. 1]:

1.7. Satz. Jede endliche Halbgruppe S besitzt eine normierte Präsentation (X, R) mit $\delta(X, R)=|S|$.

Beweis. S sei eine endliche Halbgruppe. S hat eine Kompositionsreihe

$$\Sigma_0 \subseteqq \Sigma_1 \subseteqq \ldots \subseteqq \Sigma_n = S,$$

wobei Σ_0 einfach und Σ_{i+1}/Σ_i 0-einfach oder 0 von der Ordnung 2 ist. Die Behauptung wird durch Induktion nach n bewiesen.

$n=0$: $S=\Sigma_0$ ist als endliche einfache Halbgruppe vollständig einfach und daher Rechteckhalbgruppe isomorpher Gruppen H_{jl} mit $j=0(1)k_0$, $l=0(1)m_0$. (E_0, S_0)

sei eine endliche Halbgruppenpräsentation von $H_{00}=G_0$, und $e^0 \in E_0^+$ sei das Einselement von G_0. Sei $a_0^0=e^0=b_0^0$ und $a_j^0 \in H_{j0}$, $b_l^0 \in H_{0l}$ beliebig für $j, l \geqq 1$. Sei

$$X_0 = \{a_0^0, a_1^0, \ldots, a_{k_0}^0\} \cup \{b_0^0, b_1^0, \ldots, b_{m_0}^0\} \cup E_0.$$

Die so gewählten Elemente erfüllen die Bedingungen aus 1.1 für $X=X_0$: (3a), (3b) gelten nach Wahl von X_0. Es ist $a_j^0 H_{00}=H_{j0}$ und daher $a_j^0 g=a_j^0$ für ein $g \in H_{00}$, also

$$a_j^0 e^0 = a_j^0 g e^0 = a_j^0 g = a_j^0$$

und daher (3c). Analog folgt (d). Wegen $H_{0l}H_{j0}=H_{00}$ folgt (e), wenn man für p_{lj}^0 das Produkt $b_l^0 a_j^0$ wählt. (f) und (g) sind leer. S ist daher homomorphes Bild der durch (X, R) mit

$$R = S_0 \cup \left\{ a_0^0 = e^0 = b_0^0,\ a_j^0 e^0 = a_j^0,\ e^0 b_j^0 = b_j^0,\ b_l^0 a_j^0 = p_{lj}^0 \,\middle|\, \begin{array}{l} j = 0(1)k_0 \\ l = 0(1)m_0 \\ (j, l) \neq (0, 0) \end{array} \right\}$$

präsentierten Halbgruppe $S_{(X,R)}$. Wegen

$$|S| = |H_{00}|(k_0+1)(m_0+1) = \delta(X, R)$$

ist $S \cong S_{(X,R)}$.

Die Behauptung sei nun für alle Halbgruppen mit Kompositionsketten der Länge $\leqq n-1$ für $n \geqq 1$ bewiesen. S sei eine endliche Halbgruppe mit Kompositionskette der Länge $n \geqq 1$: Für Σ_{n-1} sei eine normierte Präsentation (X', R') gegeben. Wir unterscheiden zwei Fälle:

Σ_n/Σ_{n-1} *ist Nullhalbgruppe:* $c^n \in \Sigma_n/\Sigma_{n-1}$ sei das von 0 verschiedene Element, $X_n=\{c^n\}$, $X=X' \cup X_n$,

$$R = R' \cup \{c^n c^n = r^n,\ x c^n = q_{x,c^n},\ c^n x = q_{c^n,x} | x \in X'\}.$$

Dabei sind $r^n, q_{x,c^n}, q_{c^n,x}$ Darstellungen der entsprechenden Produkte in S, die in $X'^+=(X^{n-1})^+$ gewählt werden können, weil sie in Σ_{n-1} liegen. Damit ist $(X, R) \in \mathfrak{N}$ mit $\delta(X, R)=|S|$. Weil R in S gilt, wird S durch (X, R) präsentiert.

Σ_n/Σ_{n-1} *ist 0-einfach:* Mit $j=0(1)k_n$, $l=0(1)m_n$ seien die $\mathscr{R}$-Klassen und $\mathscr{L}$-Klassen von Σ_n/Σ_{n-1} o. B. d. A. so indiziert, daß die $\mathscr{H}$-Klasse H_{00} eine Gruppe ist. (E_n, S_n) sei eine endliche Halbgruppenpräsentation der Gruppe $G_n=H_{00}$, und e^n sei eine Darstellung ihres Einselementes. Sei

$$X_n = \{a_0^n, a_1^n, \ldots, a_{k_n}^n\} \cup \{b_0^n, b_1^n, \ldots, b_{m_n}^n\} \cup E_n,$$

wo $a_j^n \in H_{j0}$, $b_l^n \in H_{0l}$ beliebig und $a_0^n=b_0^n=e^n$ gewählt werden. (3c) und (3d) gelten wie oben. Für $(l, j) \neq (0, 0)$ liegt das Produkt $b_l^n a_j^n$ in H_{00} oder in Σ_{n-1}, kann also als

$$p_{lj}^n \in E_n^+ \cup (X^{n-1})^+$$

dargestellt werden. Damit gilt (3e). Die $q_{x,y}$ mit $x \in X_n$ oder $y \in X_n$ können in $(X^{n-1})^+$ gewählt werden, weil die entsprechenden Produkte in Σ_{n-1} liegen. Also gilt auch (f). (g) ist leer. S ist daher homomorphes Bild der durch (X, R) mit $X = X' \cup X_n$ und

$$R = R' \cup \left\{ a_0^n = e^n = b_0^n,\ a_j^n e^n = a_j^n,\ e^n b_j^n = b_j^n,\ b_l^n a_j^n = p_{lj}^n \;\middle|\; \begin{array}{l} j = 0(1)k_0 \\ l = 0(1)m_0 \\ (j, l) \neq (0, 0) \end{array} \right\}$$

$$\cup \{xy = q_{x,y} \mid (x, y) \in (X \times X) \setminus (X_n \times X_n)\}$$

präsentierten Halbgruppe $S_{(X,R)}$. Wegen

$$\begin{aligned} |S| &= |\Sigma_{n-1}| + |\Sigma_n / \Sigma_{n-1}| - 1 \\ &= \delta(X', R') + |H_{00}|(k_n + 1)(m_n + 1) \\ &= \delta(X, R) \end{aligned}$$

gilt $S \cong S_{(X,R)}$. □

Die normierten Präsentationen bestimmen also genau die endlichen Halbgruppen, und jede endliche Halbgruppe besitzt sogar eine solche normierte Präsentation, bei der jedes Element durch genau ein normiertes Wort dargestellt wird. Dieser Fall ist unter algorithmischen Gesichtspunkten besonders interessant, weil sich dann sämtliche Rechenoperationen mit den Elementen der Halbgruppe auf das Bestimmen der zugehörigen normierten Wörter zurückführen lassen. Einer normierten Präsentation (X, R) kann man es jedoch im allgemeinen nicht ansehen, ob $S_{(X,R)}$ die Ordnung $\delta(X, R)$ hat. Einige Kriterien ergeben sich aus Sätzen über Idealerweiterungen von Halbgruppen [z. B. 9]. Algorithmisch läßt sich diese Frage „modulo Gruppen" z. B. mit dem Programm aus [5] lösen.

Es ist noch — insbesondere hinsichtlich der im weiteren zu behandelnden algorithmischen Fragen — zu bemerken, daß in 1.6 vorausgesetzt wird, daß von der Präsentation (X, R) nicht nur die Normiertheit, sondern auch die Partition in die X_i und E_i bekannt ist. Dies ist wegen des folgenden Satzes wichtig:

1.8. Satz. Es ist unentscheibdar, ob eine endliche Halbgruppenpräsentation normiert ist. Setzt man ein Orakel zur Entscheidung der Frage „definiert eine endliche Halbgruppenpräsentation eine endliche Gruppe?" voraus, so wird die Normiertheit für endliche Halbgruppenpräsentationen entscheidbar.

Beweis. Bekanntlich ist die Endlichkeit präsentierter Halbgruppen oder Gruppen unentscheidbar. Sei also (X', R') eine beliebige endliche Grüppenpräsentation. Durch Hinzunahme der Inversen erhält man in kanonischer Weise eine Halbgruppenpräsentation derselben Gruppe. Mit dem Beweis von 1.7 konstruiert man daraus eine Präsentation (X, R) dieser Gruppe, die genau dann normiert ist, wenn die Gruppe endlich ist. Damit ist die Normiertheit unentscheidbar. Setzt man jedoch ein Orakel für die genannte Frage voraus, so kann man die Normiertheit einer Präsentation folgendermaßen entscheiden: Für jede Partition von X gemäß 1.1 (2) prüfe man

(a) 1.1 (3b) bis (3g),
(b) ob die übrigen Relationen sich zu Mengen S_i über den E_i^+ aufteilen lassen,
(c) ob die (E_i, S_i) endliche Gruppen präsentieren,
(d) ob das durch (a) gegebene $e^i \in E_i^+$ Einselement der entsprechenden Gruppe ist.

Davon sind (a) und (b) formal zu entscheiden, (c) erhält man vom Orakel, (d) ist entscheidbar (z. B. mit [5]), wenn (c) bejaht wird. □

Selbstverständlich kann man das Orakel von 1.8 durch geeignete formale Voraussetzungen über die Präsentationen der Gruppen vermeiden. Auf diese Frage kommen wir im Abschnitt 3 zurück.

2. Der Multiplikationsalgorithmus

Sei $(X, R) \in \mathfrak{N}$ zusammen mit der Partition von X gemäß 1.1 gegeben, und sei $\delta(X, R) = |S_{(X,R)}|$. Wir formulieren „modulo Gruppen" einen Algorithmus, der zu zwei normierten Wörtern $u, v \in X^+$ das zu ihrem Produkt uv $\varrho_{(X,R)}$-äquivalente normierte Wort berechnet.

Wegen 1.2 kann man „modulo Gruppen" die zu den p_{lj}^i, $q_{x,y}$, r^i $\varrho_{(X,R)}$-äquivalenten normierten Wörter berechnen. Indem man in R die p_{lj}^i, $q_{x,y}$, r^i durch die entsprechenden normierten ersetzt, erhält man — „modulo Gruppen" effektiv — eine normierte Präsentation für $S_{(X,R)}$, in der die rechten Seiten zu 1.1 (3e—g) normiert sind. Mann kann also, und dies soll im folgenden geschehen, o. B. d. A. voraussetzen, daß R selbst schon diese Gestalt hat. Durch diese theoretisch irrelevante Forderung wird die Lösung der obigen Aufgabe sehr vereinfacht.

Zu $w \in X^+$ sei $\bar{w}$ das $\varrho_{(X,R)}$-äquivalente normierte Wort und $\tau(w)$ der Index i mit $\bar{w} \in X_i^+$. Für $|X_{\tau(w)}| = 1$ ist somit $\bar{w} = c^{\tau(w)}$; andernfalls hat $\bar{w}$ die Form

$$a_{\lambda(w)}^{\tau(w)} g_{w_1}^{\tau(w)} \dots g_{w_{\mu(w)}}^{\tau(w)} b_{\nu(w)}^{\tau(w)}$$

mit $g_{w_i}^{\tau(w)} \in E_{\tau(w)}$.

Die Beschreibung des Multiplikationsalgorithmus MULT erfolgt in einer leicht programmierbaren und (hoffentlich) aus sich verständlichen Weise. Er besteht neben wenigen elementaren Anweisungen aus zahlreichen Aufrufen von Unterprogrammen und Verteilersprüngen auf Marken, die in der Form

⟨Name⟩[⟨Parameterliste⟩] (⟨Argumentenliste⟩)

bzw.

⟨Name⟩[⟨Parameterliste⟩]

geschrieben werden. Die zunächst wohl künstlich anmutende Unterscheidung zwischen Parametern und Argumenten dient dazu, Programmverzweigungen, die nach Kenntnis der Präsentation unabhängig von den zu multiplizierenden Elementen feststehen, und solche, die von den jeweils zu multiplizierenden Elementen abhängen, zu trennen. Damit bereiten wir die spätere zweistufige Programmrealisierung von MULT vor, bei der ähnlich [2, 3, 6, 7, 8] aus (X, R) in einem Vorbereitungsschritt das eigentliche Multiplikationsprogramm erst berechnet wird. Aus diesem Grunde verzichten wir auch auf formale Vereinfachungen und Zusammenfassungen, die für diese Realisierung nur hinderlich wären. Es folgt die Definition des Algorithmus:

```
MULT $(u, v)$:                 Voraussetzung: $u$, $v$ normiert.
                               Wirkung: $\overline{uv}$ berechnen.
    VERTEILERSPRUNG AUF $M[\tau(u), \tau(v)]$.
$M[i, j]$:                     Für $i, j = 0(1)n$.
        Fall 1: $|X_i| = 1,\ i = j$.
    RÜCKSPRUNG MIT $r^i$.
        Fall 2: $|X_i| = |X_j| = 1,\ i \neq j$.
    RÜCKSPRUNG MIT $q_{c^i, c^j}$.
        Fall 3: $|X_i| = 1 \neq |X_j|$.
    $v := C[i, j](v)$
    RÜCKSPRUNG MIT $v$.
        Fall 4: $|X_i| \neq 1$.
    $v := B[i, j](\nu(u), v)$
    $v := E[i](g^i_{u_{\mu(u)}}, v)$
     ⋮
    $v := E[i](g^i_{u_1}, v)$
    $v := A[i](\lambda(u), v)$
    RÜCKSPRUNG MIT $v$.


$B[i, j](l, v)$:               Für $i, j = 0(1)n$ mit $|X_i| \neq 1$.
                               Voraussetzung: $v$ normiert, $v \in X_j^+$.
                               Wirkung: $\overline{b_l^i v}$ berechnen.
        Fall 1: $|X_j| = 1$.
    RÜCKSPRUNG MIT $q_{b_l^i, c^j}$.
        Fall 2: $|X_j| \neq 1,\ i = j$.
    IST $l = \lambda(v) = 0$?
    WENN JA: RÜCKSPRUNG MIT $v$
    SONST: VERTEILERSPRUNG AUF $BA[i, l, j, \lambda(v)]$.
        Fall 3: $|X_j| \neq 1,\ i \neq j$.
    VERTEILERSPRUNG AUF $BA[i, l, j, \lambda(v)]$.
$BA[i, l, j, k]$:              Für $i, j = 0(1)n$ mit $|X_i| \neq 1 \neq |X_j|$ und für $l = 0(1)m_i$,
                               $k = 0(1)k_j$ und $(l, k) \neq (0, 0)$ bei $i = j$.
        Fall 1: $i = j,\ x = p^i_{lk} \in a^i_0 E_i^+ b^i_0$.
    $g := GF[i, g^i_{x_{\mu(x)}}](g^i_{v_1} \cdots g^i_{v_{\mu(v)}})$
    $g := GF[i, g^i_{x_{\mu(x)-1}}](g)$
     ⋮
    $g := GF[i, g^i_{x_1}](g)$
    RÜCKSPRUNG MIT $a^i_0 g b^i_{\nu(v)}$.
        Fall 2: $i = j,\ x = p^i_{lk} \notin a^i_0 E_i^+ b^i_0$ oder $i \neq j,\ x = q_{b^i_l, a^j_k}$.
        Fall 2a: $|X_{\tau(x)}| = 1$.
    $v := CS[\tau(x), j](g^j_{v_1} \cdots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    RÜCKSPRUNG MIT $v$.
        Fall 2b: $|X_{\tau(x)}| \neq 1$.
    $v := BS[\tau(x), \nu(x), j](g^j_{v_1} \cdots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    $v := EF[\tau(x), g^{\tau(x)}_{x_{\mu(x)}}](v)$
     ⋮
    $v:=EF[\tau(x), g_{x_1}^{\tau(x)}](v)$
    $v:=AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.

$BS[i, l, j](v)$:     Für $i, j=0(1)n$, $|X_i| \neq 1 \neq |X_j|$, $l=0(1)m_i$.
                      Voraussetzung: $v=g^j_{v_1} \cdots g^j_{v_{\mu(v)}} b^j_{\nu(v)}$ mit $\mu(v) \geqq 1$ ist
                      Postfix eines normierten Wortes.
                      Wirkung: $\overline{b^i_l v}$ berechnen.
        Fall 1: $i=j$, $l=0$.
    RÜCKSPRUNG MIT $a^i_0 v$.
        Fall 2: $i=j$, $l \neq 0$, $x=p^i_{l0} \in a^i_0 E_i^+ b^i_0$.
    $g:=GF[i, g^i_{x_{\mu(x)}}](g^i_{v_1} \cdots g^i_{v_{\mu(v)}})$
    $g:=GF[i, g^i_{x_{\mu(x)-1}}](g)$
    ⋮
    $g:=GF[i, g^i_{x_1}](g)$
    RÜCKSPRUNG MIT $a^i_0 g b^i_{\nu(v)}$.
        Fall 3: $i=j$, $l \neq 0$, $x=p^i_{l0} \notin a^i_0 E_i^+ b^i_0$.
        Fall 3a: $|X_{\tau(x)}|=1$.
    $v:=CS[\tau(x), j](v)$
    RÜCKSPRUNG MIT $v$.
        Fall 3b: $|X_{\tau(x)}| \neq 1$.
    $v:=BS[\tau(x), \nu(x), j](v)$
    $v:=EF[\tau(x), g^{\tau(x)}_{x_{\mu(x)}}](v)$
    ⋮
    $v:=EF[\tau(x), g^{\tau(x)}_{x_1}](v)$
    $v:=AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.
        Fall 4: $i \neq j$
    VERTEILERSPRUNG AUF $BSS[i, l, j, g^j_{v_1}]$.

$BSS[i, l, j, g]$:    Für $i, j=0(1)n$, $|X_i| \neq 1 \neq |X_j|$, $i \neq j$, $l=0(1)m_i$, $g \in E_j$.
                      Sei $x=q_{b^i_l, g}$.
        Fall 1: $|X_{\tau(x)}|=1$.
    IST $\mu(v)=1$?
    WENN JA: RÜCKSPRUNG MIT $q_{c^{\tau(x)}, b^j_{\nu(v)}}$
    SONST: $v:=CS[\tau(x), j](g^j_{v_2} \cdots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    RÜCKSPRUNG MIT $v$.
        Fall 2: $|X_{\tau(x)}| \neq 1$.
    IST $\mu(v)=1$?
    WENN JA: $v:=BB[\tau(x), \nu(x), j](\nu(v))$; WEITER BEI *
    SONST: $v:=BS[\tau(x), \nu(x), j](g^j_{v_2} \cdots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
*   $v:=EF[\tau(x), g^{\tau(x)}_{x_{\mu(x)}}](v)$
    ⋮
    $v:=EF[\tau(x), g^{\tau(x)}_{x_1}](v)$
    $v:=AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.
```

```
$BB[i, l, j](k)$:            Für $i, j=0(1)n$, $|X_i| \neq 1 \neq |X_j|$, $l=0(1)m_i$.
                             Wirkung: $\overline{b_l^i b_k^j}$ berechnen.
        Fall 1: $i=j$, $x=p_{l0}^i$, $|X_{\tau(x)}|=1$.
    RÜCKSPRUNG MIT $q_{c^{\tau(x)}, b_k^j}$.
        Fall 2: $i=j$, $x=p_{l0}^i$, $|X_{\tau(x)}| \neq 1$.
    $v := BB[\tau(x), \nu(x), j](k)$
    $v := EF[\tau(x), g_{x_{\mu(x)}}^{\tau(x)}](v)$
    ⋮
    $v := EF[\tau(x), g_{x_1}^{\tau(x)}](v)$
    $v := AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.
        Fall 3: $i \neq j$.
    RÜCKSPRUNG MIT $q_{b_l^i, b_k^j}$.

$E[i](g, v)$:                Für $i=0(1)n$, $|X_i| \neq 1$.
                             Voraussetzung: $g \in E_i$, $v$ normiert.
                             Wirkung: $\overline{gv}$ berechnen.
    VERTEILERSPRUNG AUF $EFF[i, g]$.

$EF[i, g](v)$:               Für $i=0(1)n$, $|X_i| \neq 1$, $g \in E_i$.
                             Voraussetzung und Wirkung wie $E[i](g, v)$.
$EFF[i, g]$:
    IST $|X_{\tau(v)}|=1$?
    WENN JA: RÜCKSPRUNG MIT $q_{g,v}$
    SONST: VERTEILERSPRUNG AUF $EF1[i, g, \tau(v), \lambda(v)]$.
$EF1[i, g, j, l]$:           Für $i, j=0(1)n$, $|X_i| \neq 1 \neq |X_j|$, $g \in E_i$, $l=0(1)k_j$.
        Fall 1: $i=j$, $l=0$.
    $h := GF[i, g](g_{v_1}^i \dots g_{v_{\mu(v)}}^i)$
    RÜCKSPRUNG MIT $a_0^i h b_{\nu(v)}^i$.
        Fall 2: $i=j$, $l \neq 0$, $x=p_{0l}^i \in a_0^i E_i^+ b_0^i$.
    $h := GF[i, g_{x_{\mu(x)}}^i](g_{v_1}^i \dots g_{v_{\mu(v)}}^i)$
    $h := GF[i, g_{x_{\mu(x)-1}}^i](h)$
    ⋮
    $h := GF[i, g_{x_1}^i](h)$
    $h := GF[i, g](h)$
    RÜCKSPRUNG MIT $a_0^i h b_{\nu(v)}^i$.
        Fall 3: $i=j$, $l \neq 0$, $x=p_{0l}^i \notin a_0^i E_i^+ b_0^i$, $|X_{\tau(x)}|=1$.
    $v := CS[\tau(x), j](g_{v_1}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^j)$
    $v := EF[i, g](v)$
    RÜCKSPRUNG MIT $v$.
        Fall 4: $i=j$, $l \neq 0$, $x=p_{0l}^i \in a_0^i E_i^+ b_0^i$, $|X_{\tau(x)}| \neq 1$.
    $v := BS[\tau(x), \nu(x), j](g_{v_1}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^i)$
    $v := EF[\tau(x), g_{x_{\mu(x)}}^{\tau(x)}](v)$
    ⋮
    $v := EF[\tau(x), g_{x_1}^{\tau(x)}](v)$
    $v := AF[\tau(x), \lambda(x)](v)$
    $v := EF[i, g](v)$
    RÜCKSPRUNG MIT $v$.
        *Fall 5:* $i \neq j,\ x = q_{g,a_l^j},\ |X_{\tau(x)}| = 1$.
    $v := CS[\tau(x), j](g^j_{v_1} \dots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    RÜCKSPRUNG MIT $v$.
        *Fall 6:* $i \neq j,\ x = q_{g,a_l^j},\ |X_{\tau(x)}| \neq 1$.
    $v := BS[\tau(x), \nu(x), j](g^j_{v_1} \dots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    $v := EF[\tau(x), g^{\nu(x)}_{x_{\mu(x)}}](v)$
    ⋮
    $v := EF[\tau(x), g^{\tau(x)}_{x_1}](v)$
    $v := AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.

$A[i](k, v)$:                Für $i = 0(1)n,\ |X_i| \neq 1$.
                             Voraussetzung: $v$ normiert.
                             Wirkung: $\overline{a^i_k v}$ berechnen.
    VERTEILERSPRUNG AUF $AFF[i, k]$.

$AF[i, k](v)$:               Für $i = 0(1)n,\ |X_i| \neq 1,\ k = 0(1)k_i$.
                             Voraussetzung und Wirkung wie $A[i](k, v)$.
$AFF[i, k]$:
    IST $|X_{\tau(v)}| = 1$?
    WENN JA: RÜCKSPRUNG MIT $q_{a^i_k, v}$.
    SONST: VERTEILERSPRUNG AUF $AF1[i, k, \tau(v), \lambda(v)]$.
$AF1[i, k, j, l]$:           Für $i, j = 0(1)n,\ |X_i| \neq 1 \neq |X_j|,\ k = 0(1)k_i,\ l = 0(1)m_i$.
        *Fall 1:* $i = j,\ l = 0$.
    RÜCKSPRUNG MIT $a^i_k g^i_{v_1} \dots g^i_{v_{\mu(v)}} b^i_{\nu(v)}$.
        *Fall 2:* $i = j,\ l \neq 0,\ x = p^i_{0l} \in a^i_0 E_i^+ b^i_0$.
    $g := GF[i, g^i_{x_{\mu(x)}}](g^i_{v_1} \dots g^i_{v_{\mu(v)}})$
    $g := GF[i, g^i_{x_{\mu(x)-1}}](g)$
    ⋮
    $g := GF[i, g^i_{x_1}](g)$
    RÜCKSPRUNG MIT $a^i_k g b^i_{\nu(v)}$.
        *Fall 3:* $i = j,\ l \neq 0,\ x = p^i_{0l} \notin a^i_0 E_i^+ b^i_0,\ |X_{\tau(x)}| = 1$.
    $v := CS[\tau(x), j](g^j_{v_1} \dots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    $v := AF[i, k](v)$
    RÜCKSPRUNG MIT $v$.
        *Fall 4:* $i = j,\ l \neq 0,\ x = p^i_{0l} \notin a^i_0 E_i^+ b^i_0,\ |X_{\tau(x)}| \neq 1$.
    $v := BS[\tau(x, \nu(x), j]][g^j_{v_1} \dots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    $v := EF[\tau(x), g^{\tau(x)}_{x_{\mu(x)}}](v)$
    ⋮
    $v := EF[\tau(x), g^{\tau(x)}_{x_1}](v)$
    $v := AF[\tau(x), \lambda(x)](v)$
    $v := AF[i, k](v)$
    RÜCKSPRUNG MIT $v$.
        *Fall 5:* $i \neq j,\ x = q_{a^i_k, a^j_l},\ |X_{\tau(x)}| = 1$.
    $v := CS[\tau(x), j](g^j_{v_1} \dots g^j_{v_{\mu(v)}} b^j_{\nu(v)})$
    RÜCKSPRUNG MIT $v$.
        Fall 6: $i \neq j,\ x = q_{a_k^i, a_l^j},\ |X_{\tau(x)}| \neq 1$.
    $v := BS[\tau(x), \nu(x), j](g_{v_1}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^j)$
    $v := EF[\tau(x), g_{x_{\mu(x)}}^{\tau(x)}](v)$
      ⋮
    $v := EF[\tau(x), g_{x_1}^{\tau(x)}](v)$
    $v := AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.

$C[i, j](v)$:                Für $i, j = 0(1)n,\ |X_i| = 1 \neq |X_j|$.
                             Voraussetzung: $v \in X_j^+$ normiert.
                             Wirkung: $\overline{c^i v}$ berechnen.
    VERTEILERSPRUNG AUF $CA[i, j, \lambda(v)]$.
$CA[i, j, k]$:               Für $i, j = 0(1)n,\ |X_i| = 1 \neq |X_j|,\ k = 0(1)k_j$.
                             Sei $x = q_{c^i, a_k^j}$.
        Fall 1: $|X_{\tau(x)}| = 1$.
    $v := CS[\tau(x), j](g_{v_1}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^j)$
    RÜCKSPRUNG MIT $v$.
        Fall 2: $|X_{\tau(x)}| \neq 1$.
    $v := BS[\tau(x), \nu(x), j](g_{v_1}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^j)$
    $v := EF[\tau(x), g_{x_{\mu(x)}}^{\tau(x)}](v)$
      ⋮
    $v := EF[\tau(x), g_{x_1}^{\tau(x)}](v)$
    $v := AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.

$CS[i, j](v)$:               Für $i, j = 0(1)n,\ |X_i| = 1 \neq |X_j|$.
                             Voraussetzung: $v = g_{v_1}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^j \in X_j^+$ mit $\mu(v) \geqq 1$ ist
                             Postfix eines normierten Wortes.
                             Wirkung: $\overline{c^i v}$ berechnen.
    VERTEILERSPRUNG AUF $CSS[i, j, g_{v_1}^j]$.
$CSS[i, j, g]$:              Für $i, j = 0(1)n,\ |X_i| = 1 \neq |X_j|,\ g \in E_j$.
                             Sei $x = q_{c^i, g}$.
        Fall 1: $|X_{\tau(x)}| = 1$.
    IST $\mu(v) = 1$?
    WENN JA: RÜCKSPRUNG MIT $q_{c^{\tau(x)}, b_{\nu(v)}^j}$.
    SONST: $v := CS[\tau(x), j](g_{v_2}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^j)$
    RÜCKSPRUNG MIT $v$.
        Fall 2: $|X_{\tau(x)}| \neq 1$.
    IST $\mu(v) = 1$?
    WENN JA: $v := BB[\tau(x), \nu(x), j](\nu(v))$; WEITER BEI *
    SONST: $v := BS[\tau(x), \nu(x), j](g_{v_2}^j \dots g_{v_{\mu(v)}}^j b_{\nu(v)}^j)$
*   $v := EF[\tau(x), g_{x_{\mu(x)}}^{\tau(x)}](v)$
      ⋮
    $v := EF[\tau(x), g_{x_1}^{\tau(x)}](v)$
    $v := AF[\tau(x), \lambda(x)](v)$
    RÜCKSPRUNG MIT $v$.
```

```
GF[i, g](h):        Für $i=0(1)n$, $|X_i|\neq 1$, $g\in E_i$.
                    Voraussetzung: $h\in E_i^+$ in $G_i$ normiert.
                    Wirkung: $gh$ in $G_i$ normiert berechnen.
    ORAKEL (Zur Realisierung vgl. Abschnitt 3).
```

3. Programmierung und Einsatz von MULT

Für den beschriebenen Algorithmus MULT gilt in verstärktem Maße das für das Rechnen mit den Elementen abstrakt präsentierter Gruppen in [2, 3, 6, 7, 8] Gesagte: Durch die wiederholten Abfragen der für das Rechnen innerhalb einer Halbgruppe konstanten Relationen wird der Algorithmus extrem langsam. So bietet sich auch im vorliegenden Fall an, MULT zweistufig zu realisieren, indem alle von den jeweiligen zu multiplizierenden Elementen unabhängigen Entscheidungen in einer Vorbereitungsphase V getroffen werden. Entsprechende Programme V wurden für speziell geformte Präsentationen auflösbarer Gruppen in [2, 3, 6, 7, 8] dokumentiert. Unsere — zur Vereinfachung allerdings in LISP durchgeführte — Implementation von V für MULT basiert auf der wegen ihrer Portabilität besonders günstigen Version V_F aus [2]. Hier wie dort besteht das durch V generierte eigentliche Multiplikationsprogramm MULT$[X, R]$ aus einer Folge von Unterprogrammen, die jedes selbst wieder im wesentlichen nur aus vereinzelten Befehlen zum Holen von Konstanten usw. und aus zahlreichen Unterprogrammaufrufen bestehen — ihre jeweilige Gestalt ist für die entsprechenden Parameterwerte und Fälle durch den Algorithmus des vorigen Abschnitts vorgegeben. Die Beschleunigung von MULT$[X, R]$ gegen MULT ist im allgemeinen sehr groß, und die Erfahrungen lassen bei aller Problematik eines Vergleichs erkennen, daß MULT$[X, R]$ von der Geschwindigkeit her mit Multiplikationsprogrammen für andere Darstellungen von $S_{(X,R)}$ gut konkurrieren kann.

Der typische Einsatz von MULT ist analog den Gruppenprogrammen aus [2, 3, 6, 7, 8] folgendermaßen: Zur Lösung der Aufgabe, Eigenschaften der durch die normierte Präsentation (X, R) gegebenen Halbgruppe zu berechnen, wird zunächst mit dem Programm aus [5] nachgeprüft, ob $\delta(X, R)=|S_{(X,R)}|$ gilt. Falls nein, versucht man, die Präsentation geeignet zu modifizieren; häufig kann man dabei Zwischenergebnisse dieses Programmes ausnutzen. Falls die Bedingung erfüllt ist, wird mit V das Programm MULT$[X, R]$ generiert, welches dann vom eigentlichen Rechenprogramm (wie z. B. [4]) für die einzelnen Multiplikationen aufgerufen wird.

Die bisherigen Überlegungen erfolgten sämtlich unter der Voraussetzung geeigneter Orakel für das Rechnen mit den Elementen der durch die Halbgruppenpräsentationen (E_i, S_i) gegebenen Gruppen G_i. Die Realisierungsmöglichkeit und Realisierungsweise dieser Orakel $GF[i, g](h)$ hängt stark von der Form der Präsentationen (E_i, S_i) ab. So kann man als einen Extremfall etwa $E_i=G_i$ und S_i als die volle Cayleytafel von G_i wählen; dadurch erhält man mit den normierten Präsentationen sämtliche endlichen Halbgruppen, und die Orakel werden triviale Programme; der erforderliche Speicheraufwand wird jedoch schon für mäßig große Halbgruppen unerträglich groß. Günstiger wird es z. B., wenn man voraussetzt, daß die Präsentationen (E_i, S_i) als Halbgruppenpräsentationen von Gruppen durch die üblichen Umformungen aus „AG-Systemen" [3] (pc-presentation [2]) (E_i', S_i') hervorgehen. Als Orakel $GF[i, g](h)$ kann man dann die entsprechenden Teile des mit V_F aus (E_i', S_i') gewonnen eigentlichen Multiplikationsprogrammes für G_i verwenden. Die

Beschränkung auf die Klasse $\mathfrak{N}_{pc}$ der normierten Präsentationen, in denen die (E_i, S_i) kanonisch aus pc-Präsentationen gewonnen werden, hat noch weitere Vorteile: Ersetzt man 1.1 (3a) durch die pc-Forderungen aus [2] und die Umformungsregeln, so erhält man aus 1.1 für $\mathfrak{N}_{pc}$ eine formale, entscheidbare Charakterisierung. Unter der Voraussetzung $(X, R) \in \mathfrak{N}_{pc}$ kann man daher auf die Angabe der Partition von X verzichten, weil diese für $\delta(X, R) = |S_{(X,R)}|$ bis auf die Indizierung eindeutig mit 1.8 bestimmt werden kann. Algebraisch bedeutet die Beschränkung auf $\mathfrak{N}_{pc}$, daß nur und genau die endlichen Halbgruppen mit ausschließlich auflösbaren Untergruppen betrachtet werden, eine praktisch auch noch bei mäßig großen Halbgruppen fast unbedeutende Einschränkung.

INSTITUT FÜR THEORETISCHE INFORMATIK
TECHNISCHE HOCHSCHULE DARMSTADT
MAGDALENEN STR. 11
D—6100 DARMSTADT

Literatur

[1] CLIFFORD, A. H., und G. B. PRESTON, The algebraic theory of semigroups, I. (2. Aufl.) Providence, Rh. I., 1964.
[2] FELSCH, V., A machine independent implementation of a collection algorithm for the multiplication of group elements, *Proceedings SYMSAC 76*, New York, 1976.
[3] JÜRGENSEN, H., Calculation with the elements of a finite group given by generators and defining relations, In: *J. Leech* (Hrsg.), Computational problems in abstract algebra, *Proceedings of a conference* (Oxford, 1967) Oxford, 1970.
[4] JÜRGENSEN, H. und P. WICK, Bestimmung der Unterhalbgruppenverbände für zwei Klassen endlicher Halbgruppen, *Computing*, v. 11, 1973, pp. 337—351.
[5] JÜRGENSEN, H., Transformationendarstellungen endlicher abstrakt präsentierter Halbgruppen, *Bericht*, No. 7605, Inst. f. Informatik u. Prakt. Math., Universität Kiel, 1976.
[6] LINDENBERG, W,. Über die Darstellung von Gruppenelementen in digitalen Rechenautomaten, *Numer. Math.*, v. 4, 1962, pp. 151—153.
[7] LINDENBERG, W., Die Struktur eines Übersetzungsprogrammes zur Multiplikation von Gruppenelementen in digitalen Rechenautomaten, *Mitt. Rh.—Westf. Inst. für Instrum. Math.*, Bonn, v. 2, 1963, pp. 1—38.
[8] NEUBÜSER, J., Bestimmung der Untergruppenverbände endlicher p-Gruppen auf einer programmgesteuerten elektronischen Dualmaschine, *Numer. Math.*, v. 3, 1961, pp. 271—278.
[9] PETRICH, M., *Introduction to semigroups*, Columbus, Ohio, 1973.

(Eingegangen am 8. März 1977)

Zur Synthese von DOL-Systemen

Von W. Kämmerer

Herrn Professor László Kalmár zum Gedächtnis

Ein determiniertes von Wechselwirkungen freies Lindenmayer-System (DOL-System) [1, 2, 3] wird durch ein Tripel

$$S=\langle V, P, w_0\rangle$$

definiert. Darin ist

$V=\{a_1, a_2, \dots, a_n\}$ eine Menge von Symbolen, das Alphabet,
P eine quadratische Matrix von Rang n, die Produktionsmatrix und
$w_0=a_1^{\alpha_1^0} a_2^{\alpha_2^0} \dots a_n^{\alpha_n^0}$ ein über V gebildetes Wort, das Startwort, wobei α_i^t angibt wievielmal das Symbol a_i in dem Wort w_t vorkommt.

Die Worte der von diesem System bestimmten Sprache werden ausgehend von dem Startwort schrittweise durch einen Ableitungsprozeß gewonnen, der bei jedem Schritt auf alle Symbole des jeweiligen Worts parallel durchgeführt wird.

Von besonderem Interesse im Rahmen biologischer Fragen, wobei die einzelnen Symbole als spezifische Zellen interpretiert werden, ist die Länge des nach t Ableitungsschritten erreichten Worts w_t, die durch die Wachstumsfunktion

$$f_t = \sum_{i=1}^{n} \alpha_i^t$$

definiert ist.

Für das Syntheseproblem, zu einer vorgegebenen Funktion ein DOL-System zu finden, das diese Funktion als Wachstumsfunktion besitzt, ist vorauszusetzen, daß die vorgelegte Funktion von polynomialem, oder exponentialem oder gemischtem Typa ist. In diesem Fall ist der Übergang von der Wachstumsfunktion zu einer Differenzengleichung, die diese Funktion als Lösung besitzt, stets möglich. Damit ist dann auch die charakteristische Gleichung gewonnen. Die Schwierigkeiten liegen nun in dem erforderlichen Übergang zu einer Matrix, die diese Gleichung als ihre charakteristische Gleichung besitzt und dabei Produktionsmatrix eines DOL-Systems ist.

Nur in einfachsten Fällen gelingt eine direkte Lösung durch Auflösung der resultierenden Gleichungen nach ganzzahligen nichtnegativen Elementen der Matrix.

Für rein polynomiale Wachstumsfunktionen ist ein Weg zur Durchführung der Synthese 1971 von A. L. Szilard [4] über die Erzeugungsfunktion angegeben worden.

Im folgenden soll ein Verfahren aufgezeigt werden, das für eine bestimmte Klasse von Wachstumsfunktionen eine äußerst einfache Konstruktion eines geeigneten DOL-Systems ermöglicht.

Das Wort w_{n-1} mit der Länge f_{n-1} trägt in dem Auftreten der n Symbole $a_1, a_2, \ldots, a_n$ Informationen über die n Werte der Wachstumsfunktion $f_0, f_1, \ldots, f_{n-1}$ in Form von Linearkombinationen dieser Werte.

Es liegt nahe, die Klasse von Wachstumsfunktionen zu betrachten, für die sich diese Informationen direkt aus dem Differenzenschema ergeben.

$$\begin{array}{ll} f_0 & \\ & \Delta_0 \\ f_1 & \\ & \Delta_1 \\ f_2 & \\ \vdots & \\ & \Delta_{n-2} \\ f_{n-1} & \\ & \Delta_{n-1} \\ f_n & \end{array}$$

Mit dem Ansatz

$$w_{n-1} = a_1^{f_0}\, a_2^{\Delta_0} a_3^{\Delta_1} \ldots a_n^{\Delta_{n-2}}$$

und folglich

$$w_n = a_1^{f_0+\Delta_0} a_2^{\Delta_1} a_3^{\Delta_2} \ldots a_n^{\Delta_{n-1}}$$

ergibt sich über die vorgelegte Differenzengleichung

$$f_n + q_{n-1} f_{n-1} + \ldots + q_1 f_1 + q_0 f_0 = 0$$

mit

$$\begin{aligned} f_1 &= f_0 + \Delta_0 \\ f_2 &= f_0 + \Delta_0 + \Delta_1 \\ &\ldots\ldots\ldots\ldots\ldots\ldots \\ f_{n-1} &= f_0 + \Delta_0 + \Delta_1 + \ldots + \Delta_{n-2} \\ f_n &= f_0 + \Delta_0 + \Delta_1 + \ldots + \Delta_{n-2} + \Delta_{n-1} \end{aligned}$$

eine Gleichung für Δ_{n-1}

$$\Delta_{n-1} = Q_1 f_0 + Q_2 \Delta_0 + \ldots + Q_n \Delta_{n-2},$$

wobei als Abkürzungen

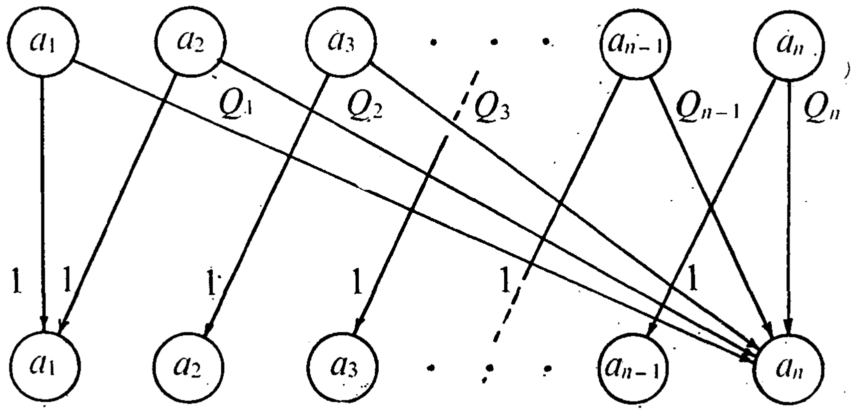
$$\begin{aligned} Q_1 &= -(1 + q_{n-1} + q_{n-2} + \ldots + q_1 + q_0) \\ Q_2 &= -(1 + q_{n-1} + q_{n-2} + \ldots + q_1) \\ &\ldots \\ Q_n &= -(1 + q_{n-1}) \end{aligned}$$

stehen.

Damit erhält man für das Auftreten der einzelnen Symbole in den aufeinanderfolgenden Wörtern w_{n-1} und w_n das folgende Schema:

	a_1	a_2	a_3	$\dots$	a_{n-1}	a_n
w_{n-1}	f_0	Δ_0	Δ_1	$\dots$	Δ_{n-3}	Δ_{n-2}
w_n	$f_0+\Delta_0$	Δ_1	Δ_2	$\dots$	Δ_{n-2}	$Q_1 f_0+Q_2\Delta_0+\dots+Q_n\Delta_{n-2}$

Daraus lassen sich die folgenden Produktionsregeln gewinnen:



$$\begin{array}{lll}
a_1 \to a_1 & & a_n^{Q_1} \\
a_2 \to a_1 & & a_n^{Q_2} \\
a_3 \to & a_2 & a_n^{Q_3} \\
\dots & & \\
a_n \to & a_{n-1} & a_n^{Q_n}
\end{array}$$

Damit diese realisierbar sind, müssen neben den Einlaufwerten f_0, $\Delta_0, \Delta_1, \dots, \Delta_{n-2}$ auch die Größen $Q_1, Q_2, \dots, Q_n$ nichtnegative ganze Zahlen sein.

Ein Beispiel erläutere das äußerst einfache Konstruktionsverfahren.

Die Wachstumsfunktion sei durch die Differenzengleichung

$$f_4-6f_3+5f_2-f_1+f_0=0$$

gegeben. Aus den Koeffizienten erhält man über

$$\begin{array}{rrrr}
1 & 1 & 1 & 1 \\
-6 & -6 & -6 & -6 \\
5 & 5 & 5 & \\
-1 & -1 & & \\
1 & & & \\
\hline
0 & -1 & 0 & -5
\end{array}$$

$$Q_1=0 \quad Q_2=1 \quad Q_3=0 \quad Q_4=5$$

und damit die Produktionsregeln eines möglichen DOL-Systems.

$$\begin{aligned} a_1 &\to a_1 \\ a_2 &\to a_1 \quad a_4 \\ a_3 &\to a_2 \\ a_4 &\to a_3 \quad a_4^5 \end{aligned}$$

Zur Überprüfung betrachte man die Produktionsmatrix.

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 5 \end{pmatrix}.$$

Diese liefert als charakteristische Gleichung

$$\begin{vmatrix} \lambda-1 & 0 & 0 & 0 \\ -1 & \lambda & 0 & -1 \\ 0 & -1 & \lambda & 0 \\ 0 & 0 & -1 & \lambda-5 \end{vmatrix} = 0$$

also

$$\lambda^4 - 6\lambda^3 + 5\lambda^2 - \lambda + 1 = 0.$$

Damit ergibt sich die geforderte Differenzengleichung.

Sind als Einlaufwerte z. B. $f_0 = 1, f_1 = 2, f_2 = 4, f_3 = 8$ gefordert, so ergibt sich über die Differenzengleichung

$$\begin{array}{ll} \textcircled{1} & \\ 2 & \textcircled{1} \\ 4 & \textcircled{2} \\ 8 & \textcircled{4} \end{array}$$

für w_3

$$w_3 = a_1 a_2 a_3^2 a_4^4$$

mit $f_3 = 8$. Von den daraus ableitbaren Wörtern seien hier nur

$$w_4 = a_1^2 a_2^2 a_3^4 a_4^{21}$$

$$w_5 = a_1^4 a_2^4 a_3^{21} a_4^{107}$$

mit $f_5 = 136$ angegeben. Diese Werte stehen ebenfalls mit der Differenzengleichung in Übereinstimmung.

Soll die geforderte Folge schon mit ihren Einlaufwerten ableitbar sein, so sind die $(n-1)$ vorhergehenden Funktionswerte aus der Differenzengleichung zu bestimmen. Ergeben sich dabei für das Differenzenschema nicht realisierbare Werte, so kann man das Produktionssystem geeignet erweitern.

Im vorliegenden Beispiel durch

$$\begin{aligned} x &\to y\,u \\ y &\to z\,u^3 \\ z &\to a_1 a_2 a_3^2 a_4^4 \\ u &\to \emptyset. \end{aligned}$$

Damit ergibt sich

$$\begin{aligned} w_0 &= x &\quad \text{mit} \quad f_0 &= 1 \\ w_1 &= yu & f_1 &= 2 \\ w_2 &= zu^3 & f_2 &= 4 \end{aligned}$$

und

$$w_3 = a_1 a_2 a_3^2 a_4^2 \qquad f_3 = 8.$$

Der angefügte Teil des Produktionssystems ließe sich etwa als „embryonales“ Entwicklungssystem interpretieren.

Literaturverzeichnis

[1] HERMAN, G. T., G. ROZENBERG and A. LINDENMAYER, *Developmental systems and languages*, Amsterdam—Oxford, North-Holland Publishing Company, 1975.

[2] LINDENMAYER, A., Mathematical models for cellular interactions in development, Parts I and II, *J. Theoret. Biol.*, v. 18, 1968, pp. 280—315.

[3] ROZENBERG, G. and A. SALOMAA (eds), *L-systems*, Springer Lecture Notes in Computer Science, No. 15, Berlin—Heidelberg—New York, Springer-Verlag. 1974.

[4] SZILARD, A. L., *Growth functions of LINDENMAYER systems*, Universita of Western Ontario, Computer Science Department, Technical Report No. 4, 1971.

[5] KÄMMERER, W., Eine Einführung in *L*-Systeme und *L*-Sprachen, *Nova acta Leopoldina*, Vorträge anläßlich des Symposiums „Naturwissenschaftliche Linguistik“ vom 25. bis 29. Juli 1976, zu Halle (Saale), z. Z. im Druck.

(*Eingegangen am 9. März. 1978*)

Differentiability properties of computable functions — a summary

By M. B. Pour-El and I. Richards

To the memory of Professor László Kalmár

Contemporary computing machinery includes many analog devices — machines which, by a direct process, produce continuous functions of a real variable as their output. These functions appear to be computable by virtue of the fact that there are real existing devices which generate them. One might attempt to understand these functions by means of traditional computer — oriented techniques. For example, one might begin with an effectively generable set of functions each of which is "obviously computable". One might then consider the class of functions obtained from this set by finite programs (=interpreted schemes=flow charts=finite algorithms), [3]. However, as Shepherdson has pointed out [8], no procedure of this kind can encompass all the computable functions of a real variable. Fortunately the literature of recursion theory provides a precise definition of this concept — Grzegorczyk [1]. We give this definition below.

Our paper is concerned with the differentiability properties of computable functions of a real variable. Unless stated otherwise, we restrict our attention to functions defined on compact intervals. Grzegorczyk raised the question [1, p. 201] whether differentiation and integration are computable processes. The indefinite integral of a computable function is computable ([4], [7]). Lacombe [5] stated a negative result for differentiation, although he gave no proof. He made no mention of higher derivatives. In 1971, Myhill [6] showed that the derivative of a computable continuously differentiable function need not be computable. He suggested in a footnote that the same should hold for infinitely differentiable functions. This seems at first glance to follow from a modification of his construction — a modification so obvious that it need not be written down. However, the result turns out to be false. We prove that if $f(x)$ is infinitely differentiable and computable, then all of its derivatives are computable. This follows from the stronger statement:

Proposition. If $f(x)$ is computable and of class C^2 (twice continuously differentiable) on a compact interval $[-M, M]$ (with M a positive integer), then $f'(x)$ is computable.

This proposition is best possible. For by modifying Myhill's counterexample [6] slightly, we can construct a computable function which is twice differentiable (but not continuously), and whose derivative is not computable. Using a completely different construction we can show:

Example. There is a computable continuously differentiable function $f(x)$ on $[0, 1]$ whose derivative $f'(x)$ is not computable, but such that $f'(x)$ is "Banach—Mazur computable" (definition below).

As an immediate consequence of the proposition we have:

Corollary. If $f(x)$ is computable and C^∞ (infinitely differentiable) on a compact interval $[-M, M]$ (M a positive integer), then the n-th derivative $f^{(n)}(x)$ is computable for each n.

We now give Grzegorczyk's definition of a computable function of a real variable. The fundamental definition is phrased in terms of general recursive functionals. In [2], Grzegorczyk presented seven definitions all of which were proved equivalent to the fundamental definition. In this paper we find it convenient to use one of these other definitions. First we need:

Definition 1. A sequence of reals $\{x_n\}$ is *computable* if there exist recursive functions $a(n, k)$, $b(n, k)$, and $s(n, k)$ such that

$$\left| x_n - (-1)^{s(n,k)} \frac{a(n, k)}{b(n, k)} \right| < \frac{1}{k+1}$$

for all n, k (with $b(n, k) \neq 0$).

Roughly, this means that there is a recursive double sequence of rationals r_{nk} which converge effectively to x_n as $k \to \infty$.

Definition 2. A function $f(x)$ from a compact interval of $\mathbf{R}$ into $\mathbf{R}$ is computable if:

(i) f maps every computable sequence of reals into a computable sequence of reals (the Banach—Mazur property);

(ii) f is "effectively uniformly continuous", i.e. there is a recursive function $g(n) > 0$ such that

$$|x-y| < \frac{1}{g(n)} \quad \text{implies} \quad |f(x)-f(y)| < \frac{1}{n+1}.$$

(This is Grzegorczyk's definition reduced to the case of a compact interval. Grzegorczyk considered functions from $\mathbf{R}$ to $\mathbf{R}$, and used a more complicated version of condition ii) to take account of the noncompactness of the domain.)

Two further results follow from our work. 1) The example above shows that there exists a computable continuously differentiable function $f(x)$ whose derivative satisfies condition i) of definition 2 (the Banach—Mazur condition), but not condition ii). By contrast, there is no case where $f(x)$ is computable and $f'(x)$ satisfies ii) but not i).

2) An attempt to extend the corollary leads to the following counterexample. There is a computable infinitely differentiable function $f(x)$ on $[0, 1]$ such that the sequence of n-th derivatives is not uniformly computable as a function of n. In other words, although by the corollary each derivative is computable, the sequence of derivatives need not be.

We now consider the proofs. The proposition is fairly easy, and so we shall give a sketch. However, the counterexamples are rather intricate. For the sake of brevity we omit an account of the constructions involved.

To prove the proposition we proceed as follows: Since $f''(x)$ is continuous on a compact set, it is bounded. Thus $|f''(x)| \leqq K$, an integer. Now by the mean value theorem, for any $x, y \in [-M, M]$ with $x<y$, there exists a ξ with $x<\xi<y$ such that:

$$f'(y)-f'(x) = f''(\xi)(y-x).$$

Hence $f'(x)$ is effectively uniformly continuous — in fact, $|f'(y)-f'(x)| \leqq K|y-x|$. Now applying the mean value theorem again (this time to f and f') we have:

For all $x, y \in [-M, M]$ with $x<y$, there exists a ξ with $x<\xi<y$ such that:

$$f'(\xi) = \frac{f(y)-f(x)}{y-x}.$$

The difference quotient $[f(y)-f(x)]/(y-x)$ is computable since f is. And the effective uniform continuity of f' means that $f'(\xi)$ converges effectively to $f'(x)$ as $\xi \to x$. □

The result proved above does not hold for functions defined on noncompact intervals such as the real line. (Here we return to Grzegorczyk's original definition [1], with its more complicated condition (ii).) By modifying Myhill's counterexample in an obvious way, we can show that: There is a computable infinitely differentiable function on the real line whose first derivative is not computable.

A detailed account of the results discussed in this note is planned for a forthcoming paper.

SCHOOL OF MATHEMATICS
UNIVERSITY OF MINNESOTA
MINNEAPOLIS, MINNESOTA 55455

Bibliography

[1] GRZEGORCZYK, A., Computable functionals, *Fund. Math.*, v. 42, 1955, pp. 168—202.
[2] GRZEGORCZYK, A., On the definitions of computable real continuous functions, *Fund. Math.*, v. 44, 1957, pp. 61—71.
[3] HERMAN, G. and S. ISARD, Computability over arbitrary fields, *J. London Math. Soc.*, v. 2, 1970, pp. 73—79.
[4] LACOMBE, D., Extension de la notion de fonction récursive aux fonctions d'une ou plusieurs variables reelles III., *C. R. Acad. Sci. Paris*, v. 241, 1955, pp. 151—153.
[5] LACOMBE, D., Quelques proprietés d'analyse récursive, *C. R. Acad. Sci. Paris*, v. 244, 1957, pp. 996—997.
[6] MYHILL, J., A recursive function defined on a compact interval and having a continuous derivative that is not recursive, *Michigan Math. J.*, v. 18, 1971, pp. 97—98.
[7] POUR-EL, M. B. and J. CALDWELL, On a simple definition of computable function of a real variable — with applications to functions of a complex variable, *Z. Math. Logik Grundlagen Math.*, v. 21, 1975, pp. 1—19.
[8] SHEPHERDSON, J. C., On the definition of computable function of a real variable, *Z. Math. Logik Grundlagen Math.*, v. 22, 1976, pp. 391—402.

(Received March 22, 1978)

Equality sets for homomorphisms of free monoids

By A. SALOMAA

To the memory of László Kalmár[1]

1. Introduction

Some of the very basic questions concerning homomorphisms have recently turned out to be of crucial importance for some of the most interesting decision problems in language theory. Although homomorphisms of free monoids are very simple and, at least from the mathematical point of view, the most natural operations defined for languages, some of these very basic questions remain still unanswered.

The basic set-up in this paper is as follows. We are given two homomorphisms h_1 and h_2 mapping the free monoid Σ^* generated by an alphabet Σ into Σ_1^*, where Σ_1 is a possibly different alphabet. We study the language $E(h_1, h_2)$ consisting of all words w over Σ^* such that $h_1(w)=h_2(w)$. This language $E(h_1, h_2)$ is referred to as the *equality set* or *equality language* for the pair (h_1, h_2). This paper investigates properties of equality languages, especially with respect to certain decision problems.

A brief outline of the contents of the paper follows. After the basic definitions and some preliminary results presented in Section 2, we investigate in Section 3 the case where the equality language is regular. This is a very desirable state of affairs from the point of view of decision problems. We show, for instance, that the equality language is regular if and only if it can be expressed in terms of so-called bounded balance. This situation occurs always when we are dealing with the "elementary homomorphisms" of [5]. In Section 4, we show that every recursively enumerable language is obtained from an equality language by a deterministic *gsm* mapping. Equality languages are context-sensitive star languages (where "star language" is a "star event" in the sense of [1]). If the homomorphisms h_1 and h_2 are into the monoid generated by one letter, then $E(h_1, h_2)$ is context-free but not necessarily regular.

[1] The author had the pleasure of spending a longer time with Laci bácsi during the summer 1976. I was then in Szeged with my wife, son and Mr. and Mrs. Esko Teräväinen. We were all impressed by Laci bácsi's generous hospitality and friendliness. For instance, he was always ready to carry the two biggest suitcases. In August 1976, Laci bácsi was supposed to present a paper at the XVII Scandinavian Congress of Mathematicians in Turku. He was working also on this during our stay in Szeged.

The final Section 5 deals with some decidability results, and also points out some open problems.

We assume the reader to be familiar with basic formal language theory. For all unexplained notions we refer to [9].

2. Definitions and preliminary results

Consider the free monoid Σ^* generated by a finite alphabet Σ. The identity element of Σ^* (i.e., the empty word) is denoted by λ, and the length of a word $w \in \Sigma^*$ by $\lg(w)$. Consider, further, two homomorphisms h_1 and h_2 mapping Σ^* into Σ_1^*, where Σ_1 is another (possibly the same) alphabet. We denote by $E(h_1, h_2)$ the collection of all words $w \in \Sigma^*$ such that

$$h_1(w) = h_2(w).$$

The set $E(h_1, h_2)$ is referred to as the *equality set* or *equality language* of h_1 and h_2. (In [6], equality sets are denoted by MID (h_1, h_2).) The family of all languages L such that $L = E(h_1, h_2)$, for some homomorphisms h_1 and h_2, is denoted by F_E.

It is clear that $E(h_1, h_2)$ remains unchanged under a renaming of Σ_1. Moreover, it is immediately seen by standard coding techniques that any language L over Σ in F_E can be given as $L = E(h_1, h_2)$, where h_1 and h_2 map Σ^* into $\{a, b\}^*$, i.e., Σ_1 consists of two letters only. A further reduction to a one-letter alphabet is not possible, as will be explicitly shown in Sections 3 and 4.

We now repeat two definitions given in [4]. Consider a language L over Σ, and two homomorphisms h_1 and h_2 defined on Σ^*. We say that h_1 and h_2 are *compatible* (resp. *equivalent*) on L iff for some $w \in L$ (resp. for all $w \in L$) $h_1(w) = h_2(w)$ holds.

The following theorem is immediate from the definitions. It shows how the decision problems investigated in [4] can be considered as inclusion problems involving $E(h_1, h_2)$.

Theorem 2.1. Two homomorphisms h_1 and h_2 are equivalent (resp. compatible) on a language L if and only if L is contained in $E(h_1, h_2)$ (resp. L is not contained in the complement of $E(h_1, h_2)$).

In most cases we are able to decide whether a given language is contained in a given regular language. Thus, Theorem 2.1 shows that, as regards the homomorphism compatibility and equivalence problems, it is a very desirable situation that $E(h_1, h_2)$ is regular. More explicitly, we can express this as the following

Theorem 2.2. Assume that K is a family of (effectively given) languages such that the equation

$$L \cap R = \varphi \tag{1}$$

is decidable for L in K and R in the family of regular languages. Assume, further, that H is a family of homomorphisms such that $E(h_1, h_2)$ is regular for all h_1 and h_2 in H. Then it is decidable whether two homomorphisms h_1 and h_2 from H are equivalent on a language L in K.

Proof. The assertion is obvious if we can effectively construct the regular language $E(h_1, h_2)$: we just check the validity of (1) for R being the complement of $E(h_1, h_2)$. Otherwise, we run concurrently two semialgorithms, one for equivalence and, the other, for nonequivalence. The latter semialgorithm is obvious: we just consider an effective enumeration $w_0, w_1, w_2, \ldots$ for L and check whether $h_1(w_i)=h_2(w_i)$. For the semialgorithm A for equivalence, let $R_0, R_1, R_2, \ldots$ be an effective enumeration of regular languages. In the $(i+1)$ st step of A, we consider R_i and check whether h_1 and h_2 are equivalent on R_i. (This can be done by a result in [4], the result being easy enough to verify also directly.) If the answer is positive, we check the validity of (1) for R being the complement of R_i. The correctness and termination of this algorithm are now obvious. (A similar argument was used already in [3].) □

Under the additional assumption that $E(h_1, h_2)$ can be found effectively, we can extend Theorem 2.2 to the compatibility problem: it is decidable whether two homomorphisms h_1 and h_2 from H are compatible on a language L in K.

A very interesting and important class of homomorphisms for which $E(h_1, h_2)$ is always regular consists of the elementary homomorphisms introduced in [5] and studied further in [6]. By definition, a homomorphism $h: \Sigma^* \to \Sigma_1^*$ is *elementary* if there is no alphabet Σ_2 of smaller cardinality than Σ such that h can be represented as $h=h_2h_1$, where

$$h_1: \ \Sigma^* \to \Sigma_2^* \quad \text{and} \quad h_2: \ \Sigma_2^* \to \Sigma_1^*.$$

The following theorem is established in [6]. A modified version of it will be established also in Section 3 below.

Theorem 2.3. For elementary homomorphisms h_1 and h_2, $E(h_1, h_2)$ is regular.

One of the most famous problems in formal language theory during recent years has been the DOL equivalence problem: given two homomorphisms h_1 and h_2 mapping Σ^* into Σ^* and a word w in Σ^*, decide whether or not

$$h_1^i(w) = h_2^i(w)$$

holds for all $i \geqq 0$. A decision method was given in [2] and [3]. The notion of an elementary homomorphism seems to capture the essense of the problem and, consequently, the solution given in [6] avoids many of the difficulties present in the earlier solution. As regards the DOL equivalence problem, the reader is referred also to [7] and [8]. Clearly, the DOL equivalence problem amounts to deciding whether or not the DOL language consisting of all words $h_1^i(w)$, where $i \geqq 0$, is contained in $E(h_1, h_2)$.

The notion of balance, defined originally in [2], turns out to be very useful in discussing the regularity of $E(h_1, h_2)$.

Consider two homomorphism h_1 and h_2 defined on Σ^* and a word w in Σ^*. Then the *balance* of w is defined by

$$\beta(w) = \lg(h_1(w)) - \lg(h_2(w)).$$

(Thus, $\beta(w)$ is an integer depending, apart from w, also on h_1 and h_2. We write it simply $\beta(w)$ because the homomorphisms, as well as their ordering, will always be clear from the context.) This definition is in accordance with [4], the notion of balance defined in [2] equals $|\beta(w)|$ in our notation.

It is immediate that β is a homomorphism of Σ^* into the additive monoid of all integers. Consequently, we can write

$$\beta(w_1 w_2) = \beta(w_1) + \beta(w_2)$$

which shows that the balance of a word w depends only on the Parikh vector of w.

We say that the pair (h_1, h_2) has *k-bounded balance* on a given language L if k is an integer $\geqq 0$ and

$$|\beta(w)| \leqq k$$

holds for all initial subwords w of the words in L.

The property of having bounded balance gives a method of deciding homomorphism equivalence, a point exploited in detail in [4].

For $k \geqq 0$, we denote by $E_k(h_1, h_2)$ the largest subset of $E(h_1, h_2)$ such that the pair (h_1, h_2) has k-bounded balance on $E_k(h_1, h_2)$. Clearly, for all k,

$$E_k(h_1, h_2) \subseteqq E_{k+1}(h_1, h_2)$$

and

$$E(h_1, h_2) = \bigcup_{i=0}^{\infty} E_i(h_1, h_2). \tag{2}$$

The following theorem was established in [8], essentially the same result being contained also in [2].

Theorem 2.4. For each $k \geqq 0$ and arbitrary homomorphisms h_1 and h_2, the language $E_k(h_1, h_2)$ is regular.

The relation (2) and Theorem 2.4 show that $E(h_1, h_2)$ can always be approximated by a sequence of regular languages. Note also that $E_0(h_1, h_2) = \{\lambda\}$ or Σ'^*, where Σ' is the subset of Σ consisting of all letters a for which $h_1(a) = h_2(a)$.

We conclude this section by showing that all languages in F_E possess a special property. Indeed, consider any language $E(h_1, h_2) = L$. By definition, whenever w_1 and w_2 are in L then so is $w_1 w_2$. This implies that $L = L^*$, i.e., L is a star language (a star event in the sense of [1]). The minimal star root of L, i.e., the smallest language M satisfying $L = M^*$, consists of all words w of L such that no proper initial subword of w is in L. Same results hold true also with respect to languages $E_k(h_1, h_2)$. These results are summarized in the following

Theorem 2.5. Every language L in F_E is a star language. The subset M of L, consisting of all words w such that no proper initial subword of w is in L, is the smallest language satisfying

$$M^* = L.$$

For each k, h_1, h_2, $E_k(h_1, h_2)$ is a star language.

Theorem 2.5 shows, for instance, that F_E contains no finite languages with the exception of $\varphi^* = \{\lambda\}$. Since, for any language L and homomorphism h, we have $h(L^*) = (h(L))^*$, it shows also that even if we take morphic images of the languages of F_E, we get only star languages. However, it will be seen in Section 4 that every recursively enumerable language is obtained by a deterministic *gsm* mapping from a language in F_E.

On the other hand, it is clear that only star languages of a special type are in F_E: F_E does not even contain all star languages with a finite star root. This follows by the next theorem, the proof of which is obvious.

Theorem 2.6. Whenever a language L in F_E contains the words w_1 and $w_1 w_2$, then it contains also the word w_2.

3. Regular equality sets

We begin this section with two examples. Consider first two homomorphisms h_1 and h_2 mapping $\{a, b\}^*$ into $\{a\}^*$, defined by

$$h_1(a) = h_2(b) = a, \quad h_2(a) = h_1(b) = aa.$$

It is immediately verified that $E(h_1, h_2)$ consists of all words w such that the number of occurrences of a in w equals that of b in w. Thus, we have here a simple example of a context-free nonregular equality set.

Consider, next, the two homomorphisms g_1 and g_2 defined by

$$g_1(a) = ab, \quad g_1(b) = b, \quad g_1(c) = a,$$

$$g_2(a) = a, \quad g_2(b) = b, \quad g_2(c) = ba.$$

Clearly, $E(g_1, g_2)$ is now denoted by the regular expression $(ab^*c \cup b)^*$. In this case, $E(g_1, g_2)$ is a regular language possessing no finite star root.

We now return to the equation (2) and show that $E(h_1, h_2)$ is regular exactly in case the right side can be replaced by a finite union, i.e., $E(h_1, h_2)$ equals one of the sets $E_k(h_1, h_2)$.

Theorem 3.1. The set $E(h_1, h_2)$ is regular if and only if, for some k,

$$E(h_1, h_2) = E_k(h_1, h_2). \tag{3}$$

Proof. The "if"-part follows by Theorem 2.4. To prove the "only if"-part, assume that $E(h_1, h_2) = L$ is regular. Thus, the homomorphisms h_1 and h_2 are equivalent on the regular language L. This implies that the pair (h_1, h_2) has k-bounded balance on L, for some k and, thus, (3) holds true. (The implication is established in [4]. It follows from the observation that if a word w causes a loop in the minimal finite automaton accepting L, then $\beta(w) = 0$. Thus, an upper bound for the balance of initial subwords of the words in L can be computed by considering such words only which cause a transition from the initial state to one of the final states without loops. If n is the number of states in the automaton and

$$t = \max\{|\beta(a)| \mid a \text{ in } \Sigma\}$$

then

$$k \leq t[(n-1)/2].$$

One can show by examples that this estimate is the best possible in the general case.) □

By Theorems 2.3 and 3.1, we obtain now the following

Theorem 3.2. If h_1 and h_2 are elementary homomorphisms then there exists a k such that

$$E(h_1, h_2) = E_k(h_1, h_2).$$

Remark. Theorem 3.1 shows the importance of the notion of balance in characterizing the regular sets $E(h_1, h_2)$. We want to point out that we are dealing here with a property typical for equality sets which cannot be deduced from (2) and properties of star languages. More specifically, there are regular star languages L_i^*, $i \geqq 0$, satisfying

$$L_i^* \subseteqq L_{i+1}^*, \quad \text{for all} \quad i,$$

and

$$\bigcup_{i=0}^{\infty} L_i^* = L^*$$

and, furthermore, $L^* \neq L_i^*$ for all i, although L^* is regular. An example is given by

$$L_i = \bigcup_{j \leqq i} ab^j c.$$

Thus, Theorem 3.1 cannot be deduced from (2) and properties of star languages.

We want to emphasize that $E(h_1, h_2)$ may be regular although h_1 and h_2 are not elementary, i.e., the converse of Theorem 2.3 is not valid. For instance, define

$$h_1(a) = a, \quad h_1(b) = h_1(c) = b,$$

$$h_2(a) = a, \quad h_2(b) = h_2(c) = c.$$

Then $E(h_1, h_2) = a^*$ although neither h_1 nor h_2 is elementary.

Apart from the sequence $E_k(h_1, h_2)$, $k = 0, 1, \ldots$, there seems to exist no other approximating sequence for $E(h_1, h_2)$ with similar properties (in particular, Theorem 3.1).

This section is concluded by a result exhibiting a special case in which the language $E(h_1, h_2)$ is always context-free. We point out that it will be shown in Section 5 that the general problem of determining whether a given language in F_E is regular (resp. context-free) is undecidable.

Theorem 3.3. Assume that h_1 and h_2 are homomorphisms mapping Σ^* into $\{a\}^*$. Then the language $E(h_1, h_2)$ is context-free but not necessarily regular.

Proof. The second assertion follows by the example given at the beginning of this section. To show that $E(h_1, h_2) = L$ is context-free, we assume that $\Sigma = \{a_1, \ldots, a_k\}$ and

$$h_1(a_i) = a^{m_i}, \quad h_2(a_i) = a^{n_i}, \quad i = 1, \ldots, k.$$

We denote $d_i = m_i - n_i$. By a suitable renumbering of the alphabet Σ, we may assume the existence of numbers u and v, $0 \leqq u \leqq v \leqq k$, such that

$$d_i \text{ is} \begin{cases} 0 & \text{for} \quad 1 \leqq i \leqq u, \\ \text{positive for} & u+1 \leqq i \leqq v, \\ \text{negative for} & v+1 \leqq i \leqq k. \end{cases}$$

Consider now the language $L_1 = L \cap a_1^* a_2^* \dots a_k^*$. L_1 consists of all words w such that (i) the letters of the alphabet Σ occur in w in the "right" alphabetical order, and (ii)

$$d_{u+1}x_{u+1} + \dots + d_v x_v = (-d_{v+1})x_{v+1} + \dots + (-d_k)x_k, \tag{4}$$

where x_i denotes the number of occurrences of a_i in w. (Note that all the coefficients of x_i in (4) are positive.) But the validity of (4) can be checked by a deterministic one-counter machine M. Indeed, when reading a letter a_i with $u+1 \leqq i \leqq v$, M pushes d_i copies of the counter symbol, and when reading a letter a_i with $v+1 \leqq \leqq i \leqq k$, M pops d_i copies of the counter symbol. Hence, L_1 is a deterministic one-counter language.

On the other hand, $L = C(L_1)$, where C denotes the "commutative variant" of the language, i.e., $C(L_1)$ is the language obtained from L_1 by taking all permutations of its words. Because it is easy to see that $C(L_1)$ is context-free, we have concluded the proof. □

4. More general equality sets, their scope

We now turn to the discussion of the general question of the "size" and typical features of the family F_E. We show that every recursively enumerable language can be obtained by a deterministic *gsm* mapping from a language in F_E. By the remark made after Theorem 2.5, homomorphism is not sufficient for this purpose; all recursively enumerable languages cannot be obtained as morphic images of languages in F_E. However, we shall establish the following weaker result: if L_0 is a recursively enumerable language, then the language $(C(L_0))^*$ is a morphic image of a language in F_E. Here C denotes the commutative variant discussed in the proof of Theorem 3.3.

To understand the technical details in this section, familiarity with the proof of Theorem VIII.2.1 in [9] is required on part of the reader. In the examples and arguments below, we try to follow the notation of this proof as much as possible.

We begin with the following simple result.

Theorem 4.1. Every language in F_E is context-sensitive.

Proof. Consider an arbitrary $L = E(h_1, h_2)$. Let m be the maximum length among the words $h_1(a)$ and $h_2(a)$, where a ranges over Σ. Then L is accepted by a linear bounded automaton M whose work tape is at most m times the length of the input w. Indeed, M first writes $h_1(w)$ and $h_2(w)$ on two tracks, and makes then the comparison on its final run. □

We now give an example of a language in F_E which is not context-free. The example also serves the purpose of providing some intuitive background for the proof of Theorem 4.2.

The alphabet Σ in the example consists of the letters 1, 2, ..., 18. (Thus, two-digit numbers are viewed as single letters.) The target alphabet Σ_1 will become apparent in the following definition of h_1 and h_2. In the definition, letters a of Σ are listed in the first row, and the values $h_1(a)$ (resp. $h_2(a)$) in the second (resp. third) row.

1	2	3	4	5	6	7	8	9
BSc	c'	c	E	S'	S_1'	S_2'	S	S_1
B	c	c'	$c'E$	S	S_1	S_2	S'	S_1'

10	11	12	13	14	15	16	17	18
S_2	$S_1'S'S_2'$	λ	λ	λ	S_1SS_2	λ	λ	λ
S_2'	S	S	S_1	S_2	S'	S'	S_1'	S_2'

To show that $E(h_1, h_2)=L$ is not context-free, we argue as follows. Our example is constructed according to the proof of Theorem VIII.2.1 in [9] from the grammar G with the productions

$$S \to S_1SS_2, \quad S_1 \to \lambda, \quad S_2 \to \lambda, \quad S \to \lambda.$$

Note that the Szilard language of G is not context-free (cf. [9, p. 185]). This implies that L cannot be context-free because the Szilard language of G is obtained from L by a suitable homomorphism. Indeed, it suffices to erase all letters not "representing" applications of productions. (We can also get from L the language $\{a^nb^nc^n|\geq 1\}$ by taking first the intersection with a regular language and then a morphic image. The intuitive idea behind this is to apply the four productions in the order they are listed above.) □

We want to emphasize that if we just want an example of a non-context-free language in F_E then the example given above is unnecessarily complicated. (For instance, the distinction between primed and non-primed letters is superfluous from this point of view. It is, however, quite essential in other arguments because we do not want a solution to the Post Correspondence Problem starting from the "middle".) The above example serves the additional purpose of making the reader familiar with the idea behind the proof of the following theorem.

Theorem 4.2. For every recursively enumerable language L_0, one can effectively construct a language L in F_E and a deterministic generalized sequential machine M such that $L_0=M(L)$.

Proof. Following the notation of [9], we assume that L_0 is generated by the type-0 grammar $G=(V_N, V_T, a_1, F)$, where

$$V = V_N \cup V_T = \{a_1, \dots, a_r\}, \quad F = \{P_i \to Q_i | 1 \leq i \leq n\},$$
$$V_T = \{a_s, \dots, a_r\}, \quad 1 < s \leq r.$$

Denote $V'=\{a'|a\in V\}$. Thus, for any word Q over V, we can consider the "primed version" Q' obtained from Q by replacing every letter a with a'.

Without loss of generality, we assume that $a_1 \to a_1$ is one of the productions in F. (This is done because of the same reason as in the proof of Theorem VIII.2.1 in [9]: to get the right parity for the length of a derivation. Note, however, that we do not have to eliminate the λ-productions over F as we did in [9].)

We now introduce two homomorphisms h_1 and h_2 mapping Σ^* into Σ_1^*, where

$$\Sigma = \{1, 2, \ldots, 2r+2n+4,\ a_s, \ldots, a_r\},$$
$$\Sigma_1 = V \cup V' \cup \{B, c, c'\}.$$

Again, the homomorphisms are given by the following table listing $h_1(a)$ and $h_2(a)$ below a. In the table, i (resp. j) ranges through the numbers $1, \ldots, r$ (resp. $1, \ldots, n$), and x through the letters $a_s, \ldots, a_r$.

x	1	2	3	4	$4+i$	$4+r+i$	$4+2r+j$	$4+2r+n+j$
λ	Ba_1c	c'	c	λ	a_i'	a_i	Q_j'	Q_j
x	B	c	c'	c'	a_i	a_i'	P_j	P_j'

Consider the language $L = E(h_1, h_2)$. We denote

$$\Sigma_T = \{a_s, \ldots, a_r\}, \quad \Sigma_N = \{1, 2, 3, 5, \ldots, 2r+2n+4\}.$$

(Thus, Σ_T and Σ_N are subalphabets of Σ. The former consists of all "letters" and, the latter, of all "numbers" except 4.)

Let now M be the deterministic generalized sequential machine which, when reading an input word w over Σ, checks whether w is of the following form: a nonempty word over Σ_N, followed by exactly one occurrence of the letter 4, followed by a (possibly empty) word w' over Σ_T. In the positive case, the output is w', in the negative case no output is produced.

Comparing the construction with the proof of Theorem VIII.2.1 in [9], it is now easy to see that $L_0 = M(L)$ holds true. Indeed, the above construction differs from that in [9] only with respect to the letters 4 and $a_s, \ldots, a_r$. But the machine M makes sure that the effect of these letters is the same as that of α_4 and β_4 in [9]. Thus, M outputs exactly the words of the original language L_0. Note, in particular, that we have λ as an output exactly in case λ is in L_0. L, as every equality language, contains λ but M does not accept it as an input. □

Remark 1. Let h be the homomorphism mapping the letters of Σ_T into themselves and erasing the other letters of Σ. By the proof above, we get the representation

$$L_0 = h(L \cap \Sigma_N^+ 4 \Sigma_T^*).$$

(By an easy modification, 4 can be eliminated.) Thus, every type-0 language is obtained from a language L in F_E by intersecting L with a regular language and then taking a morphic image (under a very simple morphism) of the result. This representation theorem has been obtained by another method by G. Rozenberg (personal communication).

Remark 2. We have already pointed out why there are recursively enumerable (in fact, even finite) languages L_0 not representable in the form $L_0 = h(L)$, where h is a morphism and L is in F_E. Clearly, by Theorem 4.1, the operation of taking intersections with regular languages alone is not sufficient for such a representation of recursively enumerable languages in terms of equality languages. As regards homomorphisms, the following theorem gives a weaker result.

Theorem 4.3. For every recursively enumerable language L_0, one can effectively construct a language L in F_E and a homomorphism h mapping every letter either to itself or to λ such that

$$(C(L_0))^* = h(L). \tag{5}$$

Proof. The language L is constructed exactly as in the proof of Theorem 4.2. The only additional requirement we have now is that in the original grammar G terminal letters occur in productions of the form $B \rightarrow b$, where B is a nonterminal and b a terminal, only. The homomorphism h is defined as in Remark 1 above.

To prove (5), note first that the right side is included in the left side. This follows because if we take one of the letters $x = a_s, \ldots, a_r$ "too early" to a word in L, then the terminal letter x has already been derived according to G. The reverse inclusion is obtained by noting that any word $w = b_1 \ldots b_t$ in L_0 can be derived by deriving first the corresponding nonterminal word $B_1 \ldots B_t$. From the latter, the terminal letters b_i can be introduced in any order and, hence, any permutation of w is in $h(L)$. Clearly, $h(L) = (h(L))^*$. □

The following result is now immediate from Theorem 4.3.

Theorem 4.4. For every recursively enumerable star language L_0 over a one-letter alphabet $\{a\}$, one can effectively construct a language L in F_E and a homomorphism h, mapping a into itself and erasing other letters, such that $L_0 = h(L)$.

It is an open problem whether or not Theorem 4.4 holds true for arbitrary recursively enumerable star languages, i.e., whether or not (5) in Theorem 4.3 can be replaced by the equation

$$L_0^* = h(L).$$

5. Decidability

In this final section we consider some decision problems for F_E, as well as some applications to other decision problems, in particular, problems concerning homomorphism equivalence.

Clearly, membership is decidable for languages in F_E. Such a language is never empty because it always contains λ. An arbitrary Post Correspondence Problem PCP defines a language L_{PCP} in F_E such that L_{PCP} is infinite if and only if PCP has a solution. Hence, infinity is undecidable for languages in F_E. Since $\{\lambda\}$ belongs to F_E, we see in the same way that the equivalence problem is undecidable for F_E, i.e., there is no algorithm for determining of two given languages in F_E whether or not they are the same.

These results are summarized in the following

Theorem 5.1. Membership problem is decidable for languages in F_E. Emptiness problem is trivial but infinity problem undecidable for languages in F_E. Given a language L in F_E, it is undecidable whether $L = \{\lambda\}$. Hence, equivalence problem is undecidable for F_E.

Note that it is decidable whether a language L in F_E equals Σ^*.

We have already pointed out that in some investigations it is very desirable that $E(h_1, h_2)$ is regular. However, the following theorem shows that this property is undecidable.

Theorem 5.2. It is undecidable whether a language in F_E is (i) regular, (ii) context-free.

Proof. We consider the following modified Post Correspondence Problems PCP over an alphabet V

$$(\alpha_1, \ldots, \alpha_n), \quad (\beta_1, \ldots, \beta_n), \tag{6}$$

where

$$\alpha_1 = BA, \quad \beta_1 = B, \quad \alpha_2 = C, \quad \beta_2 = A,$$

and every solution to PCP must begin with the indices 1, 2. Furthermore, it is assumed that B and A do not occur in any of the words $\alpha_3, \ldots, \alpha_n, \beta_3, \ldots, \beta_n$. Clearly, there is no algorithm to solve such modified PCP's.

We argue now indirectly and assume that either (i) or (ii) is decidable. We show that we can then solve also the modified PCP. Let (6) be an arbitrary given instance. We construct new words

$$(\alpha_{n+1}, \ldots, \alpha_{n+m}), \quad (\beta_{n+1}, \ldots, \beta_{n+m}) \tag{7}$$

over an alphabet consisting of C and letters not in V such that (i) the PCP (7) has no solution, and (ii) the language L over the alphabet $\{n+1, \ldots, n+m\}$ consisting of words $i_1 \ldots i_t$ such that

$$C\alpha_{i_1} \ldots \alpha_{i_t} = \beta_{i_1} \ldots \beta_{i_t} C$$

is not context-free. Such a construction is possible along the lines of the example given in Section 4. Condition (i) is taken care of by making sure that, for no pair of words (α_i, β_i), $i = n+1, \ldots, n+m$, one of the words is an initial subword of the other.

Let h be the homomorphism defined on the monoid $\{1, \ldots, n+m\}^*$ by

$$h(i) = \lambda \quad \text{for} \quad i \leqq n, \quad h(i) = i \quad \text{for} \quad i > n.$$

Furthermore, let h_1 and h_2 be homomorphisms defined by

$$h_1(i) = \alpha_i, \quad h_2(i) = \beta_i, \quad i = 1, \ldots, n+m.$$

Consider the language $E(h_1, h_2)$.

Assume first that our original given PCP (6) has no solution. Then it is immediate by the definition of (7) that $E(h_1, h_2) = \{\lambda\}$, i.e., $E(h_1, h_2)$ is regular.

Assume, next, that the PCP (6) has a solution. In this case, $E(h_1, h_2)$ consists of λ and of all words over $1, \ldots, n+m$ of the form

$$1w2w',$$

where $12w'$ is a solution of (6) and w is in L. Hence, $h(E(h_1, h_2)) = L$. (Note that λ is in L.) This implies that $E(h_1, h_2)$ is not context-free.

Thus, if either (i) or (ii) in the statement of Theorem 5.2 were decidable, we would be solving the modified PCP (6), a contradiction. $\square$

Although it is undecidable whether a language in F_E is regular, we conjecture that the converse is decidable, i.e., it is decidable whether a given regular language is in F_E. The proof of this conjecture requires results stronger than Theorems 2.5 and 2.6. (Note that some other similar results can be easily established. For instance, whenever a word w^i, $i>1$, is in a language L in F_E, then also w is in L.)

We have already pointed out the significance of $E(h_1, h_2)$ in some decision problems, notably the problem of homomorphism equivalence. It was conjectured in [4] that homomorphism equivalence is decidable for indexed languages. By Theorems 2.2 and 2.3, we get the following partial result.

Theorem 5.3. It is decidable whether two given elementary homomorphisms are equivalent on a given indexed language.

The fact that $E(h_1, h_2)$ is context-free (in situations like the one exhibited in Theorem 3.3) is not so easily applicable to decision problems. The reason is that inclusion of a given language in a context-free language is, in general, a difficult problem. Of course, results corresponding to Theorem 2.2 can be formulated also in this case.

In [4], the following generalization (referred to as the DTOL sequence equivalence) of the DOL equivalence problem was investigated: given two pairs of homomorphisms (g_1, g_2) and (h_1, h_2) and a word w, decide whether

$$g_{i_1} \dots g_{i_t}(w) = h_{i_1} \dots h_{i_t}(w)$$

holds for all words $i_1 \dots i_t$ over the alphabet $\{1, 2\}$. It was shown in [4] that more general DTOL equivalence problems can be reduced to this problem.

Since the equation (1) is decidable for DTOL languages L, we get the following partial result by an argument similar to the one used in the proof of Theorem 2.2.

Theorem 5.4. The DTOL sequence equivalence problem is decidable for elementary homomorphisms g_1, g_2, h_1, h_2. It is also decidable whether two given elementary homomorphisms are equivalent on an arbitrary given DTOL language.

The second sentence of Theorem 5.4 follows also by Theorem 5.3. That Theorem 5.4 cannot be used to solve the DTOL sequence equivalence problem (in the same way as the DOL equivalence problem was solved in [6]) is due to the fact that the analogous decomposition technique is not valid for DTOL systems.

6. Conclusion

Apart from their importance in certain decision problems, the languages $E(h_1, h_2)$ seem to be rather interesting also from other points of view. We have established some of their basic properties. However, there are many open problems. Many aspects (such as closure properties) of these interesting languages were not discussed at all in this paper.

MATHEMATICS DEPARTMENT
UNIVERSITY OF TURKU
FINLAND

References

[1] Brzozowski, J. A., Roots of star events, *J. Assoc. Comput. Mach.*, v. 14, 1967, pp. 466—477.
[2] Culik, K. II, On the decidability of the sequence equivalence problem for DOL-systems, *Theoretical Computer Science*, v. 3, 1977, pp. 75—84.
[3] Culik, K. II and I. Fris, The decidability of the equivalence problem for DOL-systems, *Information and Control*, v. 35, 1977, pp. 20—39.
[4] Culik, K. II and A. Salomaa, On the decidability of homomorphism equivalence for languages, *J. Comput. System Sci.*, to appear.
[5] Ehrenfeucht, A. and G. Rozenberg, Simplifications of homomorphisms, *Information and Control*, to appear.
[6] Ehrenfeucht, A. and G. Rozenberg, Elementary homomorphisms and a solution of the DOL sequence equivalence problem, *Theoretical Computer Science*, to appear.
[7] Rozenberg, G. and A. Salomaa, The mathematical theory of *L* systems, *Advances in Information Systems Science*, Vol. 6, J. Tou (ed.) Plenum Press, New York, 1976, pp. 161—206.
[8] Salomaa, A., DOL equivalence: the problem of iterated morphisms, *Bulletin of the EATCS*, to appear.
[9] Salomaa, A., *Formal languages*, Academic Press, New York, 1973.

(Received Jan. 23, 1978)

Ein Ansatz zum Entscheidungsverfahren für eine Formelklasse der Prädikatenlogik mit Identität

Von K. Schütte

Herrn Professor László Kalmár zum Gedächtnis

Das Entscheidungsproblem der Prädikatenlogik ohne Identität ist für pränexe Formelklassen vollständig gelöst, nachdem in [6] die Klasse der $\forall\exists\forall$-Formeln als unentscheidbar nachgewiesen wurde. Dabei stellte sich als stärkste entscheidbare Formelklasse diejenige Formelklasse heraus, für die von L. Kalmár [3] im Jahre 1933 ein Entscheidungsverfahren gegeben wurde.

Für die Prädikatenlogik mit Identität ist jedoch die Entscheidbarkeit der entsprechenden Formelklasse ein bisher noch ungelöstes Problem. (In [2] und [4] war irrtümlich erklärt, daß die dort angegebenen Verfahren auch bei Hinzunahme des Gleichheitszeichens zum Ziel führen.)

Im folgenden wird ein Ansatz zur Behandlung des Entscheidungsproblems für die problematische Klasse der $\forall\forall\exists$-Formeln mit Identität entwickelt, aus dem hervorgeht, welche Schwierigkeiten hierbei auftreten und was zu beweisen wäre, falls die betreffende Formelklasse entscheidbar ist.

Wir gehen aus von einer pränexen Formel

$$\forall x \forall y \exists z A(a_1, \ldots, a_m, x, y, z) \tag{1}$$

der Prädikatenlogik mit Identität, in der keine anderen freien Objektvariablen als $a_1, \ldots, a_m$ $(m \geqq 1)$ auftreten. Als Primformeln, aus denen sich die Formel $A(a_1, \ldots, a_{m+3})$ mittels aussagenlogischer Junktoren zusammensetzt, dürfen folgende vier Arten auftreten:

1. Die Konstanten $\top$ (verum) und $\bot$ (falsum),
2. Aussagenvariablen,
3. Prädikaten-Primformeln der Gestalt

$$P_j^k a_{i_1} \ldots a_{i_k} \quad (k \geqq 1),$$

wobei P_j^k eine k-stellige Prädikatenvariable ist,

4. Gleichungen $a_i = a_j$.

W sei die Menge der Aussagenvariablen, die in der Formel (1) auftreten, $\{W_1, \ldots, W_t\}$ die Menge aller Teilmengen von W. Für $i = 1, \ldots, t$ sei $A_i(a_1, \ldots, a_{m+3})$

diejenige Formel, die sich aus $A(a_1, \dots, a_{m+3})$ ergibt, wenn jede Aussagenvariable der Menge W_i durch $\top$ und jede andere Aussagenvariable durch $\perp$ ersetzt wird. Ferner sei

$$C_1 :\equiv \top, \quad C_n :\equiv \bigwedge_{\substack{i,j=1\\(i<j)}}^{n} a_i \neq a_j \quad \text{für} \quad 1 < n \leqq m.$$

Die Formel (1) ist genau dann erfüllbar, wenn es eine positive ganze Zahl $n \leqq m$ und eine Abbildung f von $\{1, \dots, m\}$ auf $\{1, \dots, n\}$ gibt, so daß eine der Formeln

$$\forall x \forall y \exists z A_i(a_{f1}, \dots, a_{fm}, x, y, z) \wedge C_n \quad (i = 1, \dots, t)$$

erfüllbar ist. Es genügt daher, ein Entscheidungsverfahren für die Erfüllbarkeit von Formeln der Gestalt

$$\forall x \forall y \exists z A^*(a_1, \dots, a_n, x, y, z) \wedge C_n \quad (n \geqq 1) \tag{2}$$

zu entwickeln, wobei $A^*(a_1, \dots, a_n, x, y, z)$ eine quantorenfreie Formel ist, die keine Aussagenvariablen und keine anderen Objektvariablen als $a_1, \dots, a_n, x, y, z$ enthält.

$$B^*(a_1, \dots, a_n, x, y)$$

sei die Formel

$$A^*(a_1, \dots, a_n, x, y, x) \vee A^*(a_1, \dots, a_n, x, y, y) \bigvee_{k=1}^{n} A^*(a_1, \dots, a_n, x, y, a_k).$$

Für $i = 1, \dots, n$ definieren wir

$$F_i^*(a_1, \dots, a_n, x, z) :\equiv A^*(a_1, \dots, a_n, a_i, x, z)$$

$$G_i^*(a_1, \dots, a_n, x) :\equiv A^*(a_1, \dots, a_n, a_i, x, x) \bigvee_{k=1}^{n} A^*(a_1, \dots, a_n, a_i, x, a_k)$$

$$F_{n+i}^*(a_1, \dots, a_n, x, z) :\equiv A^*(a_1, \dots, a_n, x, a_i, z)$$

$$G_{n+i}^*(a_1, \dots, a_n, x) :\equiv A^*(a_1, \dots, a_n, x, a_i, x) \bigvee_{k=1}^{n} A^*(a_1, \dots, a_n, x, a_i, a_k)$$

$$F_{2n+1}^*(a_1, \dots, a_n, x, z) :\equiv A^*(a_1, \dots, a_n, x, x, z)$$

$$G_{2n+1}^*(a_1, \dots, a_n, x) :\equiv A^*(a_1, \dots, a_n, x, x, x) \bigvee_{k=1}^{n} A^*(a_1, \dots, a_n, x, x, a_k)$$

Für $i, j = 1, \dots, n$ definieren wir

$$H_{(i-1)\cdot n+j}^*(a_1, \dots, a_n, z) :\equiv A^*(a_1, \dots, a_n, a_i, a_j, z)$$

$$K_{(i-1)\cdot n+j}^*(a_1, \dots, a_n) :\equiv \bigvee_{k=1}^{n} A^*(a_1, \dots, a_n, a_i, a_j, a_k).$$

Zur Abkürzung setzen wir $r := 2n+1$ und $s := n^2$. In den Formeln $A^*(a_1, \dots, a_{n+3})$, $B^*(a_1, \dots, a_{n+2})$, $F_i^*(a_1, \dots, a_{n+2})$, $G_i^*(a_1, \dots, a_{n+1})$ $(i = 1, \dots, r)$ und $H_i^*(a_1, \dots, a_{n+1})$, $K_i^*(a_1, \dots, a_n)$ $(i = 1, \dots, s)$ ersetzen wir jede Gleichung $a_j = a_j$ durch $\top$ und jede Gleichung $a_j = a_k$ $(j \neq k)$ durch $\perp$. Hierdurch ergeben sich Formeln $A(a_1, \dots, a_{n+3})$,

$B(a_1, \ldots, a_{n+2})$, $F_i(a_1, \ldots, a_{n+2})$, $G_i(a_1, \ldots, a_{n+1})$, $H_i(a_1, \ldots, a_{n+1})$ und $K_i(a_1, \ldots, a_n)$, in denen keine Gleichung auftritt. Wir schreiben kurz $A(x, y, z)$, $B(x, y)$, $F_i(x, z)$, $G_i(x)$, $H_i(z)$ und K_i für $A(a_1, \ldots, a_n, x, y, z)$, $B(a_1, \ldots, a_n, x, y)$, $F_i(a_1, \ldots, a_n, x, z)$, $G_i(a_1, \ldots, a_n, x)$, $H_i(a_1, \ldots, a_n, z)$ und $K_i(a_1, \ldots, a_n)$. Ferner definieren wir

$$U_n(x) :\equiv \bigwedge_{i=1}^{n} x \neq a_i.$$

Die Formel (2) ist dann äquivalent mit der Formel

$$\left\{\begin{array}{l} \forall x \forall y \{U_n(x) \wedge U_n(y) \wedge x \neq y \rightarrow \exists z[A(x, y, z) \wedge U_n(z) \wedge x \neq z \wedge y \neq z] \vee B(x, y)\} \\ \bigwedge_{i=1}^{r} \forall x \{U_n(x) \rightarrow \exists z[F_i(x, z) \wedge U_n(z) \wedge x \neq z] \vee G_i(x)\} \\ \bigwedge_{i=1}^{s} \{\exists z[H_i(z) \wedge U_n(z)] \vee K_i\} \wedge C_n. \end{array}\right. \tag{3}$$

Im folgenden sei F eine Formel der Gestalt (3). Dabei sind die $A(x, y, z)$, $B(x, y)$, $F_i(x, z)$, $G_i(x)$, $H_i(z)$ und K_i quantorenfreie Formeln, die keine Aussagenvariablen, keine Gleichungen und außer den angegebenen Objektvariablen höchstens die Objektvariablen $a_1, \ldots, a_n$ enthalten. n, r und s sind positive ganze Zahlen.

P sei eine nichtleere endliche Menge von Prädikatenvariablen, die alle in F auftretenden Prädikatenvariablen enthält. Ist V eine nichtleere endliche Menge von Objektvariablen, so verstehen wir unter einer *vollständigen V-Konjunktion* eine widerspruchsfreie Konjunktion aus Primformeln und negierten Primformeln, in der jede Prädikaten-Primformel, die sich aus den Prädikatenvariablen der Menge P mit den Objektvariablen der Menge V bilden läßt, genau einmal (negiert oder nichtnegiert) auftritt und in der kein anderes Konjunktionsglied vorkommt. Unter einer *V-Normalform* verstehen wir dann eine Disjunktion

$$\bigvee_{i=1}^{m} A_i \quad (m \geqq 0)$$

aus paarweise inäquivalenten vollständigen V-Konjunktionen $A_1, \ldots, A_m$, wobei es sich im Fall $m=0$ um die Formel $\perp$ handeln soll.

Zu den in (3) auftretenden Formeln $A(x, y, z)$, $B(x, y)$, $F_i(x, z)$, $G_i(x)$, $H_i(z)$ lassen sich nun eine äquivalente $\{a_1, \ldots, a_n, x, y, z\}$-Normalform

$$\bigvee_{j=1}^{m_0} A_j(x, y, z),$$

eine äquivalente $\{a_1, \ldots, a_n, x, y\}$-Normalform

$$\bigvee_{j=1}^{n_0} B_j(x, y),$$

äquivalente $\{a_1, \ldots, a_n, x, z\}$-Normalformen

$$\bigvee_{j=1}^{m_i} F_{ij}(x, z) \quad (i = 1, \ldots, r),$$

äquivalente $\{a_1, \ldots, a_n, x\}$-Normalformen

$$\bigvee_{j=1}^{n_i} G_{ij}(x) \quad (i = 1, \ldots, r),$$

äquivalente $\{a_1, \ldots, a_n, z\}$-Normalformen

$$\bigvee_{j=1}^{p_i} H_{ij}(z) \quad (i = 1, \ldots, s)$$

und äquivalente $\{a_1, \ldots, a_n\}$-Normalformen

$$\bigvee_{j=1}^{q_i} K_{ij} \quad (i = 1, \ldots, s)$$

bilden.

Es läßt sich entscheiden, ob die Formel F in einem Bereich von höchstens $n+1$ Elementen erfüllbar ist. Wir setzen im folgenden voraus, daß sie in keinem Bereich von weniger als $n+2$ Elementen erfüllbar ist. Sie ist dann unerfüllbar, wenn eine der Zahlen m_i+n_i $(i=0, \ldots, r)$ oder p_i+q_i $(i=1, \ldots, s)$ gleich 0 ist. Wir nehmen nun an, daß alle diese Zahlen positiv sind.

Unter einem *Indexsystem* der Formel F verstehen wir dann ein System

$$M = \langle M_0, \ldots, M_r,\ N_0, \ldots, N_r,\ P_1, \ldots, P_s,\ Q_1, \ldots, Q_s\rangle$$

von Teilmengen $M_i \subset \{1, \ldots, m_i\}$, $N_i \subset \{1, \ldots, n_i\}$, $P_i \subset \{1, \ldots, p_i\}$ und $Q_i \subset \{1, \ldots, q_i\}$, von denen keine der Mengen $M_i \cup N_i$ $(i=0, \ldots, r)$ und $P_i \cup Q_i$ $(i=1, \ldots, s)$ leer ist. Bezüglich eines solchen Indexsystems führen wir folgende Bezeichnungen ein:

Als *M-Hauptglieder* bezeichnen wir die Formeln $A_j(x, y, z)$ $(j \in M_0)$.

Als *M-Doppelglieder* bezeichnen wir die Formeln $B_j(x, y)$, $B_j(y, x)$, $(j \in N_0)$, $F_{ij}(x, y)$, $F_{ij}(y, x)$ $(i=1, \ldots, r; j \in M_i)$ und alle Formeln, die sich aus einer der Formeln $A_j(x, y, z)$, $A_j(x, z, y)$, $A_j(y, x, z)$, $A_j(y, z, x)$, $A_j(z, x, y)$, $A_j(z, y, x)$ $(j \in M_0)$ ergeben, wenn alle Konjuktionsglieder, in denen die Variable z auftritt, gestrichen werden.

Als *M-Einzelglieder* bezeichnen wir die Formeln $G_{ij}(x)$ $(i=1, \ldots, r;\ j \in N_i)$, $H_{ij}(x)$ $(i=1, \ldots, s;\ j \in P_i)$ sowie jede Formel, die sich aus einem M-Doppelglied $D(x, y)$ ergibt, wenn alle Konjunktionsglieder, in denen die Variable y auftritt, gestrichen werden.

Als *M-Grundglieder* bezeichnen wir die Formeln K_{ij} $(i=1, \ldots, s; j \in Q_i)$ sowie jede Formel, die sich aus einem M-Einzelglied $E(x)$ ergibt, wenn alle Konjunktionsglieder, in denen die Variable x auftritt, gestrichen werden.

$E_1(x), \ldots, E_e(x)$ sei nun eine maximale Folge von paarweise inäquivalenten M-Einzelgliedern. Mit F_M bezeichnen wir dann die Formel

$$\begin{cases} \forall x \forall y\{U_n(x) \wedge U_n(y) \wedge x \neq y \rightarrow \bigvee\limits_{j \in M_0} \exists z[A_j(x, y, z) \wedge U_n(z) \wedge x \neq z \wedge y \neq z] \bigvee\limits_{j \in N_0} B_j(x, y)\} \\ \bigwedge\limits_{i=1}^{r} \forall x\{U_n(x) \rightarrow \bigvee\limits_{j \in M_i} \exists z[F_{ij}(x, z) \wedge U_n(z) \wedge x \neq z] \bigvee\limits_{j \in N_i} G_{ij}(x)\} \\ \bigwedge\limits_{i=1}^{s} \{\bigvee\limits_{i \in P_i} \exists z[H_{ij}(z) \wedge U_n(z)] \bigvee\limits_{j \in Q_i} K_{ij}\} \bigwedge\limits_{i=1}^{e} \exists x[E_i(x) \wedge U_n(x)] \wedge C_n. \end{cases} \tag{4}$$

Für quantorenfreie Formeln, A, B gebrauchen wir folgende Bezeichnungen:

$A \subset B$ bedeute, daß A mit einer Teilkonjunktion von B äquivalent ist.
$A \sim B$ bedeute, daß A und B miteinander äquivalent sind.
Die Formel F_M heiße eine *Normalformel*, wenn folgende Bedingungen (I)—(IV) erfüllt sind:

(I) Zu jedem M-Doppelglied $D(x, y)$ gibt es $j \in M_0$ mit $D(x, y) \subset A_j(x, y, z)$ oder $j \in N_0$ mit $D(x, y) \sim B_j(x, y)$.

(II) Zu je zwei inäquivalenten M-Einzelgliedern $E(x)$ und $E^*(x)$ gibt es ein M-Doppelglied $D(x, y)$ mit $E(x) \wedge E^*(y) \subset D(x, y)$.

(III) Zu jedem M-Einzelglied $E(x)$ und jeder Zahl $i \in \{1, \ldots, r\}$ gibt es $j \in M_i$ mit $E(x) \subset F_{ij}(x, z)$ oder $j \in N_i$ mit $E(x) \sim G_{ij}(x)$.

(IV) Alle M-Grundformeln sind miteinander äquivalent.

Ist eine Formel F_M erfüllbar, so ist offenbar auch die Formel F erfüllbar. Umgekehrt gilt: Liegt ein Modell der Formel F vor, so erhält man eine Normalformel F_M durch dasjenige maximale Indexsystem

$$M = \langle M_0, \ldots, M_r, N_0, \ldots, N_r, P_1, \ldots, P_s, Q_1, \ldots, Q_s \rangle$$

der Formel F, für das alle Formeln

$$\exists x \exists y \exists z [A_j(x, y, z) \wedge U_n(x) \wedge U_n(y) \wedge U_n(z) \wedge x \neq y \wedge x \neq z \wedge y \neq z] \quad (j \in M_0),$$

$$\exists x \exists y [B_j(x, y) \wedge U_n(x) \wedge U_n(y) \wedge x \neq y] \quad (j \in N_0),$$

$$\exists x \exists y [F_{ij}(x, y) \wedge U_n(x) \wedge U_n(y) \wedge x \neq y] \quad (i = 1, \ldots, r;\ j \in M_i),$$

$$\exists x [G_{ij}(x) \wedge U_n(x)] \quad (i = 1, \ldots, r;\ j \in N_i),$$

$$\exists x [H_{ij}(x) \wedge U_n(x)] \quad (i = 1, \ldots, s;\ j \in P_i)$$

und K_{ij} $(i=1, \ldots, s;\ j \in Q_i)$ in dem betreffenden Modell erfüllt sind. Hiermit ergibt sich:

Satz 1. Die Formel F ist genau dann erfüllbar, wenn es ein Indexsystem M von F gibt, so daß F_M eine erfüllbare Normalformel ist.

Es gibt nur endlich viele Indexsysteme von F, und man kann für jedes Indexsystem M entscheiden, ob F_M eine Normalformel ist. Es kommt also nur darauf an, von einer Normalformel zu entscheiden, ob sie erfüllbar ist.

Im folgenden sei (4) eine Normalformel F_M. Ein M-Einzelglied $E(x)$ heiße *regulär*, wenn es ein M-Doppelglied $D(x, y)$ mit

$$E(x) \wedge E(y) \subset D(x, y)$$

gibt. Jedes andere M-Einzelglied heiße *singulär*.

1. Fall. Jedes M-Einzelglied sei regulär.

In diesem Fall sind die Eigenschaften (I)—(IV) der Normalformel F_M bereits hinreichend für die Erfüllbarkeit der Formel F_M. Es ist leicht zu erkennen, daß die Formel F_M in diesem Fall im Unendlichen erfüllbar ist. Man kann aber auch ähnlich wie in der Prädikatenlogik ohne Identität beweisen, daß die Formel F_M dann ebenfalls in einem geeigneten endlichen Bereich erfüllbar ist.

2. Fall. Mindestens ein M-Einzelglied sei singulär.

Wir wählen dann eine maximale Folge

$$R_1(x), \ldots, R_\varrho(x) \quad (\varrho \geqq 0)$$

von paarweise inäquivalenten regulären M-Einzelgliedern und eine maximale Folge

$$S_1(x), \ldots, S_\sigma(x) \quad (\sigma \geqq 1)$$

von paarweise inäquivalenten singulären M-Einzelgliedern aus.

Es läßt sich entscheiden, ob die Formel F_M in einem Bereich von höchstens $n+\sigma+1$ Elementen erfüllbar ist. Wir setzen im folgenden voraus, daß sie in keinem Bereich von weniger als $n+\sigma+2$ Elementen erfüllbar ist. Sie ist dann unerfüllbar, falls jedes M-Einzelglied singulär ist. Wir nehmen nun an, daß es mindestens ein reguläres M-Einzelglied gibt, also $\varrho \geqq 1$ ist. Dann definieren wir folgende Formeln:

$$R(x) :\equiv \bigwedge_{i=1}^{\varrho} R_i(x), \quad S :\equiv \bigwedge_{u=1}^{\sigma} S_u(a_{n+u})$$

$$A'(x, y, z) :\equiv \bigvee_{j \in M_0} A_j(x, y, z) \wedge R(x) \wedge R(y) \wedge R(z) \wedge S$$

$$B'(x, y) :\equiv [\bigvee_{j \in M_0} \bigvee_{u=1}^{\sigma} A_j(x, y, a_{n+u}) \bigvee_{j \in N_0} B_j(x, y)] \wedge R(x) \wedge R(y) \wedge S$$

$$\left.\begin{aligned}
F'_u(x, z) &:\equiv \bigvee_{j \in M_0} A_j(a_{n+u}, x, z) \wedge R(x) \wedge R(z) \wedge S \\
G'_u(x) &:\equiv [\bigvee_{j \in M_0} \bigvee_{\substack{v=1 \\ (v \neq u)}}^{\sigma} A_j(a_{n+u}, x, a_{n+v}) \bigvee_{j \in N_0} B_j(a_{n+u}, x)] \wedge R(x) \wedge S \\
F'_{\sigma+u}(x, z) &:\equiv \bigvee_{j \in M_0} A_j(x, a_{n+u}, z) \wedge R(x) \wedge R(z) \wedge S \\
G'_{\sigma+u}(x) &:\equiv [\bigvee_{j \in M_0} \bigvee_{\substack{v=1 \\ (v \neq u)}}^{\sigma} A_j(x, a_{n+u}, a_{n+v}) \bigvee_{j \in N_0} B_j(x, a_{n+u})] \wedge R(x) \wedge S
\end{aligned}\right\} (u = 1, \ldots, \sigma)$$

$$\left.\begin{aligned}
F'_{2\sigma+i}(x, z) &:\equiv \bigvee_{j \in M_i} F_{ij}(x, z) \wedge R(x) \wedge R(z) \wedge S \\
G'_{2\sigma+i}(x) &:\equiv [\bigvee_{j \in M_i} \bigvee_{u=1}^{\sigma} F_{ij}(x, a_{n+u}) \bigvee_{j \in N_i} G_{ij}(x)] \wedge R(x) \wedge S
\end{aligned}\right\} \quad (i = 1, \ldots, r)$$

Wir setzen $r' := 2\sigma + r$ und $s' := \binom{\sigma}{2} + r \cdot \sigma + s$. $H'_i(z)$ $(i = 1, \ldots, s')$ seien der Reihe nach folgende Formeln:

$$\bigvee_{j \in M_0} A_j(a_{n+u}, a_{n+v}, z) \wedge R(z) \wedge S \quad (u, v = 1, \ldots, \sigma;\ u \neq v)$$

$$\bigvee_{j \in M_i} F_{ij}(a_{n+u}, z) \wedge R(z) \wedge S \quad (i = 1, \ldots, r;\ u = 1, \ldots, \sigma)$$

$$\bigvee_{j \in P_i} H_{ij}(z) \wedge R(z) \wedge S \quad (i = 1, \ldots, s).$$

K_i' $(i=1, \ldots, s')$ seien der Reihe nach folgende Formeln:

$$[\bigvee_{j\in M_0} \bigvee_{\substack{w=1\\(u\neq w\neq v)}}^{\sigma} A_j(a_{n+u}, a_{n+v}, a_{n+w}) \bigvee_{j\in N_0} B_j(a_{n+u}, a_{n+v})]\wedge S \quad (u, v = 1, \ldots, \sigma;\ u\neq v)$$

$$[\bigvee_{j\in M_i} \bigvee_{\substack{v=1\\(v\neq u)}}^{\sigma} F_{ij}(a_{n+u}, a_{n+v}) \bigvee_{j\in N_i} G_{ij}(a_{n+u})]\wedge S \quad (i = 1, \ldots, r;\ u = 1, \ldots, \sigma)$$

$$[\bigvee_{j\in P_i} \bigvee_{u=1}^{\sigma} H_{ij}(a_{n+u}) \bigvee_{j\in Q_i} K_{ij}]\wedge S \quad (i = 1, \ldots, s).$$

Mit F' bezeichnen wir dann die Formel

$$\left\{\begin{array}{l} \forall x\, \forall y\{U_{n+\sigma}(x)\wedge U_{n+\sigma}(y)\wedge x\neq y\rightarrow \exists z[A'(x, y, z)\wedge U_{n+\sigma}(z)\wedge x\neq z\wedge y\neq z]\vee B'(x, y)\} \\ \bigwedge\limits_{i=1}^{r'} \forall x\{U_{n+\sigma}(x)\rightarrow \exists z[F_i(x, z)\wedge U_{n+\sigma}(z)\wedge x\neq z]\vee G_i'(x)\} \\ \bigwedge\limits_{i=1}^{s'} \{\exists z[H_i'(z)\wedge U_{n+\sigma}(z)]\vee K_i'\} \bigwedge\limits_{i=1}^{\varrho} \exists x[R_i(x)\wedge U_{n+\sigma}(x)]\wedge S\wedge C_{n+\sigma}. \end{array}\right. \tag{5}$$

Satz 2. Ist F_M eine Normalformel, die genau σ paarweise inäquivalente singuläre M-Einzelglieder und mindestens ein reguläres M-Einzelglied hat und in keinem Bereich von weniger als $n+\sigma+2$ Elementen erfüllbar ist, so ist F_M genau dann erfüllbar, wenn die Formel F' erfüllbar ist.

Falls eine Normalformel F_M nicht den Voraussetzungen des Satzes 2 genügt, läßt sich in einfacher Weise entscheiden, ob F_M erfüllbar ist. Andernfalls wird durch Satz 2 ein Reduktionsverfahren gegeben, das der Normalformel F_M eine „Reduzierte“ F' zuordnet, die jedoch länger als F_M ist. Es ist daher problematisch, ob das Reduktionsverfahren abbricht.

Falls das angegebene Reduktionsverfahren für jede Formel (1) abbricht, ist jede derartige Formel, wenn sie überhaupt erfüllbar ist, bereits im Endlichen erfüllbar. In diesem Fall ist die betrachtete Formelklasse entscheidbar.

Falls das Reduktionsverfahren für eine Formel (1) nicht abbricht, ist diese Formel nur im Unendlichen erfüllbar. Es ist jedoch problematisch, ob sich für jede Formel der betrachteten Formelklasse entweder das Abbrechen des Reduktionsverfahrens nachweisen oder effektiv eine unendliche Reduktionskette angeben läßt.

Für gewisse Teilklassen K und Sy der pränexen Formelklassen $\exists^m \forall^2 \exists^n$ der Prädikatenlogik mit Identität konnte M. Wirsing [7] beweisen, daß das hier angegebene Reduktionsverfahren jeweils nach endlich vielen Schritten abbricht, so daß jede erfüllbare Formel der betreffenden Klassen bereits im Endlichen erfüllbar ist.

MATHEMATISCHES INSTITUT
DER UNIVERSITÄT MÜNCHEN
D—8 MÜNCHEN
THERESIENSTR. 39

Literatur

[1] Gödel, K., Ein Spezialfall des Entscheidungsproblems der theoretischen Logik, *Ergebn. math. Kolloquium* 2, 1932, p. 27.
[2] Gödel, K., Zum Entscheidungsproblem des logischen Funktionenkalküls, *Monatsh. Math. Phys.*, v. 40, 1933, pp. 433—443.
[3] Kalmár, L., Über die Erfüllbarkeit derjenigen Zählausdrücke, welche in der Normalform zwei benachbarte Allzeichen enthalten, *Math. Ann.*, v. 108, 1933, pp. 466—484.
[4] Schütte, K., Untersuchungen zum Entscheidungsproblem der mathematischen Logik, *Math. Ann.* v. 109, 1934, pp. 572—603.
[5] Schütte, K., Über die Erfüllbarkeit einer Klasse von logischen Formeln, *Math. Ann.*, v. 110, 1934, pp. 161—194.
[6] Kahr, A. S., E. F. Moore, and H. Wang, Entscheidungsproblem reduced to the AEA Case, *Proc. Nat. Acad. Sci. U.S.A.*, v. 48, 1962, pp. 365—377.
[7] Wirsing, M., *Das Entscheidungsproblem der Prädikatenlogik 1. Stufe mit Identität und Funktionszeichen in Herbrandformeln*, Dissertation, München, *1976*.

(Eingegangen am 22. März 1978)

IN MEMORIAM LÁSZLÓ KALMÁR
INDEX — TARTALOM

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Gécseg Ferenc
A kézirat a nyomdába érkezett: 1978. szeptember hó
Megjelenés: 1978. december hó
Példányszám: 1025. Terjedelem: 13, (A/5) ív
Készült monószedéssel, íves magasnyomással
az MSZ 5601 és az MSZ 5602—55 szabvány szerint
78-4411 — Szegedi Nyomda — F. v.: Dobó József igazgató