CENTRUM OBLICZENIOWE
POLSKIEJ AKADEMII NAUK

ANTONI W. MAZURKIEWICZ

CLOSED PROGRAMMING SYSTEMS

WARSZAWA 1972
PAŃSTWOWE WYDAWNICTWO NAUKOWE

CENTRUM OBLICZENIOWE
POLSKIEJ AKADEMII NAUK

ANTONI W. MAZURKIEWICZ

CLOSED PROGRAMMING SYSTEMS

WARSZAWA 1972
PAŃSTWOWE WYDAWNICTWO NAUKOWE

The paper deals with equivalence of programs and proving their properties. A notion of closed programming system is introduced and some properties of this notion are considered. An algorithm for equivalent transformations or programs is given.

Praca dotyczy równoważności programów i dowodzenia ich własności. Wprowadzone jest pojęcie zupełnego systemu programowania i zbadane są jego własności. Podany jest algorytm równoważnościowej transformacji programów.

REDAKTOR WYDAWNICZY
CENTRUM OBLICZENIOWEGO PAN
Jan Lipski

Printed in Poland

Państwowe Wydawnictwo Naukowe
Oddział w Łodzi 1972

Wydanie I. Nakład 350+90 egz. Ark. wyd. 1,25. Ark. druk. 1 10/16.
Papier offset kl. III, 80 g, 70×100. Podpisano do druku 30 IX 1972 r.
Druk ukończono w październiku 1972 r. Zam. nr 369. D-10. Cena zł 10,—

Zakład Graficzny Wydawnictw Naukowych
Łódź, ul. Gdańska 162

1. INTRODUCTION

The growth of the computational power of a single statement is a general tendency in the development of programming languages. One of goals in designing new languages is to express in few statements the action caused by more statements in former languages. In particular, the more advanced language, the lower number of assignments statements in each program. How for can one go in this direction?

In the present paper we try to answer this question. We consider a programming system (abr. *ps*) with instructions of the form:

$$a : \underline{\text{if}}\ B\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b$$

where B is a condition, f is an operator, and a, b are labels (initial and terminal). Finite sets of instructions are programs in our *ps*. The restriction to the above form of instructions is not essential, e.g. each instruction of the type

$$a : \underline{\text{if}}\ B\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b\ \underline{\text{else}}\ g\ \underline{\text{and}}\ c$$

can be replaced in a program by the two following

$$a : \underline{\text{if}}\ B\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b$$

$$a : \underline{\text{if}} \sim B\ \underline{\text{then}}\ g\ \underline{\text{and}}\ c .$$

We are mostly interested in such *ps* where single instructions are of the possibly greatest computational efficiency. To this effect we introduce a notion of a closed programming system (abr. *cps*) with the property that for each (well-defined) program P in such a system there exists a semantically equivalent two-instruction program:

$$S_P = \left\{ \underline{\text{start}}:\ \underline{\text{if}}\ B_P\ \underline{\text{then}}\ f_P\ \underline{\text{and}}\ \underline{\text{stop}},\quad \underline{\text{start}}:\ \underline{\text{if}} \sim B_P\ \underline{\text{then}}\ \underline{\text{loop}} \right\}$$

where neither B_P nor f_P contain other instructions. It should be stressed that the reduced program S_P (said to be the canonical form of P) is written in the same system that P is. In other words, any closed programming system is reach enough to express properties of programs written in this system. Usually, such properties can not be described in the system itself but need some stronger meta--system. In fact, S_P describes the properties of P : for any data vector x , if $B_P(x)$ is satisfied, then S_P, hence also P ,stops and gives the result $f_P(x)$. In the opposite case, i.e. if $\sim B_P(x)$ is satisfied, the result of P is not defined.

Now, the following question arises: what properties should have a *ps* in order to be closed? How to reduce a given program in *cps* to its canonical form?

In the paper, after introducing some basis notions,a very simple *ps* is defined and conditions for such *ps* to be closed are formulated. Next, we show what rules of replacement can be used to transform programs into semantically equivalent canonical form (Theorem 1). In the rest of paper we define a set of derivation rules permitting to reduce programs into the canonical form, like theorems in a deductive systems can be derived from the axioms of this system. The basic result of this part is Theorem 2, on the completeness of the set of the derivation rules.

2. BASIC NOTIONS

Definition 1. A programming system A , considered in this paper, is defined by its language L_A and its semantics M_A. The language of A consists of:

(i) an alphabet Σ_A ;

(ii) a subset E_A of Σ_A^*, called the set of labels and containing three distinguished symbols <u>start</u>, <u>stop</u> and <u>loop</u>;

(iii) a subset C_A of Σ_A^*, called the set of conditions and containing a distinguished symbol <u>true</u>;

(iv) a subset F_A of Σ_A^*, called the set of operators and containing a distinguished symbol <u>empty</u>.

The set

$$I_A = (E_A - \{\underline{\text{stop}}, \underline{\text{loop}}\}) \times C_A \times F_A \times (E_A - \{\underline{\text{start}}\})$$

is called the set of instructions in A. For any $r = (a,B,f,b)$ in I_A we shall write

$$a: \underline{\text{if}}\ B\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b.$$

We shall use also an abbreviated notation, writing

$a: f\ \underline{\text{and}}\ b$ instead of $a: \underline{\text{if}}\ \underline{\text{true}}\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b$

$a: \underline{\text{if}}\ B\ \underline{\text{then}}\ b$ instead of $a: \underline{\text{if}}\ B\ \underline{\text{then}}\ \underline{\text{empty}}\ \underline{\text{and}}\ b$

$a:b$ instead of $a: \underline{\text{if}}\ \underline{\text{true}}\ \underline{\text{then}}\ \underline{\text{empty}}\ \underline{\text{and}}\ b$.

The label a is said to be the *initial label* of r, the label b is said to be the *terminal label* of r. Two instructions with the same initial labels and the same terminal labels are called *similar*. Every instruction such that its initial label is identical with its terminal label is called *reflexive*. A finite subset P of the set I_A is called a *program* in A. By $E(P)$ we denote the set of all labels occurring in the instructions of the program P. A label a is said to be *blind* in P, if $\underline{\text{stop}} \neq a \neq \underline{\text{loop}}$ and there is no instruction in P where a is initial. A label a is said to be *inaccessible* in P, if $a \neq$ start and there is no instruction in P where a is terminal. Then, any label not in $E(P)$ is blind as well as inaccessible in P.

Every program P can be represented by a labelled graph Γ_P, having $E(P)$ as the set of vertices and P as the set of directed arcs. To each instruction $a: \underline{\text{if}}\ B\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b$ corresponds an arc, starting in a, entering into b, and labelled with (B,f). Such a graph is called the *flow-diagram* of P.

The semantics M_A of ps A is a system consisting of:

(i) a nonempty set X_A (of states), called the domain of interpretation,

(ii) a mapping $\varphi_A: F_A \times X_A \times X_A \to \{0,1\}$, called the interpretation of operators,

(iii) a mapping $\psi_A: C_A \times X_A \to \{0,1\}$, called the interpretation of conditions.

We shall assume that for every f in F_A and x,y,z in X_A,

$$\varphi_A(\underline{\text{empty}},x,y) = 1 \text{ iff } x = y$$
$$\psi_A(\underline{\text{true}},x) = 1$$
$$\varphi_A(f,x,y) = 1 \text{ and } \varphi_A(f,x,z) = 1 \text{ implies } y = z.$$

Given a semantics M_A, we write Qx instead of $\psi_A(Q,x)$ and $y=fx$ instead of $\varphi_A(f,x,y)=1$.

We shall write $Q_1 \supset Q_2$(resp. $Q_1 \equiv Q_2$), if for all x in X_A, $Q_1x=1$ implies $Q_2x=1$ (resp. $Q_1x=1$ iff $Q_2x=1$). We shall write $Q \supset (f_1 = f_2)$ if for all x, y, z in X_A such that $y=f_1x$, $z=f_2x$ $Qx=1$ implies $y=z$. The set $S_A = E_A \times X_A$ is called the set of *situations* in A. If P_A is a program in A, then the subset Sp_A of S_A, $Sp_A = E(P_A) \times X_A$ is called the set of situations in P_A. For every instruction r and any situations s_1, s_2, we write

$$r: s_1 \rightarrow s_2$$

if r is an instruction: a: <u>if</u> Q <u>then</u> f <u>and</u> b, $s_1=(a,x)$, $s_2=(b,y)$, and $Qx=1$, $y=fx$. For any program P_A and situations s_1, s_2, we write

$$P_A: s_1 \rightarrow s_2$$

if there exists in P_A an instruction r such that r: $s_1 \rightarrow s_2$. A sequence of situations:

$$(s_0, s_1, \ldots, s_n), \quad n>0,$$

is called a *computation* in P_A beginning with s_0 and ending with s_n, if P_A: $s_{i-1} \rightarrow s_i$ for $1 < i < n$. We write P_A: $s_1 \Rightarrow s_2$ (or simply: $s_1 \Rightarrow s_2$, if P_A is known), if there exists a computation in P_A beginning with s_1 and ending with s_2.
We write

$$\mathrm{Comp}_{P_A}(x,y)$$

if P_A: (<u>start</u>, x) $\Rightarrow$ (<u>stop</u>, y).

P r o p o s i t i o n 1. For any program P_A, if $s_1 \Rightarrow s_2$, and $s_2 \Rightarrow s_3$, then $s_1 \Rightarrow s_3$.

Proof is obvious.

Let P_A be a program in A, and let

$$P_A = \{a_i: \underline{\text{if}}\ Q_i\ \underline{\text{then}}\ f_i\ \underline{\text{and}}\ b_i \mid i = 1,2,\ldots,N\}, \quad N>0.$$

We shall say that P_A is *consistent*, if for each i, j, $i \neq j$, $1 < i, j < N$, and every x in X_A,

$$a_i = a_j \text{ implies } Q_i x = 0 \text{ or } Q_j x = 0.$$

We shall say that P_A is *complete*, if for each i, $1 \leq i \leq N$, and every x in X_A, there exists j, $1 \leq j \leq N$, such that

$$a_i = a_j \text{ and } Q_j x = 1.$$

We shall say that program P_A is *executable*, if for each i, $1 \leq i \leq N$, and every x in X_A, there exists y in X_A such that

$$Q_i x = 1 \text{ implies } y = f_i x.$$

A program P_A is said to be *well-defined*, if it is consistent, complete, and executable. Note that the empty program is well-defined.

Let s be in S_A, let P_A be a program in A. We shall write stop (s), if $s = (\underline{\text{stop}}, x)$ for some x in X_A ; we shall write loop(s), if there exists no such s' in S_A that $P_A : s \twoheadrightarrow s'$ and stop(s').

P r o p o s i t i o n 2. For any well-defined program P_A:

(i) if $s_1 \Rightarrow s_2$ and $s_1 \Rightarrow s_3$, then either $s_2 \Rightarrow s_3$, or $s_2 = s_3$, or $s_3 \Rightarrow s_2$.

(ii) if $s_1 \Rightarrow s_2$, $s_1 \Rightarrow s_3$, stop(s_2), and stop(s_3), then $s_2 = s_3$.

(iii) stop(s) implies loop(s) does not hold.

(iv) if loop(s_2) and $s_1 \Rightarrow s_2$, then loop(s_1).

(v) Let T be a subset of $S_A - \{s \mid \text{stop}(s)\}$. If for all s_1 in T, $s_1 \Rightarrow s_2$ implies s_2 is in T, then loop(s) for all s in T.

(vi) If $P_A : s \twoheadrightarrow s_1$ and $P_A : s \twoheadrightarrow s_2$, then $s_1 = s_2$.

(vii) If $\text{Comp}_{P_A}(x,y)$ and $\text{Comp}_{P_A}(x,z)$, then $y = z$.

Definition 2. We say that a programming system A is *closed*, if the following conditions are satisfied:

1. There are defined in Q_A operations: $\sim$ (unary), $\vee$ (binary), $\wedge$ (binary), such taht for all x in X_A and Q, Q_1, Q_2 in Q_A:

$$(\sim Q)x = 1 \quad \text{iff} \quad Qx = 0,$$
$$(Q_1 \vee Q_2)x = 1 \quad \text{iff} \quad Q_1 x = 1 \quad \text{or} \quad Q_2 x = 1,$$
$$(Q_1 \wedge Q_2)x = 1 \quad \text{iff} \quad Q_1 x = 1 \quad \text{and} \quad Q_2 x = 1.$$

We shall assume $\sim$ stronger than $\wedge$, $\wedge$ stronger than $\vee$. We shall write false instead $\sim$ true. Note that the pair of instructions:

$$a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b\ ;$$

$$a: \underline{\text{if}} \sim Q\ \underline{\text{then}}\ g\ \underline{\text{and}}\ c$$

is written usually as

$$a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b\ \underline{\text{else}}\ g\ \underline{\text{and}}\ c.$$

2. There is defined in F_A an operation o (binary) such that for all x in X_A, and f_1, f_2 in F_A

$y = (f_1 \circ f_2)x$ iff there exists z in X_A such that $z = f_2 x$ and $y = f_1 z$.

3. There are defined mappings $\alpha: Q_A \times F_A \times F_A \to F_A$, $\beta: Q_A \times F_A \to Q_A$, $\gamma: Q_A \times F_A \to F_A$, such that for any Q in Q_A, f, g in F_A, and for all x, y in X_A:

$y = \alpha(Q,f,g)x$ iff either $Qx = 1$ and $y = fx$, or $Qx = 0$ and $y = gx$;

$\beta(Q,f)x = 1$ iff there exists z in X_A such that $Qz = 1$ and $z = fx$;

$y = \gamma(Q,f)x$ iff there exists a sequence $(x_0, x_1, \ldots, x_n)$, $n \geqslant 0$, such that $x_0 = x$, $Qx_{i-1} = 1$, $x_i = fx_{i-1}$, $1 \leqslant i \leqslant n$, $Qx_n = 0$, $x_n = y$, x_n is in X_A.

Instead of $f \circ g$, $\alpha(Q,f,g)$, $\beta(Q,f)$, $\gamma(Q,f)$ we shall write fg, $Q|f|g$, Qf, $Q*f$, respectively. We shall assume o to be stronger than $*$, and β stronger than $\sim$, $\wedge$, $\vee$. It should be noted the difference between $Q_1 \supset Q_2$ and, for instance, $Q_1 \vee Q_2$; the first denotes a binary relation in the set Q_A, while the second denotes an element of Q_A. The same note concernes $Q_1 \equiv Q_2$.

If A is closed, then we can define, for each non negative integer k, every f in F_A, and any $Q_1, Q_2, \ldots, Q_k$, in Q_A, the following operators and conditions:

$$f^0 \equiv \underline{\text{empty}}, \qquad f^{k+1} \equiv ff^k,$$

$$\bigvee_{i=1}^{0} Q_i \equiv \underline{\text{false}}, \quad \bigvee_{i=1}^{k+1} Q_i \equiv \left(\bigvee_{i=1}^{k} Q_i\right) \vee Q_{k+1},$$

$$\bigwedge_{i=1}^{0} Q_i \equiv \underline{\text{true}}, \quad \bigwedge_{i=1}^{k+1} Q \equiv \left(\bigwedge_{i=1}^{k} Q_i\right) \wedge Q_{k+1}.$$

We shall interprete fg^k as $f(g^k)$.

P r o p o s i t i o n 3. For any closed programming system A:

(i) $\bigvee_{i=1}^{k}(Q_i f) \equiv \left(\bigvee_{i=1}^{k} Q_i\right) f$, $\bigwedge_{i=1}^{k}(Q_i f) \equiv \left(\bigwedge_{i=1}^{k} Q_i\right) f$;

(ii) $(\underline{\text{true}}\ f)x = 1$ iff there exists z in X_A such that $z = fx$;

(iii) $f_1(f_2 f_3) \equiv (f_1 f_2) f_3$, $(Qf_1)f_2 \equiv Q(f_1 f_2)$;

(iv) Let P_A be a program in A and let $P_A = P_0 \cup P_1$, P_0 contains no instructions with a as the initial label, $P_1 = \{a:$ if Q_i then f_i and $b_i \mid i = 1,2,\ldots,N\}$, $N>0$.

a. $\bigvee_{i=1}^{N} Q_i \equiv \underline{\text{true}}$ iff P_A is complete,

b. $Q_i \wedge Q_j \equiv \underline{\text{false}}$ for $i \neq j$, $i < j$, $i < N$, iff P_A is consistent,

c. if P_A is complete, then:

$\bigvee_{i=1}^{N} \underline{\text{true}}\ f_i \equiv \underline{\text{true}}$ iff P_A is executable.

3. PROGRAM TRANSFORMATIONS

In this section we shall consider an arbitrary but fixed closed programming system A; we shall use an abbreviated notation, writing P, $E(P)$, X,... instead P_A, $E(P_A)$, X_A,... and similarly for other symbols. In whole this section programs are assumed to be well-defined. The main result of this section concernes program transformations preserving the relation Comp. Our purpose is to prove that every well-defined program P in a closed programming system can be transformed into the well-defined, two-instruction program S containing no labels but start, stop, and loop, and such that $\text{Comp}_P = \text{Comp}_S$. After such a transformation, the semantic analysis of P becomes quite simple.

Definition 3. Program P is said to be *strongly equivalent* to a program R (or, simply, *equivalent*), if for every x, y in X

$$\text{Comp}_P(x,y) \text{ if and only if } \text{Comp}_R(x,y).$$

L e m m a 1. (On the elimination of blind labels). Let P be a program. If b is blind in P, then $P_1 = P \cup \{b: \underline{\text{loop}}\}$ is a program equivalent to P.

Proof is obvious.

L e m m a 2. (On the elimination of inaccessible labels). Let P be a program. If a is inaccessible in P and start is not blind in P, then $P_1 = P \cup \{\underline{\text{start}}: \underline{\text{if}}\ \underline{\text{false}}\ \underline{\text{then}}\ a\}$ is is a program equivalent to P.

Proof is obvious.

L e m m a 3. (On the reduction of similar instructions). A program

$$P_1 = P_0 \cup \{a: \underline{\text{if}}\ Q_1\ \underline{\text{then}}\ f_1\ \underline{\text{and}}\ b,\ a: \underline{\text{if}}\ Q_2\ \underline{\text{then}}\ f_2\ \underline{\text{and}}\ b\}.$$

is equivalent to the program

$$P_2 = P_0 \cup \{a: \underline{\text{if}}\ Q_1 \vee Q_2\ \underline{\text{then}}\ Q_1 | f_1 | f_2\ \underline{\text{and}}\ b\}.$$

Proof follows directly from the definition of the operator $(Q_1|f_1|f_2)$.

L e m m a 4. (On the elimination of reflexive instructions). Let $P_1 = P_0 \cup \{a_0: \underline{\text{if}}\ Q_i\ \underline{\text{then}}\ f_i\ \underline{\text{and}}\ a_i \mid i = 0,1,\ldots,M\}$ $M>0$, be such a program that:

(i) P_0 does not contain any instruction with the initial label a_0;

(ii) $a_0 \neq a_i$ for $1 < i < M$;

Then

$$P_2 = P_0 \cup \{a_0: \underline{\text{if}} \sim \underline{\text{true}}\ (Q_0 * f_0)\ \underline{\text{then}}\ \underline{\text{loop}}\} \cup$$
$$\{a_0: \underline{\text{if}}\ Q_i(Q_0 * f_0)\ \underline{\text{then}}\ f_i(Q_0 * f_0)\ \underline{\text{and}}\ a_i |$$
$$i = 1,2,\ldots,M\}$$

is a program equivalent to P_1.

Proof. At first, check P_2 is well-defined. The program P_2 is complete; indeed, $\bigvee_{i=1}^{M} Q_i(Q_0 * f_0) \equiv \bigvee_{i=0}^{M} Q_i(Q_0 * f_0)$ because $Q_0(Q_0 * f_0) \equiv \underline{\text{false}}$, and by Proposition 3(i) $\bigvee_{i=0}^{M} Q_i(Q_0 * f_0) \equiv (\bigvee_{i=1}^{M} Q_i)(Q_0 * f_0) \supset \underline{\text{true}}(Q_0 * f_0)$. Hence,

$$\bigvee_{i=1}^{M} Q_i(Q_0 * f_0) \vee \sim \underline{\text{true}}(Q_0 * f_0) \equiv \underline{\text{true}},$$

and, once more by Proposition 3(iv)(a), P_2 is complete.

The program P_2 is consistent. In fact, by proposition 3(i) $(Q_i(Q_0 * f_0)) \wedge (Q_j(Q_0 * f_0)) = (Q_i \wedge Q_j)(Q_0 * f_0) = \underline{\text{false}}$ by assumption, for $i \neq j$, $1 \leqslant i$, $j \leqslant M$; $(Q_i(Q_0 * f_0)) \wedge \sim \underline{\text{true}}(Q_0 * f_0)) \supset \underline{\text{true}}\ (Q_0 * f_0) \wedge \sim \underline{\text{true}}\ (Q_0 * f_0) \equiv \underline{\text{false}}$. Finally, P_2 is executable: there exists always y in Y such that $y = \underline{\text{empty}}\ x$ (namely, x); if $Q_i(Q_0 * f_0)x = 1$, then there exists z in X such that $z = (Q_0 * f_0)x$ and $Q_i z = 1$, hence, by assumption, there exists y in X such that $y = f_i z$, what proves the executability of P_2. Thus, P_2 is well-defined.

Now, assume $\text{Comp}_{P_1}(x, y)$. It means that there exists a computation in P_1:

$$(s_0, s_1, \ldots, s_n), \quad n \geqslant 1,$$

such that $s_0 = (\underline{\text{start}}, x)$, $s_n = (\underline{\text{stop}}, y)$. Let us consider the subsequence $(s_{j_0}, s_{j_1}, \ldots, s_{j_m})$, $m \geqslant 1$, of this computation, defined as follows:

(i) $s_{j_0} = s_0 = (\underline{\text{start}}, x)$

(ii) $s_{j_m} = s_n = (\underline{\text{stop}}, y)$

(iii) Let $s_{j_k} = (a, z)$. Then, if $a \neq a_0$ we put $s_{j_{k+1}} = s_{j_k+1}$; if $a = a_0$ we put $s_{j_{k+1}} = s_{j_k+p+1}$, where p is the smallest non-negative integer such that $s_{j_k+p+1} = (b, t)$ implies $b \neq a_0$. Such an integer always exists, because $\underline{\text{stop}} \neq a_0$.

We claim that for each k, $1 \leqslant k \leqslant m$,

$$P_2\colon s_{j_k} \to s_{j_{k+1}} \tag{*}$$

Let $s_{j_k} = (a, z)$. If $a \neq a_0$, then $P_0\colon s_{j_k} \to s_{j_{k+1}}$ and by definition of the subsequence and the program P_2 we obtain (*).

Assume $a = a_0$ and consider the following sequence:

$$(s_{j_k}, s_{j_{k+1}}, \dots, s_{j_{k+p}}, s_{j_{k+p+1}}).$$

This sequence is a computation in P_1; by definition of p, there is l, $1 < l < M$, and $z^{(q)}$ in X, $1 \le q \le p$, such that:

$$s_{j_k+q} = (a_0, z^{(q)}), \; z^{(q)} = f_0^q z, \; Q_0 f_0^{q-1} z = 1, \; 1 \le q \le p,$$

$$s_{j_k+p+1} = (a_l, z'), \; z' = f_l f^p z, \; Q_0 f_0^p z = 0, \; Q_l f_0^p z = 1,$$

that is, by Proposition 3(v) $f_0^p = Q_0 * f_0$, $Q_l(Q_0 * f_0)z = 1$, $z' = f_l(Q_0 * f_0)z$, what implies $(*)$. Hence, the considered subsequence is a computation in P_2 what proves $\mathrm{Comp}_{P_2}(x,y)$.

Assume now $\mathrm{Comp}_{P_2}(x,y)$, and let $(s_0, s_1, \dots, s_n)$, $n \geq 1$, be a computation in P_2 such that $s_0 = (\underline{\text{start}}, x)$, $s_n = (\underline{\text{stop}}, y)$. W shall prove that for all j, $0 \le j \le n-1$, $P_1: s_j \Rightarrow s_{j+1}$. Let $s_j = (a, z)$, $s_{j+1} = (b, t)$. If $a \neq a_0$, then $P_0: s_j \Rightarrow s_{j+1}$ what implies $P_1: s_j \rightarrow\Rightarrow s_{j+1}$. If $a = a_0$, then by the definition of P_1 there must be

$$Q_l(Q_0 * f_0)z = 1, \quad t = f_l(Q_0 * f_0)z.$$

Thus, there is such u in X that $u = (Q_0 * f_0)z$ and $t = f_l u$; hence, there exists an integer $p > 0$ and a sequence $z^{(1)}, z^{(2)}, \dots, z^{(p)}$ of elements of X, such that

$$z^{(q)} = f_0^q z, \; Q_0 f_0^{q-1} z = 1 \quad \text{for} \quad 1 \le q \le p,$$

and

$$u = z^{(p)}, \; Q_0 f_0^p z = 0, \; Q_l f_0^p z = 1.$$

Consider the sequence:

$$((a_0, z), (a_0, z^{(1)}), \dots, (a_0, z^{(p)}), (a_l, t)).$$

As it follows from the definition of P_1, this sequence is a computation in P_1, what yields $P_1: s_j \Rightarrow s_{j+1}$. By transitivity of $\rightarrow$ we obtain $P_1: s_0 \rightarrow s_n$, what completes the proof of Lemma 4.

L e m m a 5. (On the elimination of labels). Let $P_1 = P_0 \cup A \cup B$ be a program such that:

(i) $A = \{a_j : \underline{\text{if}}\ R_j\ \underline{\text{then}}\ f_j\ \underline{\text{and}}\ b \mid j = 1,2,\ldots,N\}$,

(ii) $B = \{b : \underline{\text{if}}\ Q_l\ \underline{\text{then}}\ g_l\ \underline{\text{and}}\ c_l \mid l = 1,2,\ldots,M\}$,

(iii) P_0 does not contain any instruction with initial or terminal label identical with b,

(iv) $a_j \neq b \neq c_l$ (There are no reflexive instructions in $A \cup B$), $1 < l < M$, $1 < j < N$,

(v) $N > 0$, $M > 0$ (b is neither blind nor inaccessible).

Then the program

$$P_2 = P_0 \cup \{a_j : \underline{\text{if}}\ R_j \wedge Q_l f_j\ \underline{\text{then}}\ g_l f_j\ \underline{\text{and}}\ c_l \mid j = 1,2,\ldots,N,\ l = 1,2,\ldots,M\}$$

is equivalent to P_1.

Proof. We shall prove only P_2 is well-defined; the rest of the proof, as similar to that of Lemma 4, will be omitted. To prove consistency, consider

$$(R_j \wedge Q_l f_j) \wedge (R_k \wedge Q_m f_k) \tag{1}$$

$1 < j,\ k < N,\ 1 < l,\ m < M,\ j \neq k$ or $l \neq m$. If $j \neq k$, then since P_1 is well-defined, $R_j \wedge R_k \equiv \underline{\text{false}}$, and $(1) \equiv \underline{\text{false}}$.

If $j = k$ and $l \neq m$, then (1) is equivalent to $R_j \wedge ((Q_l \wedge \wedge Q_m) f_j)$ by Proposition 3(i). On the other hand $Q_l \wedge Q_m \equiv \underline{\text{false}}$ by the assumption thus (1) is also $\underline{\text{false}}$.

To prove completeness, it suffices to show that

$$\bigvee_{j=1}^{N} \bigvee_{l=1}^{M} (R_j \wedge Q_l f_j) \equiv \bigvee_{j=1}^{N} R_j \tag{*}$$

In fact, by Proposition 3(i)

$$\bigvee_{j=1}^{N} \bigvee_{l=1}^{M} (R_j \wedge Q_l f_j) \equiv \left(\bigvee_{j=1}^{N} R_j \wedge \left(\bigvee_{l=1}^{M} Q_l \right) \cancel{\underline{\text{true}}}\ f_j \right)$$

and by assumed completeness of P_1, $\bigvee_{l=1}^{M} Q_l \equiv \underline{\text{true}}$ $(M > 0)$ hence * is equivalent to

$$\bigvee_{j=1}^{N} (R_j \wedge \underline{\text{true}}\ f_j)$$

But P_1 is well-defined, hence executable, thus $R_j \supset \underline{\text{true}}\ f_j$ for all j, $1 \leq j \leq N$, and thus $\bigvee_{j=1}^{N} (R_j \wedge \underline{\text{true}}\ f_j) = \bigvee_{j=1}^{N} R_j$.

To prove executability, observe that if $(R_j \wedge Q_i f_j)x = 1$, then $R_j x = 1$, hence by the assumption there exists y in X such that $y = f_j x$; since $Q_i f_j x = 1$, we have $Q_i y = 1$. Hence, by the assumption, there is u in X such that $u = g_i y$, i.e. $u = g_i f_j x$, what proves P_2 to be executable.

T h e o r e m 1. For any closed programming system A and any well-defined program P_A in A, there exists a condition Q_P in Q_A, and an operator f_P in F_A, such that P is equivalent to the program:

$$S_P = \{\underline{\text{start}}\text{: } \underline{\text{if}}\ Q_P\ \underline{\text{then}}\ f_P\ \underline{\text{and}}\ \underline{\text{stop}},$$

$$\underline{\text{start}}\text{: } \underline{\text{if}} \sim Q_P\ \underline{\text{then}}\ \underline{\text{loop}}\}.$$

S_P will be called in the sequel the *canonical form* of P.

Proof. Consider the set $E(P)$. If <u>start</u> is not in $E(P)$, then <u>start</u> is blind in P. Hence, by Lemma 1 we can replace P by its equivalent

$$P \cup \{\underline{\text{start}}\text{: } \underline{\text{loop}}\},$$

If <u>start</u> is in $E(P)$, but <u>stop</u> is not, then <u>stop</u> is inaccessible in P and by Lemma 2 we can replace P by its equivalent

$$P \cup \{\underline{\text{start}}\text{: } \underline{\text{if}}\ \underline{\text{false}}\ \underline{\text{then}}\ \underline{\text{stop}}\}.$$

Thus, we can assume that <u>start</u> and <u>stop</u> are in $E(P)$. Now, if <u>loop</u> is not in $E(P)$, then <u>loop</u> is inaccessible in P and by Lemma 2 we can replace P by its equivalent

$$P \cup \{\underline{\text{start}}\text{: } \underline{\text{if}}\ \underline{\text{false}}\ \underline{\text{then}}\ \underline{\text{loop}}\}.$$

Hence, we can assume that $E(P)$ contains <u>start</u>, <u>stop</u> and <u>loop</u>.

Let $G(P) = E(P) - \{\underline{\text{start}}, \underline{\text{stop}}, \underline{\text{loop}}\}$. We shall prove Theorem 1 by induction with respect to $\text{card}(G(P))$.

a. Assume $\text{card}(G(P)) = 0$. In this case, applying the result of Lemma 3 we obtain the following well-defined program, equivalent to P:

$$\{\underline{\text{start}}\text{: } \underline{\text{if}}\ Q_P\ \underline{\text{then}}\ f_P\ \underline{\text{and}}\ \underline{\text{stop}},$$

$$\underline{\text{start}}\text{: } \underline{\text{if}}\ R_P\ \underline{\text{then}}\ \underline{\text{loop}}\},$$

and, since this program is well-defined, R_P is $\sim Q_P$; in this case Theorem 1 is valid.

b. Suppose Theorem 1 is true for all programs P, such that $\mathrm{card}(G(P')) < n$, $n > 0$. We shall prove it for P such that $\mathrm{card}(G(P)) = n$, by transforming P into P', $\mathrm{card}(G(P')) = n - 1$. This transformation will be performed in four steps.

Step 1. As in Lemma 3, we reduce all similar instructions in P.

Step 2. As in Lemma 4, we eliminate all reflexive instructions in P; since there are no similar instructions in P, Lemma 4 can be applied.

Step 3. Since $\mathrm{card}\,(G(P)) > 0$, we can find a label in $G(P)$, say, b. If b is blind in P, then we apply Lemma 1; if b is inaccessible, we apply Lemma 2.

Step 4. Since b is neither blind nor inaccessible in P, and P contains no reflexive instructions, we can eliminate label b as in Lemma 5.

Every step listed above fransforms a program into its equivalent, preserving the property "to be well-defined". Hence, the result of this transformation is a program P', equivalent to P, and if P is well-defined, then so is P'. Since $E(P')$ contains all labels of $E(P)$ excluding b, $\mathrm{card}(G(P')) = n - 1$. Hence, the proof is completed by induction.

Corollary 1. Let P be a well-defined program in A, $S_P = \{$ <u>if</u> Q <u>then</u> f <u>and</u> <u>stop</u>, <u>if</u> $\sim Q$ <u>then</u> <u>loop</u>$\}$ be the canonical form of P. The following equivalence holds for all x in X_A:

$$Qx = 1 \text{ iff there is } y \text{ in } X_A \text{ such that } \mathrm{Comp}_P(x,y).$$

Proof. Since P is well defined, so is S_P. Hence S_P is executable; it means that if $Qx = 1$, then there exists y in X_A such that $y = fx$, i.e. that $\mathrm{Comp}_P(x,y)$. On the other hand, if $Qx = 0$, then $(\sim Q)x = 1$ and there exists no such y in X_A that $\mathrm{Comp}(x,y)$, what completes the proof.

4. DERIVATION RULES

In this section we suggest another approach: each program will be considered as a set of "axioms" (instructions) that describe the next state function of the program. Now what we want is to give a set of "derivation rules" that permit to produce new instructions (theorems) describing the transitive closure of the next state function. Such derivation rules are introduced in this section; the main result of this section is a theorem to the effect that the introduced derivation rules are reach enough to derive the canonical form for every well-defined program.

Definition 4. Let A be a closed programming system, and let P_A be a program in A. The set $\mathrm{Cons}(P_A)$ is the smallest subset of I_A satisfying the following conditions. For arbitrary Q, Q_1, Q_2 in Q_A, f, f_1, f_2 in F_A, and a, b, c in E_A (writing $P_A \vdash r$ instead of r is in $\mathrm{Cons}(P_A)$):

1. If

$$P_A \vdash a: \underline{\text{if}}\ Q_1\ \underline{\text{then}}\ f_1\ \underline{\text{and}}\ b,\ Q_2 \supset Q_1,\ Q_2 \supset (f_1 = f_2)$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q_2\ \underline{\text{then}}\ f_2\ \underline{\text{and}}\ b;$$

2. If

$$P_A \vdash a: \underline{\text{if}}\ Q_1\ \underline{\text{then}}\ f_1\ \underline{\text{and}}\ b,\ P_A \vdash b: \underline{\text{if}}\ Q_2\ \underline{\text{then}}\ f_2\ \underline{\text{and}}\ c,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q_1 \wedge Q_2 f_1\ \underline{\text{then}}\ f_2 f_1\ \underline{\text{and}}\ c;$$

3. If

$$P_A \vdash a: \underline{\text{if}}\ Q_1\ \underline{\text{then}}\ f_1\ \underline{\text{and}}\ b,\ P_A \vdash a: \underline{\text{if}}\ Q_2\ \underline{\text{then}}\ f_2\ \underline{\text{and}}\ b,\ Q_1 \wedge Q_2 = \underline{\text{false}},$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q_1 \vee Q_2\ \underline{\text{then}}\ Q_1 | f_1 | f_2\ \underline{\text{and}}\ b;$$

4. If

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ a,\ Q \supset Qf,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ \underline{\text{loop}};$$

5. If

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ a,$$

then

$$P_A \vdash a: \underline{\text{if}}\ \underline{\text{true}}\ (Q * f)\ \underline{\text{then}}\ Q * f\ \underline{\text{and}}\ a;$$

6. If b is blind in P_A, then

$$P_A \vdash b: \underline{\text{loop}}$$

7. If a is inaccesible in P_A, then

$$P_A \vdash \underline{\text{start}}: \underline{\text{if}}\ \underline{\text{false}}\ \underline{\text{then}}\ a.$$

The above definition can be treated as a set of rules, by means of which we can derive some instructions from anothers. In fact, it follows from the definition that $P_A \vdash r$ if and only if there exists a derivation of r from P_A, i.e. a sequence

$$(r_0, r_1, \dots, r_n), \quad n \geq 0,$$

where $r = r_n$ and where each r_i satisfies one of the following conditions:

(1) r_i is in P_A or r_i can be derived by means of rules 6 or 7;

(2) there exists r_j with $j < i$ that derives r_i by means of rules 1 or 4 or 5;

(3) there exists r_j, r_k with $j, k < i$ that derive r_i by means of rules 2 or 3.

P r o p o s i t i o n 4. For every well-defined program P_A in a closed programming system A, and every Q, S in Q_A, f, g in F_A, a, b, c in E_A:

8. For every integer $m \geq 1$, if $P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ a$, then

$$P_A \vdash a: \underline{\text{if}}\ \bigwedge_{k=1}^{m} Qf^{k-1}\ \underline{\text{then}}\ f^m\ \underline{\text{and}}\ a;$$

9. For every integer $m \geq 1$, if

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ a,$$

$$P_A \vdash a: \underline{\text{if}}\ S\ \underline{\text{then}}\ g\ \underline{\text{and}}\ b,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Sf^m \wedge \bigwedge_{k=1}^{m} Qf^{k-1}\ \underline{\text{then}}\ gf^m\ \underline{\text{and}}\ b;$$

10. If

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ b, \quad P_A \vdash b: f\ \underline{\text{and}}\ c,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ c;$$

11. If

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b, \quad P_A \vdash b: c,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ c;$$

12. If

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b, \quad P_A \vdash a: \underline{\text{if}}\ S\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q \vee S\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b;$$

13. If

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q \wedge S\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b;$$

14. If

$$P_A \vdash a: \underline{\text{if}}\ Q \vee S\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b;$$

15. If

$$P_A \vdash a: f\ \underline{\text{and}}\ b,$$

then

$$P_A \vdash a: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ b.$$

L e m m a 6. For any well-defined program P_A in a closed programming system A, if R is a canonical form of P_A:

$$R = \{\underline{\text{start}}: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ \underline{\text{stop}},\ \underline{\text{start}}: \text{if}\ \sim Q\ \underline{\text{then}}\ \underline{\text{loop}}\},$$

then

$$P_A \vdash \underline{\text{start}}: \underline{\text{if}}\ Q\ \underline{\text{then}}\ f\ \underline{\text{and}}\ \underline{\text{stop}},$$

$$P_A \vdash \underline{\text{start}}: \underline{\text{if}}\ \sim Q\ \underline{\text{then}}\ \underline{\text{loop}}.$$

Proof. It suffices to show that each step in reducing a program to its equivalent, as in Lemmas 1, 2, 3, 4, 5 can be performed by means of derivation rules. Reduction of similar instructions can be performed by using rule 3; elimination of blind labels, inaccesible labels, and labels can be made by means of rules 3, 6 and 7; elimination of reflexive instructions can be performed by means of rules 2, 4 and 5. It is only to show, that

$$(\sim \underline{\text{true}}\ (Q * f)) \supset (\sim \underline{\text{true}}\ (Q * f))f.$$

Indeed, assume $\sim \underline{\text{true}}\ (Q*f)x = 1$ for some x in X_A. That is, there is no such y in X_A that $y = (Q*f)x$, hence, by definition of $Q*f$, for every z in X_A there is no such y in X_A that $y = (Q*f)z$ and $z = fx$, but it means that $(\sim \underline{\text{true}}\ (Q*f))fx = 1$.

L e m m a 7. If P_A is a well-defined program in a closed programming system A, and $P_A \vdash a$: if Q then loop, then for all x in X_A such that $Qx = 1$, there is no such y in X_A that $P_A: (a,x) \Longrightarrow (\underline{\text{stop}}, y)$. (*)

Proof. Let r denotes the instruction a: if Q then loop, and let $(r_0, r_1, \ldots, r_n)$, $n \geq 0$, be a derivation of r from P_A.

If $n = 0$, then either r is in P_A and by the definition of a program and by Proposition 2 the assertion holds, or r arises by rule 6 or 7 from P_A, and then obviously the assertion holds as well. Assume the assertion is true for $k < n$, $k > 0$; we shall prove it for $n = k$. There are three cases to be considered.

(1) if r_k arises from r_i, $i < k$, by rule 1, then r_i is of the form a: if Q_1 then loop, and by induction hypothesis, $Q_1x = 1$ implies the conclusion. But, in the case, $Qx = 1$ implies $Q_1x = 1$, hence $Qx = 1$ implies the conclusion, too;

(2) if r_k arises from r_i, r_j, $i<k$, $j<k$, by rule 2 or rule 3, then the conclusion is true by Proposition 2;

(3) if r_k arises from r_i, $i<k$, by rule 4, then by Proposition 2 (v) the assertion is true.

Note that r_k can not arise by rule 5, since there is no instruction with loop as its initial label. Hence, by induction, we obtain the desired result.

T h e o r e m 2. For any well-defined program P_A in a closed programming system A, and for arbitrary Q in Q_A, f in F_A, the following equivalences are true:

(i) $P_A \vdash$ start: if Q then f and stop if and only if for all x, y in X_A, $Qx = 1$ and $y = fx$ implies $\mathrm{Comp}_{P_A}(x,y)$;

(ii) $P_A \vdash$ start: if Q then loop if and only if for all x in X_A, $Qx = 1$ implies that there is no y in X_A such that $\mathrm{Comp}_{P_A}(x,y)$.

Proof. (i) (a) Assume $P_A \vdash$ start: if Q then f and stop and $Qx = 1$, $y = fx$. First, observe that for any instruction a: if Q then f and b in P_A and for any x, y, in X_A such that $Qx = 1$ and $y = fx$, we have $P_A: (a,x) \Longrightarrow (b,y)$. Next, on the basis of Proposition 2, the derivation rules preserve this property, namely, if $P_A \vdash a$: if Q then f and b, then for all x, y in X_A such that $Qx = 1$, $y = fx$, $P_A: (a,x) \Longrightarrow (b,y)$. Hence, by the definition of Comp_P, we obtain $\mathrm{Comp}_{P_A}(x,y)$.

(b) Assume that $Qx = 1$ and $y = fx$ implies $\mathrm{Comp}_{P_A}(x,y)$. By Corollary 1 there exists a condition Q_{P_A} and an operator f_{P_A} such that $\mathrm{Comp}_{P_A}(x,y)$ implies $Q_{P_A}x = 1$, $y = f_{P_A}x$. Then, $Q \supset Q_{P_A}$, $Q \supset (f = f_{P_A})$. By Lemma 6 we have

$$P_A \vdash \text{start: if } Q_{P_A} \text{ then } f_{P_A} \text{ and stop}$$

Thus, by rule 1, we obtain $P_A \vdash$ start: if Q then f and stop, what together with (a) gives the first part of Theorem 2.

(ii) (a) Assume $P_A \vdash$ start: if Q then loop. By Lemma 7 we obtain directly that for all x in X_A such that $Qx = 1$, there is no y in X_A such that $\mathrm{Comp}_{P_A}(x,y)$.

(b) Assume $Qx = 1$ implies that there is no y in X_A such that $\mathrm{Comp}_{P_A}(x,y)$. By Corollary 1 there exists Q_{P_A} in Q_A with $(\sim Q_{P_A})x = 1$ if there is no y in X_A such that $\mathrm{Comp}_{P_A}(x,y)$. Thus $Q \supset (\sim Q_{P_A})$. By Lemma 6 we have

$$P_A \vdash \text{start: if } \sim Q_{P_A} \text{ then loop,}$$

and by rule 1 we obtain $P_A \vdash$ start: if Q then loop, what, together with (a), completes the proof of Theorem 2.

This Theorem is a kind of "completeness theorem" for our derivation system.

Corollary 4. Let r_1 = start: if Q then f and stop, let r_2 = = start: if $\sim Q$ then loop, and let $R = \{r_1, r_2\}$. Then, for any well-defined program P_A, $P_A \vdash r_1$ and $P_A \vdash r_2$ implies P_A is equivalent to R.

Proof. Since $P_A \vdash r_1$ then $Qx = 1$ and $y = fx$ implies $\mathrm{Comp}_{P_A}(x, y)$, what proves

$$\mathrm{Comp}_R(x,y) \quad \text{implies} \quad \mathrm{Comp}_{P_A}(x,y).$$

If $\mathrm{Comp}_R(x,y)$ does not hold, then, by Theorem 2, since $P_A \vdash r_2$, the equality $(\sim Q)x = 1$ implies that there is no y in X_A such that $\mathrm{Comp}_{P_A}(x,y)$. Hence the proof is completed.

This Corollary together with Theorem 2 shows how to construct the canonical form of a given program by means of the derivation rules.

Example. Let us consider an Algol 60 program P:

start : i := 1;

b : s := m := $a[i]$;

c : if i = n then go to stop;

d : i := i + 1;

e : s := s + $a[i]$;

f : if $m > a[i]$ then go to c;

g : m := $a[i]$;

h : go to c;

We extend the Algol language allowing simultaneous assignments (as e.g. x, y := $x + y$, $2 * x - y$). We shall assume the interpretation of assignments and conditions to be known. At first, we translate the program into a program in our programming system:

1. start: i := 1 and b
2. b : s, m := $a[i]$, $a[i]$ and c
3. c : if $i = n$ then stop

4. c: **if** $i \neq n$ **then** d
5. d: $i := i + 1$ **and** e
6. e: $s := s + a[i]$ **and** f
7. f: **if** $m \geqslant a[i]$ **then** c
8. f: **if** $m < a[i]$ **then** g
9. g: $m := a[i]$ **and** h
10. h: c

Now we use the derivation rules 1-7 together with their consequences 8-15 given in Proposition 4. The signs $\vdash$ will be omitted in the derivation. On the right-hand side of every derived instruction we shall write numbers of used lines and, after R letter, the number of used rule.

11. f: **if** $m < a[i]$ **then** $m := a[i]$ **and** h — 8, 9, R10

12. f: **if** $m < a[i]$ **then** $m := a[i]$ **and** c — 11, 10, R11

13. f: **if** $m \geqslant a[i]$ **then** $m := m$ **and** c — 7, R1

note here that $(m := m) =$ **empty**;

14. f: $m := \max(m, a[i])$ **and** c — 12, 13, R3

note that $(x \geqslant y | x | y) = \max(x, y)$;

15. c: **if** $i < n$ **then** d — 4, R14

16. c: **if** $i > n$ **then** d — 4, R14

because $i \neq n \supset i < n$ or $i > n$

17. d: $i, s := i + 1, s + a[i + 1]$ **and** f — 5, 6, R2

note the effect of the composition of assignments;

18. d: $i, s, m := i + 1, s + a[i + 1], \max(m, a[i + 1])$ **and** c — 17, 14, R2

19. c: **if** $i < n$ **then** $i, s, m := i + 1, s + a[i + 1], \max(m, a[i + 1])$ **and** c — 15, 18, R2

20. c: **if** $\bigwedge_{j=1}^{n-i} (i + j - 1 < n)$ **then** $i, s, m := n, s + \sum_{j=1}^{n-i} a[i + j], \max(m, \max_{j=1}^{n-i} a[i + j])$ **and** c — 19, R8

because it can be proved by induction that if

$f = (i, s, m := i + 1, s + a[i + 1], \max(m, a[i + 1]))$,

then

$$f^k = (i,\ s,\ m := i + k,\ s + \sum_{j=1}^{k} a[i+j],\ \max(m, \max_{j=1}^{k} a[i+j]))$$

and

$$\bigwedge_{j=1}^{k} (i < n)(i := i + 1)^{j-1} \equiv \bigwedge_{j=1}^{k} (i + j - 1 < n);$$

21. c: **if** $i < n$ **then** $i, s, m := n,\ s + \sum_{j=1}^{n-i} a[i+j], \max(m,$
$\max_{j=1}^{n-i}, a[i+j])$ **and** c — 20, R1

because $i < n \supset \bigwedge_{j=1}^{n-i} (i + j - 1 < n)$;

22. c: **if** $i < n$ **then** $i, s, m := n,\ s + \sum_{j=i+1}^{n} a[j],$
$\max(m, \max_{j=i+1}^{n} a[j])$ **and** c — 21, R1

23. c: **if** $i < n \wedge n = n$ **then** $i, s, m := n,\ s + \sum_{j=i+1}^{n} a[j],$
$\max(m, \max_{j=i+1}^{n} a[j])$ **and stop** — 22, 3, R2

24. c: **if** $i < n$ **then** $i, s, m := n,\ s + \sum_{j=i+1}^{n} a[j],$
$\max(m, \max_{j=i+1}^{n} a[j])$ **and stop** — 23, R1

25. c: **if** $i = n$ **then** $i, s, m := n,\ s + \sum_{j=i+1}^{n} a[j],$
$\max(m, \max_{j=i+1}^{n} a[j])$ **and stop** — 3, R1

Because $i = n$ implies $(i, s, m := n,\ s + \sum_{j=i+1}^{n} a[j],$
$\max(m, \max_{j=i+1}^{n}, a[j])) = (i, s, m := i, s, m) =$ **empty**;

26. c: **if** $i \leq n$ **then** $i, s, m := n,\ s + \sum_{j=i+1}^{n} a[j],$
$\max(m, \max_{j=i+1}^{n} a[j])$ **and stop** — 24, 25, R12

27. c: **if** $i > n$ **then** $i, s, m := i + 1,\ s + a[i+1], \max(m,$
$a[i+1])$ **and** c — 15 18. R2

28. c: <u>if</u> $i > n$ <u>then</u> <u>loop</u> 27, R4

because $i > n$ implies $i+1>n$;

29. <u>start</u>: $i, s, m := 1, a[1], a[1]$ <u>and</u> c 1, 2, R2

30. <u>start</u>: <u>if</u> $1 \leqslant n$ <u>then</u> $i, s, m := n, \sum_{j=1}^{n} a[j], \max_{j=1}^{n} a[j]$ <u>and</u> <u>stop</u> 29, 26, R2

31. <u>start</u>: <u>if</u> $1 > n$ <u>then</u> <u>loop</u> 29, 28, R2

Hence, by Corollary 4, we have proved that P is equivalent to the following program:

$\{$<u>start</u>: <u>if</u> $1 \leqslant n$ then $i, s, m := n, \sum_{j=1}^{n} a[j], \max_{j=1}^{n} a[j]$
<u>and</u> <u>stop</u>,
<u>start</u>: <u>if</u> $1 > n$ <u>then</u> <u>loop</u>$\}$.

Of course, the presented proof seems to contain too many details; however, like in common mathematical practice,we omit usually some steps in a derivation.

REFERENCES

[1] A. B l i k l e: Algorithmically definable functions. Dissertationes Mathematicae, LXXXV, Warszawa, PWN, 1971, pp. 1-56.

[2] A. B l i k l e: Nets; complete lattices with a composition. Bull. Acad. Polon. Sci., Ser. Sci. Math. Phys. Astronom. vol. XIX, No 12, 1971, pp. 1123-1127.

[3] A. B l i k l e: Iterative systems: an algebraic approach. Ibid (to appear).

[4] A. B l i k l e: Complex iterative systems. Ibid (to appear).

[5] A. M a z u r k i e w i c z: Abstract Algorithms and Their Closures (to appear).

[6] Z. P a w l a k: Maszyny programowane (in polish). Algorytmy 10, 5-9.

SPIS TREŚCI

ERRATA

Page	Line	For	Read
7	29	taht	that
17	6	$P_A \vdash b$: <u>loo</u>	$P_A \vdash b$: <u>loop</u>
21	24	f: <u>if</u> $m \geqslant a; [i]$	f: <u>if</u> $m \geqslant a[i]$
23	3	$\bigwedge_{j=1}^{k} (i < \ )$	$\bigwedge_{j=1}^{k} (i < n)$
23	5	$\max_{j=1}^{n-i},$	$\max_{j=1}^{n-i}$
23	10	$\max_{j=i+1}^{n} a\|j\|$	$\max_{j=i+1}^{n} a[j]$
23	15	ecause	because
23	16	$\max_{j=i+1}^{n},$	$\max_{j=i+1}^{n}$

A. W. Mazurkiewicz — *Closed programming systems*

Cena zł 10,—