BANACH CENTER PUBLICATIONS
VOLUME 2

AN INTRODUCTION TO ALGORITHMIC LOGIC

METAMATHEMATICAL INVESTIGATIONS IN THE THEORY OF PROGRAMS

L. BANACHOWSKI, A. KRECZMAR, G. MIRKOWSKA, H. RASIOWA, A. SALWICKI

Department of Mathematics and Mechanics, Warsaw University, Warsaw

CONTENTS

The contribution of authors was as follows:
L. Banachowski: I. § 4, II. § 5, IV, VI. § 6, § 7
A. Kreczmar: III
G. Mirkowska: I. § 5, § 9, § 11, § 12, II. § 1–4
H. Rasiowa: Introduction, VII
A. Salwicki: I. § 1–3, § 6–8, § 10, § 13, V, VI. § 1–5

INTRODUCTION

The strong development of computer science inclines mathematicians to look for mathematical models of the most important concepts which occur in this field such as a computing machine, a program, a computation, etc.

This is a natural way of creating mathematical theories, which first try to describe a reality in a more or less detailed way and are later developed independently, being often applied to fields not previously anticipated. It seems quite obvious that there are various approaches to the concepts occurring in computer science, in particular to programs. Up to the present various ideas, methods and approaches have been applied in order to develop a mathematical programming theory.

One of the possible research methods is to present programming theory as a formalized logical system. Attempts to find a proper and simple logical system, constituting a basis of programming theory, and sufficiently rich to be applied in sophisticated investigations, led to the creation of algorithmic logic and its various extensions.

Algorithmic logic has been formulated in the doctoral dissertation of A. Salwicki and developed in several papers by mathematicians of the University of Warsaw (L. Banachowski, A. Kreczmar, G. Mirkowska-Salwicka, H. Rasiowa, A. Salwicki, and others).

The first purpose of introducing algorithmic logic has been to have a tool for finding and formulating the most important laws concerning computational processes, independently of computing machines, of programming languages, and of computation objects. Those laws are like the laws of propositional calculi or of predicate calculi. Methodological investigations concerning problems which occur in programming and investigations which could improve programming have been adopted as further topics of research.

The method of defining a system of algorithmic logic is analogous to that used to define any formalized logical system.

To begin with, formalized algorithmic languages are introduced. Any such language is obtained from a first order predicate language without quantifiers by adjoining certain new expressions, to be called substitutions, and—in addition—program operator signs: $\circ$ (composition sign), $\curlyvee$ (branching sign), $*$ (iteration sign), and the iteration quantifiers. Substitutions are adopted as atomic programs. Roughly speaking, they correspond to programs like:

$$x := y \cdot z \textbf{ and } y := z+1 \textbf{ and } p := x < y \qquad \text{(simultaneous assignment)}$$

in ALGOL-like languages. Other programs in a formalized language of algorithmic logic are constructed from substitutions by means of program operators and by using also formulas (in the case of branching and iteration). The intuition connected with a composition $\circ\,[KM]$ of two programs K and M is obvious. As regards a branching $\curlyvee\,[\alpha KM]$ it corresponds to "**if** α **then** K **else** M". An iteration $*\,[\alpha K]$ corresponds to "**while** α **do** K". The set of all "program-expressions" is precisely defined and denoted by ***FS***. The elements of ***FS*** are also called ***FS***-expressions. It is clear that ***FS***-expressions are interpreted as programs without recursive procedures. Any formalized algorithmic language contains also generalized terms and generalized formulas. Any term is a generalized term and any formula is a generalized formula. Among the general terms there are expressions of the form $K\tau$, where K is a program and τ is a term. The intuitive meaning of $K\tau$ is: τ after performing K. Among generalized formulas there occur expressions of the following forms: $K\alpha$, $\bigcup K\alpha$, $\bigcap K\alpha$, where K is a program, α is a formula and $\bigcup$, $\bigcap$ are the existential iteration quantifier and the universal iteration quantifier, respectively. The intuitive meaning of $K\alpha$ is: α after performing K; if K does not stop, then $K\alpha$ is going to be false. The formulas $\bigcup K\alpha$ and $\bigcap K\alpha$ are interpreted as infinite disjunction $\alpha \vee K\alpha \vee KK\alpha \vee \vee \ldots$ and infinite conjunction $\alpha \wedge K\alpha \wedge KK\alpha \wedge \ldots$, respectively. Any realization (in a non-empty set J) of functors (symbols of functions) and predicates (symbols of relations) occurring in an algorithmic language is extended to the set of all well-formed expressions. Valuations in J, i.e., mappings assigning to individual variables elements in J and to propositional variables elements in the two-element Boolean algebra, are considered as memory states (state vectors). Programs, i.e., ***FS***-expressions, are realized as partial mappings of the set of all state vectors into itself.

Properties of programs are expressible by means of generalized formulas. For instance, let K, M be any programs, α, β any formulas, and $\mathbf{1}$ a propositional constant interpreted as a symbol of any true sentence. Then for any realization and any state vector v, $K\mathbf{1}$ is true if and only if by this realization program K stops for the initial state vector v. Thus $K\mathbf{1}$ describes the stop property for K. Similarly, formula $(\alpha \Rightarrow K\beta)$ describes the correctness of K with respect to an initial condition α (for the data) and a terminal condition β (for the results of the computations). The formula $((\alpha \wedge K\mathbf{1}) \Rightarrow K\beta)$ describes a partial correctness of K, namely by the assumption that K stops. The formula $(K\alpha \Leftrightarrow M\alpha)$ describes an equivalence of programs K, M with respect to a terminal condition α for the results of the computations.

It follows that the investigations of properties of programs can be reduced to those concerning the satisfiability and the validity of corresponding generalized formulas in certain or in all realizations.

In any algorithmic language the semantic consequence operation is defined, as usual, by means of realizations. A formalized algorithmic language and the consequence operation in that language constitute a system of algorithmic logic. If, moreover, a set of generalized formulas is adopted as a set of specific axioms, we obtain an algorithmic theory.

Systems of algorithmic logic have been examined by G. Mirkowska-Salwicka in her doctoral dissertation [17]. Let us mention some of her results. The compactness property does not hold. One obtains an axiomatization and a formalization of the Hilbert type of the systems of algorithmic logic (based on enumerable languages) with the completeness theorem on applying the algebraic method due to H. Rasiowa and R. Sikorski (see [49]). The formalization in question uses two infinitistic rules of inference of ω-type. The completeness theorem makes it possible to reduce the examination of properties of programs to the verification whether the generalized formulas describing those properties are provable or not. A formalization of the Gentzen type of systems of algorithmic logic by using the method of diagrams of formulas (see H. Rasiowa and R. Sikorski [49]) has also been formulated. This is more convenient with regard to automata proving theorems. Proofs have been obtained of an analogue of Herbrand's theorem for certain generalized formulas. A theorem on a normal form of a program has been obtained. An algebraization of algorithmic languages and their semantics has been used to get simpler and purely algebraic proofs of various metamathematical theorems.

Effectivity problems in algorithmic logic have been examined by A. Kreczmar, most of them in his doctoral dissertation [13]. He proved that the set of all valid generalized formulas of algorithmic logic is recursively isomorphic to the set of formulas true in the standard model of the arithmetic of natural numbers and that the set of all consequences of a set A of generalized formulas is hyperarithmetical with respect to A. These theorems establish an infinitistic character of algorithmic logic. Other results concern the degrees of unsolvability of fundamental properties of programs (such as the stop property, correctness, equivalence). These properties have been examined in the class of all realizations, in the class of all realizations in finite sets, in the class of relational systems isomorphic to the standard model of the arithmetic of natural numbers, and in the class of relational systems isomorphic to the system $(R, 0, 1, +, \cdot, -, ^{-1})$ of real numbers. Using algebraic and metamathematical methods which may be applied to algorithmic logic, Kreczmar presents new simple proofs of known theorems (eliminating Gödel's numerations and Turing's machines) and certain new results.

Problems of the definability and programmability of functions, relations and relational systems in algorithmic logic have been investigated by A. Salwicki [31]. Theorems on the elimination of explicit definitions in systems of algorithmic logic express the possibility of eliminating subroutines in programs without recursive

procedures. Functions and relations are said to be programmable if they are definable in algorithmic logic by means of explicit definitions of a special type. Programmable relations are said to be stringly programmable if their complements are programmable. An extension $\mathfrak{A}$ of a relational system $\mathfrak{B}$ is said to be its strongly programmable extension if all functions and relations in $\mathfrak{A}$ are strongly programmable in $\mathfrak{B}$. Two relational systems are constructively equivalent if each of them is a strongly programmable extension of another. Relational systems constructively equivalent to the standard model of the arithmetic of natural numbers are said to be constructive. A. Salwicki [31] presents a theory of programmability and its relationship with the theory of recursive functions. In particular, he obtains a generalization of the Post theorem, a generalization of Shepherdson's and Sturgis' theorem [54] and a theorem establishing necessary and sufficient conditions for a relational system to be constructive.

Problems concerning the correctness of programs and modular properties of programs have been investigated by many authors (Floyd [37], Manna [45], Hoare [38] and others). An analysis of the notions of a modular structure and of a description of a program on the basis of algorithmic logic has been carried out by L. Banachowski [6], [7]. It turns out that the various approaches used by the authors mentioned above can be included and presented in a uniform way in a theory of modular properties of programs as constructed by Banachowski. Algorithmic logic extended by adjoining the existential and the universal quantifiers has been a tool for developing this theory.

The **case** statements

$$\textbf{case}\ \text{expression}\ \textbf{of begin}\ K_1; \ldots; K_n\ \textbf{end},\quad n = 2, 3, \ldots,$$

which occur in programming languages and are generalizations of **if then else** statements can be described in a natural way in ω^+-valued algorithmic logic. Similarly, generalizations of **while do** statements are easily expressible in that logic. Formalized ω^+-valued algorithmic languages may contain m-valued predicates for all $m \geqslant 2$. As a semantic basis of that logic a generalized Post algebra $\mathfrak{P}_\omega$ is adopted. Its elements form a chain $\nearrow = e_0 \leqslant e_1 \leqslant e_2 \leqslant \ldots \leqslant e_\omega = \swarrow$ isomorphic to the chain of all ordinals not greater than ω. $\mathfrak{P}_\omega$ is a linear pseudo-Boolean algebra with respect to the operations $\cup, \cap, \Rightarrow, \neg$. Moreover it has operations d_i, $i = 1, 2, \ldots$, defined by the equations $d_i(e_j) = \swarrow$ if $i \leqslant j$, and $d_i(e_j) = \nearrow$ if $i > j$. The set $\{\nearrow, \swarrow\}$ is the two-element Boolean algebra with respect to $\cup, \cap, \Rightarrow, \neg$. Metamathematical investigations concerning this logic are based on the theory of generalized Post algebras of order ω^+. There is a close connection between $\mathfrak{P}_\omega$ and the two-element Boolean algebra, and more generally, between Post algebras of order ω^+ and Boolean algebras. This relationship permits us to associate with any system of ω^+-valued algorithmic logic an equivalent theory based on a system of algorithmic logic. However the formalized language of this theory must be much richer. Any m-valued predicate is replaced by a sequence of $(m-1)$ two-valued predicates. The application of ω^+-valued algorithmic logic simplifies considerably the description of very

complicated programs. The formulation of this logic and a formalization of the Hilbert type with the completeness theorem are to be found in [20]–[23] by Rasiowa.

Procedures are an essential tool in programming because there are functions programmable by procedures which are not definable by any program which does not contain a recursive procedure (see e.g. Dańko [8]).

An extended ω^+-valued algorithmic logic (see Rasiowa [24], [25], [26]) is a tool for investigating procedures and programs with procedures. Investigations concerning these problems are continued. It is proper to add that this approach links procedures to Mazurkiewicz's pushdown algorithms [33] and that the application of ω^+-valued logic is essential for introducing labels and describing the control functions of pushdown algorithms. However, it is possible to eliminate from formalized languages of extended ω^+-valued algorithmic logic m-valued predicates for $m > 2$ (see Rasiowa [26]).

Another approach to recursive procedures, due to A. Salwicki, is based on the observation that procedures can be treated as implicit definitions of a special type and thus they are generalized formulas in algorithmic languages. He introduced the notion of a formal computation distinguishing computation by value and by name. Procedures are treated as axioms of theories in extended languages of algorithmic logic. This enabled him to investigate models of systems of procedures and to obtain certain theorems concerning these models.

The recent investigations on algorithmic logic and its extensions conducted by mathematicians at the University of Warsaw, deal with problems connected with recursive procedures, coroutines, concurrent programming, data structures, automata proving theorems in algorithmic logic, and algorithmic theories of certain relational systems, e.g. standard model of arithmetic of natural numbers, a system of lists LISP, etc.

The present survey of results in algorithmic logic is of a cursory and superficial character. It neither includes the whole of the research nor pretends to uniformity of presentation. The aim of this paper is to introduce the reader to problems connected with this trend of research and to refer him to original papers.

Chapter I

ALGORITHMIC LANGUAGES AND THEORIES

The aim of this chapter is to express the fundamental properties of programs as axioms of formalized algorithmic theories based on algorithmic logic. We state the completeness property of the formalization given here. In order to be able to formulate the axioms we have standardized the language, simplifying its syntax and defining semantics in a precise way. All the existing programming languages can be reduced, if necessary, to the suggested here.

In addition, the notions of definability in algorithmic logic are introduced.

The chapter is designed as an introduction to the remaining ones. We have tried to make it selfcontained; for more details the reader is advised to refer to [11], [15], [25], [28].

§ 1. Relational systems

By a *relational system* we understand the following system

$$\mathfrak{A} = \langle \mathscr{J}, \{o_i\}_{i\in I}, \{r_k\}_{k\in K}\rangle$$

where $\mathscr{J}$ is a set called the *universe* of the system $\mathfrak{A}$ for every $i \in I$, o_i is an n_i-ary operation in $\mathscr{J}$, $o_i\colon \mathscr{J}^{n_i} \to \mathscr{J}$ and, for every $k \in K$, r_k is an m_k-ary relation, $r_k \subset \mathscr{J}^{m_k}$.

EXAMPLES. 1. Let $\mathscr{J}$ be a two-element set with elements denoted by $\mathbb{V}$ and $\mathbb{A}$, respectively. Let o_1 be a unary operation defined as follows:

$$o_1(\mathbb{V}) = \mathbb{A}, \quad o_1(\mathbb{A}) = \mathbb{V}.$$

Let o_2 be a binary operation satisfying

$$o_2(\mathbb{V},\mathbb{V}) = \mathbb{V}, \quad o_2(\mathbb{V},\mathbb{A}) = \mathbb{A}, \quad o_2(\mathbb{A},\mathbb{V}) = \mathbb{A}, \quad o_2(\mathbb{A},\mathbb{A}) = \mathbb{A}.$$

We can also define further binary operations o_3 and o_4 by the equalities

$$o_3(a,b) = o_1\big(o_2(o_1(a), o_1(b))\big), \quad o_4(a,b) = o_3\big(o_1(a), b\big), \quad a, b \in \{\mathbb{A}, \mathbb{V}\}.$$

The only relation considered is the binary relation of identity $\{(\mathbb{V},\mathbb{V}), (\mathbb{A},\mathbb{A})\}$ denoted by $=$.

The relational system

$$\mathfrak{B} = \langle \{\mathbb{V}, \mathbb{A}\}, o_1, o_2, o_3, o_4, =\rangle$$

is called the *two-element Boolean algebra* and will repeatedly appear in the sequel. We shall then use the signs $-$ and $\cap$, $\cup$, $\to$ to denote operations o_1, o_2, o_3, o_4 and B to denote the set $\{\mathbb{V}, \mathbb{A}\}$; hence $\mathfrak{B} = \langle B, -, \cap, \cup, \to, =\rangle$.

2. *The arithmetic of natural numbers.* This is a relational system with the following universe: the set of natural numbers $\mathscr{N}$, one zero-argument operation 0—zero, one unary operation S—succesor (add one) and the binary relation of identity $=$.

$$\mathfrak{Ar} = \langle \mathscr{N}, 0, \mathrm{S}, =\rangle.$$

3. *The arithmetic of a computer.* The universe of this relational system is composed of all binary words of length k, i.e., all sequences of length k composed of two signs, 0 and 1. The length k as well as the operations depend on the type of the computer. This is clearly understood. The relations are usually unary: "to be equal to zero", "to be positive", etc.

4. *The relational system connected with the* ALGOL-60. The universe of this system is the sum of the following sets:

$\mathscr{R}$ of real numbers,

{**true**, **false**} of Boolean values,

STR of strings.

The operations considered are partial operations on real (or integer) numbers and on Boolean values and some operations on strings which must be expressed in code.

5. *The relational system connected with the* SIMULA-67 is more complicated. In fact, we should speak about a new relational system for every SIMULA program as it enriches the basic universe by introducing new classess of objects, with appropriate operations defined by procedures.

§ 2. Formalized language of algorithmic logic

In order to describe the properties of a relational system, programs and the properties of programs we need a language with signs that correspond to operations, relations and memory locations or variables and with signs that denote the programming constructions such as branching if ... then else ..., for example. Expressions of the formalized language will be interpreted as mappings from the set of memory states into corresponding subsets of the universe. Various kinds of expressions correspond to arithmetic and Boolean expressions, to programs and finally there are some (generalized) formulas that describe properties of programs.

The realization (interpretation) of a formalized language in a relational system is defined in the next section.

A formalized language is fully described by two elements:

—its alphabet A (i.e. a collection of signs),

—its set of well-formed expressions.

The *alphabet of a formalized algorithmic language* is the union of disjoint, at most enumerable sets

$$A = V_i \cup V_0 \cup \Phi \cup P \cup L_0 \cup L_1 \cup L_2 \cup Q \cup \Pi \cup U.$$

Elements of V_i will be called *individual variables* and denoted by x, y, z (with indices if necessary).

Elements of V_0 will be called *propositional variables* and denoted by p, q (with indices if necessary).

The set Φ is the union of disjoint sets Φ_n ($n \in \mathcal{N}$, where $\mathcal{N}$ is the set of non-negative integers). Elements of Φ_n will be called *n-argument functors* and denoted by φ, ψ (with indices if necessary). We assume that at least one of the sets Φ_n is nonempty.

The set P is the union of disjoint sets P_m (where m is a positive integer). Elements of P_m will be called *m-argument predicates* and denoted by ϱ, η (with indices if necessary). We assume that the sign of equality $=$ belongs to P_2.

The set L_0 contains exactly two elements called *propositional constants*. They will be denoted by **1** (true sentence) and **0** (false sentence).

The set L_1 contains one element called the *negation sign* and denoted by $\neg$. Negation is a 1-argument logical connective.

The set L_2 contains three elements called *2-argument logical connectives*. They are called the *disjunction sign*, the *conjunction sign* and the *implication sign*, and are denoted by $\vee$, $\wedge$, $\Rightarrow$.

The set Q contains two elements, called *iteration quantifiers* and denoted by $\bigcup$ the existential iteration quantifier and $\bigcap$ the universal iteration quantifier.

The set Π contains three elements, called *program connectives* and denoted by $\circ$ composition sign, $\curlywedge$ branching sign, $*$ iteration sign.

Elements of the set U are called *auxiliary signs*. We assume that U contains elements denoted by / slash, (,) parenthesis, [,] brackets.

By a *language $\mathscr{L}$ of algorithmic logic* or shortly, an *algorithmic language* we shall understand a system

$$\mathscr{L} = \langle A, T \cup F^0 \cup S \cup FS \cup FST \cup FSF \rangle,$$

where A is the alphabet of $\mathscr{L}$ and the set

$$T \cup F^0 \cup S \cup FS \cup FST \cup FSF$$

forms the set of expressions belonging to $\mathscr{L}$, i.e., the set of well-formed expressions.

We start with the definitions of T and F^0.

The set T of terms is the least set of expressions over A satisfying:

(t1) if $x \in V_i$ then $x \in T$,

(t2) if $\varphi \in \Phi_n$, $\tau_1, \ldots, \tau_n \in T$ then $\varphi(\tau_1, \ldots, \tau_n) \in T$.

The elements of T—terms—will be denoted by τ (with indices if necessary).

The set F^0 of open (quantifier-free) formulas is the least set of expressions over A satisfying:

(f0) $\mathbf{0}, \mathbf{1} \in F^0$,

(f1) if $a \in V_0$ then $a \in F^0$,

(f2) if $\varrho \in P_m$, $\tau_1, \ldots, \tau_m \in T$ then $\varrho(\tau_1, \ldots, \tau_m) \in F^0$,

(f3) if $\alpha, \beta \in F^0$ then the expressions

$$(\alpha \vee \beta), (\alpha \wedge \beta), (\alpha \Rightarrow \beta), \neg\alpha$$

belong to F^0.

The elements of F^0—formulas—will be denoted by α, β, γ (with indices if necessary).

EXAMPLES. $x \cup y$, $(x \cup y) \cap z$, $(x \cap y) \cup (x \cap z)$ are examples of terms in formalized language of the lattice theory. $x \cup -y$, $x \cap (y \cup -z)$ are terms in formalized language of theory of Boolean algebras.

$S(x)$, 0, $S(S(x))$ are terms in formalized language of arithmetic.

In the examples above we have used another syntax for terms with binary functors.

$$x \cap y = z, \qquad ((x \cup z) \cap z = (x \cup z) \cup y) \Rightarrow y = z$$

are examples of formulas in the formalized language of lattice theory.

The set S of substitutions is the set of all expressions of the form

$$[x_1/\tau_1 \ \dots \ x_n/\tau_n \ a_1/\alpha_1 \ \dots \ a_m/\alpha_m]$$

where

$n \geqslant 0$, $m \geqslant 0$,

$x_1, \dots, x_n$ are different individual variables,

$a_1, \dots, a_m$ are different propositional variables,

$\tau_1, \dots, \tau_n$ are terms from the set T,

$\alpha_1, \dots, \alpha_m$ are open formulas from the set F^o.

Substitutions will be denoted by the letter s (with indices if necessary).

EXAMPLES. $[x/s(x)]$,

$[x/s(x) \ \ y/ux^2+tx+v \ \ a/((x+y)+1 = z) \Rightarrow ((a \vee b) \wedge b)]$.

Substitutions are the simplest programs.

The set FS of programs (FS-expressions) is the least set satisfying

(fs1) if $s \in S$ then $s \in FS$,

(fs2) if $\alpha \in F^o$, $K, M \in FS$ then the expressions $\circ[K\,M]$, $\curlyvee[\alpha K\,M]$, $*[\alpha K]$ are in FS.

Programs will be denoted by the letters K, M, N, L (with indices if necessary).

To see the connection with programming one can translate FS-expressions into an ALGOL-like language as follows:

1. Every substitution s of the form

$$[x_1/\tau_1 \dots x_n/\tau_n \ a_1/\alpha_1 \dots a_m/\alpha_m]$$

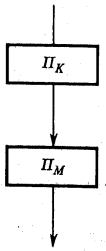
is to be read

$$x_1 := \tau_1 \textbf{ and } \dots \ x_n := \tau_n \textbf{ and } a_1 := \alpha_1 \textbf{ and } \dots \textbf{ and } a_m := \alpha_m.$$

This will be interpreted as simultaneous substitution (assignment instruction). One can also see that terms play the role of arithmetic expressions and open formulas play the role of Boolean expressions as in ALGOL.

2. Suppose that FS-expressions K and M are translated into programs Π_K and Π_M; then the program

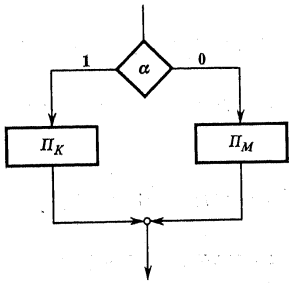
begin
Π_K;
Π_M
end



will be the translation of the FS-expression $\circ[K\,M]$.

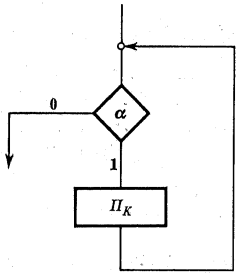
3. If the FS-expressions in question are $\curlyvee[\alpha K\,M]$ or $*[\alpha K]$ then they will be translated into

if α **then** Π_K **else** Π_M



and into

(**for** $i := i$) **while** α **do** Π_K



We call the attention of the reader to the fact that the $*$ sign used here differs in shape and meaning from a similar sign used in earlier works on algorithmic logic.

The set FST (of generalized terms) is the least set of expressions satisfying

(fst1) each individual variable $x \in V_i$ is in FST,

(fst2) if $\varphi \in \Phi_n$, $\tau_1, \dots, \tau_n \in FST$ then $\varphi(\tau_1 \dots \tau_n) \in FST$,

(fst3) if $K \in FS$, $\tau \in FST$ then the expression $K\tau \in FST$.

Elements of FST will be denoted by the letter τ (with indices if necessary). In order to distinguish between T and FST, terms belonging to T will be called *classical*.

EXAMPLES.

$[x/x+y \cdot z](z-x \cdot y)$;

$*[x < y \ \ [x/x+1]](x+y)+\curlyvee[x = y \ \ [u/x-1][u/x+1 \ \ z/2]](x+y-z)$.

The set FSF (of generalized formulas) is the least set of expressions satisfying:

(fsf0) $\mathbf{0}, \mathbf{1} \in FSF$,

(fsf1) if $a \in V_o$ then $a \in FSF$,

(fsf2) if $\varrho \in P_m$, $\tau_1, \ldots, \tau_m \in \boldsymbol{FST}$ then $\varrho(\tau_1 \ldots \tau_m) \in \boldsymbol{FSF}$,

(fsf3) if $\alpha, \beta \in \boldsymbol{FSF}$ then the expressions $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \Rightarrow \beta)$, $\neg\alpha$ belong to $\boldsymbol{FSF}$,

(fsf4) if $\alpha \in \boldsymbol{FSF}$, $K \in \boldsymbol{FS}$ then the expressions $K\alpha$, $\bigcup K\alpha$, $\bigcap K\alpha$ belong to $\boldsymbol{FSF}$.

Elements of $\boldsymbol{FSF}$ will be called *formulas* and denoted by α, β, γ (with indices if necessary).

EXAMPLES. $(x < y)$;

$\big(x < y \vee ([x/x+y \cdot z](z-x \cdot y)) = z\big)$;

$[x/0] \bigcup [x/x+1](x = y)$.

In the sequel it will be convenient to use the abbreviation $\alpha \Leftrightarrow \beta$ denoting the formula of the form $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.

By $V(\omega)$ we shall denote the set of all variables in an expression ω.

§ 3. Realization of the formalized language of algorithmic logic

The connection between formalized languages and relational systems is established through the notion of realization of a language $\mathscr{L}$ in a relational system $\mathfrak{A}$.

By a *realization of the language* $\mathscr{L}$ in a relational system $\mathfrak{A} = \langle \mathscr{J}, \{o_i\}_{i \in I}, \{r_k\}_{k \in I'} \rangle$, and the two-element Boolean algebra $\mathfrak{B}$ we shall understand any mapping R such that

(a) to every m-argument functor φ in Φ_m, R assigns an m-argument operation φ_R in $\mathfrak{A}$, i.e. $\varphi_R \colon \mathscr{J}^m \to \mathscr{J}$,

(b) to every m-argument predicate ϱ in $\mathscr{L}$, R assigns an m-argument function

$$\varrho_R \colon \mathscr{J}^m \to B$$

where ϱ_R is the characteristic function of a relation r_k, i.e.

$$\varrho_R(j_1 \ldots j_m) = \top \quad \text{iff} \quad (j_1 \ldots j_m) \in r_k \quad (k \in I').$$

Usually we shall assume that among the relations r_k $(k \in I')$ there is an identity relation and moreover that $=_R$ is its characteristic function.

By a *valuation v of variables* we understand any pair of mappings

$$v_i \colon V_i \to \mathscr{J},$$
$$v_0 \colon V_0 \to B.$$

The notion of valuation will be treated as a formalized equivalent of the notions: state of memory, vector-state, etc.

The set of valuations $\mathscr{J}^{V_i} \times B^{V_0}$ will be denoted by W.

Given the realization R of the language $\mathscr{L}$, we can interpret any expression of the language $\mathscr{L}$ as a mapping defined on the set of valuations W:

every term τ will be interpreted as

$$\tau_R \colon W \to \mathscr{J},$$

every formula α will be interpreted as a mapping

$$\alpha_R \colon W \to B,$$

every program K will be interpreted as a (partial) mapping

$$K_R \colon W \to W.$$

The details are as follows. Let $v = (v_i, v_0)$ be a valuation of variables. Then

(t1R) $$x_R(v) = v_i(x),$$

(t2R) $$\varphi(\tau_1, \ldots, \tau_n)_R(v) = \varphi_R\big(\tau_{1R}(v), \ldots, \tau_{nR}(v)\big),$$

(f0R) $$\mathbf{0}_R(v) = \nearrow, \quad \mathbf{1}_R(v) = \top,$$

(f1R) $$a_R(v) = v_0(a),$$

(f2R) $$\varrho(\tau_1, \ldots, \tau_m)_R(v) = \varrho_R\big(\tau_{1R}(v), \ldots, \tau_{mR}(v)\big),$$

(f3R) $$(\alpha \vee \beta)_R(v) = \alpha_R(v) \cup \beta_R(v),$$
$$(\alpha \wedge \beta)_R(v) = \alpha_R(v) \cap \beta_R(v),$$
$$(\alpha \Rightarrow \beta)_R(v) = \alpha_R(v) \to \beta_R(v),$$
$$(\neg\alpha)_R(v) = -\alpha_R(v);$$

(fs1R) $[x_1/\tau_1 \ldots x_n/\tau_n \; a_1/\alpha_1 \ldots a_m/\alpha_m]_R(v) = v'$ where

$$v'(x_i) = \tau_{iR}(v) \quad \text{for} \quad i = 1, \ldots, n,$$
$$v'(a_j) = \alpha_{jR}(v) \quad \text{for} \quad j = 1, \ldots, m,$$
$$v'(z) = v(z) \quad \text{for the remaining variables,}$$

(fs2R)
$$\circ[K\,M]_R(v) = \begin{cases} M_R(K_R(v)) & \text{if the valuations } v' = K_R(v) \text{ and } M_R(v') \text{ exist,} \\ \text{undefined} & \text{in the opposite case,} \end{cases}$$

$$\leq [\alpha K\,M]_R(v) = \begin{cases} K_R(v) & \text{if } \alpha_R(v) = \top \text{ and } K_R(v) \text{ is defined,} \\ M_R(v) & \text{if } \alpha_R(v) \neq \top \text{ and } M_R(v) \text{ is defined,} \\ \text{undefined} & \text{in the remaining cases,} \end{cases}$$

$$*[\alpha K]_R(v) = \begin{cases} K_R^i(v) & \text{if } i \text{ is the least integer such that all valuations } K_R^j(v) \; (j \leqslant i) \text{ are defined and } \alpha_R\big(K_R^i(v)\big) = \nearrow, \\ \text{undefined} & \text{if such an integer } i \text{ does not exist.} \end{cases}$$

EXAMPLES. Let K be the program

$$K \colon \leq [x = y[u/x-1][u/x+1 \;\; z/2]],$$

and let v be the valuation $v \colon \frac{x}{2}\,\frac{y}{3}\,\frac{z}{4}\,\frac{u}{5}$. The realization considered here is in the field of real numbers. The result $K_R(v)$ is the valuation $v' \colon \frac{x}{2}\,\frac{y}{3}\,\frac{z}{2}\,\frac{u}{3}$.

Let M be the program: $*[x < y[x/x+1]]$; then the resulting valuation $v'' = M_R(v)$ is defined and equal to v'': $\frac{x}{3}\frac{y}{3}\frac{z}{4}\frac{u}{5}$.

The realization of ***FST***-terms is defined as follows:

(fst1R) $$x_R(v) = v_i(x),$$

(fst2R) $$\varphi(\tau_1, \ldots, \tau_n)_R(v) = \begin{cases} \varphi_R(\tau_{1R}(v), \ldots, \tau_{nR}(v)) & \text{if all values } \tau_{iR}(v) \text{ are defined, } i = 1, \ldots, n, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

(fst3R) $$(K\tau)_R(v) = \begin{cases} \tau_R(K_R(v)) & \text{if } v' = K_R(v) \text{ and } \tau_R(v') \text{ are defined,} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The realization of ***FSF***-formulas is defined as follows:

(fsf0R) $$\mathbf{0}_R(v) = \nearrow, \quad \mathbf{1}_R(v) = \swarrow,$$

(fsf1R) $$a_R(v) = v_0(a),$$

(fsf2R) $$\varrho(\tau_1, \ldots, \tau_n)_R(v) = \varrho_R(\tau_{1R}(v), \ldots, \tau_{nR}(v)),$$

(fsf3R) $$(\alpha \vee \beta)_R(v) = \alpha_R(v) \cup \beta_R(v),$$
$$(\alpha \wedge \beta)_R(v) = \alpha_R(v) \cap \beta_R(v),$$
$$(\alpha \Rightarrow \beta)_R(v) = \alpha_R(v) \to \beta_R(v),$$
$$(\neg\alpha)_R(v) = -\alpha_R(v),$$

(fsf4R) $$(K\alpha)_R(v) = \begin{cases} \alpha_R(K_R(v)) & \text{if the valuation } v' = K_R(v) \text{ is defined,} \\ \nearrow & \text{otherwise,} \end{cases}$$
$$\Big(\bigcup K\alpha\Big)_R(v) = \underset{i \in \mathcal{N}}{\text{l.u.b.}}(K^i\alpha)_R(v),$$
$$\Big(\bigcap K\alpha\Big)_R(v) = \underset{i \in \mathcal{N}}{\text{g.l.b.}}(K^i\alpha)_R(v).$$

EXAMPLE. Let v be the valuation v:

$$\frac{x}{2}\frac{y}{35}\frac{z}{59}\frac{u}{9}\frac{a}{\nearrow}\frac{b}{\swarrow}\frac{c}{\swarrow};$$

then the value of the formula α: $([x/x+y \cdot z](z-x \cdot y)) = z \wedge b$ at the valuation v in the realization in the field of real numbers is $\nearrow$.

A formula $\alpha \in$ ***FSF*** is said to be *satisfied in the realization R by the valuation v* iff $\alpha_R(v) = \swarrow$.

A formula $\alpha \in$ ***FSF*** is said to be *valid in the realization R* iff for all valuations $\alpha_R(v) = \swarrow$.

We shall use the notation $\models_R \alpha[v]$ for the following phrase: the formula α is satisfied in the realization R by the valuation v. Similarly, $\models_R \alpha$ denotes that the formula α is valid in the realization R.

A formula is said to be a *tautology* iff it is valid in every realization R. Denotation: $\models \alpha$.

EXAMPLES. Formula $\bigcup [x/x+1](x = 5)$ is satisfied in the system $\mathfrak{Ar}$ of natural numbers by the valuation v: $\frac{x}{0}$. This formula, however, is not valid in $\mathfrak{Ar}$. In fact, let v' be a valuation such that $v'(x) = 6$. Then

$$\bigcup [x/x+1](x = 5)_{\mathfrak{Ar}}(v') = \underset{i \in \mathcal{N}}{\text{l.u.b.}}([x/x+1]^i(x = 5))_{\mathfrak{Ar}}(v')$$
$$= \underset{i \in \mathcal{N}}{\text{l.u.b.}}(x = 5)_{\mathfrak{Ar}}([x/x+1]^i_{\mathfrak{Ar}}(v'))$$
$$= \underset{i \in \mathcal{N}}{\text{l.u.b.}}(x = 5)_{\mathfrak{Ar}}(v'_i)$$

where $v'_i(x) = 6+i$.

Evidently, for every $i \in \mathcal{N}$,

$$(x = 5)_{\mathfrak{Ar}}(v'_i) = \nearrow \quad \text{and} \quad \bigcup [x/x+1](x = 5)_{\mathfrak{Ar}}(v') = \nearrow.$$

Formula $K(\alpha \vee \beta) \Rightarrow K\alpha \vee K\beta$ is a tautology. Let us consider a realization R and a valuation v. By the definition of realization, the equality $K(\alpha \vee \beta)_R(v) = \swarrow$ implies that $K_R(v)$ is defined and $(\alpha \vee \beta)_R(K_R(v)) = \swarrow$. Hence

$$\alpha_R(K_R(v)) \cup \beta_R(K_R(v)) = \swarrow \quad \text{and} \quad (K\alpha \vee K\beta)_R(v) = \swarrow.$$

§ 4. Algorithmic properties

We shall consider the following properties of programs.

1. *Stop property*—a program K stops (halts) in the realization R at the valuation v iff the resulting valuation $K_R(v)$ is defined. The formula $K\mathbf{1}$ is satisfied in R at v iff $K_R(v)$ is defined. We shall say that the formula $K\mathbf{1}$ *expresses the stop property*.

2. *Correctness with respect to an input formula α and an output formula β*—this property of a program K holds whenever the fact that the initial data of the program K satisfy the input formula α implies that the results are defined and satisfy the output formula β. This property is expressed by the formula $\alpha \Rightarrow K\beta$.

3. *Partial correctness with respect to an input formula α and an output formula β* is expressed by the formula $(\alpha \wedge K\mathbf{1} \Rightarrow K\beta)$. The program K is partially correct with respect to α and β iff for every valuation v if v satisfies α and K stops then the results $K_R(v)$ satisfy β.

4. *Equivalence of two programs K and M with respect to the formula α* can be defined according to different patterns as follows:

(a) $K\alpha \Leftrightarrow M\alpha$,

(b) $(K\mathbf{1} \Leftrightarrow M\mathbf{1}) \wedge (K\alpha \Leftrightarrow M\alpha)$ (denotation $K \equiv_\alpha M$),

(c) $K\mathbf{1} \wedge M\mathbf{1} \wedge (K\alpha \Leftrightarrow M\alpha)$.

In addition, the important properties of

being results of the program K, $\mathbf{1}K$;

adequacy of program for input and output conditions $(\alpha \Leftrightarrow K\beta) \wedge (\alpha K \Leftrightarrow \beta)$

can also be expressed.

The last two formulas belong to the language of extended algorithmic logic, see II, § 5.

In algorithmic logic one can characterize properties not expressible in classical logic.

1. Formula $[y/0]\bigcup[y/y+1](y = x)$ is valid iff a number x can be obtained from zero by adding a finite number of unities.

2. Formula $[x/1]\bigcap[x/x+1](x \neq 0)$ expresses the property of being of characteristic 0.

3. Formula $\big(x > 0 \wedge y > 0 \Rightarrow [z/y]\bigcup[z/z+y](x < z)\big)$ expresses the axiom of Archimedes.

§ 5. Normal form of programs

Every program of the form $\circ[K*[\alpha M]]$ where K and M are loop-free programs is called a *program in the normal form.*

5.1. *For every program K in $\boldsymbol{FS}$, there exists a program M in the normal form such that K and M are equivalent on the set $\big((V_i \cup V_0) - (V(M) - V(K))\big)$ of variables, i.e. for every realization R and for every valuation v, the valuation $K_R(v)$ is defined iff $M_R(v)$ is defined, and if they are defined then, for all variables $x \notin V(M) - V(K)$, $K_R(v)(x) = M_R(v)(x)$.* ([17].)

The variables $V(M) - V(K)$ have an auxiliary meaning only. They can always be chosen so that they do not occur in the formulas under consideration.

EXAMPLE. For any loop-free programs M, N and any open formulas α, β program $*\big[\alpha \circ \big[M*[\beta N]\big]\big]$ can be reduced to the normal form

$$\circ\Big[[q/\mathbf{0}]*\big[q \vee \alpha \circ \big[\curlyvee[qNM][q/\beta]\big]\big]\Big].$$

§ 6. Definability and programmability

Let $\mathfrak{A} = \langle \mathscr{J}, \{o_i\}_{i\in I}, \{r_k\}_{k\in I'}\rangle$ be a relational system and let $\mathscr{L}$ be a language associated with $\mathfrak{A}$ in the following manner:

for every o_i $(i \in I)$, n_i-argument operation in $\mathfrak{A}$, there exists in the alphabet of $\mathscr{L}$ an n_i-argument functor φ_{o_i},

for every r_k $(k \in I')$, m_k-argument relation in $\mathfrak{A}$, there exists in the alphabet of $\mathscr{L}$ an m_k-argument predicate ϱ_{r_k}.

The realization of the language $\mathscr{L}$ is defined in the obvious way.

We shall say that a relation $r \subset \mathscr{J}^n$ is *definable in the system* $\mathfrak{A}$ iff there exists a formula $\alpha \in \boldsymbol{FSF}$ with n free variables such that the equivalence $\varrho_r(x_1 \dots x_n) \Leftrightarrow \alpha$ is valid in $\mathfrak{A}$ provided that the predicate ϱ_r is realized as the characteristic function of the relation r.

For the further use it will be convenient to introduce a restriction of the notion of definability.

We shall say that the relation r is *programmable* iff it is definable by a formula of the form $K\alpha$ where $K \in \boldsymbol{FS}$, $\alpha \in \boldsymbol{F^o}$.

A relation r is *strongly programmable* iff r and its complement $\mathscr{J}^n - r$ are programmable.

The following theorem is an analogue of the Post theorem.

6.1. *A relation $r \subset \mathscr{J}^n$ is strongly programmable iff it is definable by a formula of the form $K\alpha$ $(K \in \boldsymbol{FS}, \alpha \in \boldsymbol{F^o})$ such that the formula $K\mathbf{1}$ is valid in $\mathfrak{A}$* ([31]).

EXAMPLE. Let us consider a language in which 0 is a constant zero-argument functor, $\mathbf{S}$ a one-argument functor, and $=$ binary predicate. The relation $<$ (less than) is defined by the following equivalence:

$$x < y \Leftrightarrow \curlyvee\Big[x = y[c/\mathbf{0}] \circ \Big[[u/\mathbf{0}] \circ \Big[*\big[\neg(u = y \vee u = x)[u/\mathbf{S}(u)]\big] \curlyvee\big[u = x\ [c/\mathbf{1}][c/\mathbf{0}]\big]\Big]\Big]c$$

which defines the relation $<$ in the set of natural numbers. This can be proved in a formal way, e.g. by showing that the right-hand side is semantically equivalent to the formula

$$[z/\mathbf{0}]\bigcup[z/\mathbf{S}(z)]\ \ x+z = y$$

in the set of natural numbers, or by proving the equivalence

$$\curlyvee\Big[x = y[c/\mathbf{0}] \circ \Big[[u/\mathbf{0}] \circ \big[*\big[\neg(u = y \vee u = x)[u/\mathbf{S}(u)]\big] \curlyvee\big[u \neq x[c/\mathbf{1}][c/\mathbf{0}]\big]\big]\Big]c \Leftrightarrow [z/\mathbf{0}]\bigcup[z/\mathbf{S}(z)]x+z = y$$

from the formula-axiom of natural numbers

$$\mathrm{Ax}\,\mathfrak{A}\mathrm{r}\colon \neg\mathbf{S}(x) = \mathbf{0} \wedge \big(\mathbf{S}(x) = \mathbf{S}(y) \Rightarrow x = y\big) \wedge [x/\mathbf{0}]\bigcup[x/\mathbf{S}(x)](x = y).$$

§ 7. Conditional definitions of functions

Explicit definitions of functions are usually equalities; we shall consider conditional definitions of functions as implications in which the antecedent describes the domain of the function to be defined.

Let K be a program and v a valuation. The resulting valuation $v' = K_R(v)$ is not always defined. We can easily see that the formula $K\mathbf{1}$ describes the domain of the function K_R for any realization R. We now define another formula with that property. We shall define the mapping

$$E\colon \boldsymbol{FS} \cup \boldsymbol{FST} \to \boldsymbol{FSF}$$

as follows:

$E(s) = \mathbf{1}$ for any substitution $s \in S$,

$E\big(\circ[K\,M]\big) = E(K) \wedge K\,E(M)$,

$E\big(\curlyvee[\alpha K\,M]\big) = \alpha \wedge E(K) \vee \neg\alpha \wedge E(M)$,

$E(*[\alpha K]) = \bigcup \varkappa[\alpha K[\ ]]\neg\alpha$,

$E(x) = \mathbf{1}$ for any individual variable $x \in V_i$,

$E(\varphi(\tau_1, \ldots, \tau_n)) = \bigwedge_{i=1}^{n} E(\tau_i)$,

$E(K\tau) = E(K) \wedge K\, E(\tau)$.

The following lemma states that the mapping E gives halting formulas for programs and terms from the ***FST*** set.

7.1. *For every realization R, every valuation v, every program $K \in \boldsymbol{FS}$, and every term $\tau \in \boldsymbol{FST}$, the following conditions are equivalent*

(i) *the resulting valuation $v' = K_R(v)$ is defined,*

(ii) *the formula $E(K)$ is satisfied in the realization R by the valuation v, $E(K)_R(v) = \checkmark$;*

similarly

(t) *the value $\tau_R(v)$ is defined iff $(E(\tau))_R(v) = \checkmark$* ([31]).

Every expression of the form $\beta \Rightarrow (\varphi_\tau(x_1 \ldots x_n) = \tau)$ such that

(1) the expression $\varphi_\tau(x_1 \ldots x_n)$ is an elementary term, the partial-functor φ_τ does not belong to the language $\mathscr{L}$,

(2) the expression τ is a term ($\tau \in \boldsymbol{FST}$),

(3) the expression β is a formula $E(\tau)$ or is semantically equivalent to $E(\tau)$,

(4) the set of free variables of τ is $\{x_1, \ldots, x_n\}$,

will be called the *conditional definition* of a partial-function.

EXAMPLES. 7.1. In the relational system $\langle \mathscr{N}, 0, \mathbf{S}, =, < \rangle$ of natural numbers with the relation $<$ (less than) the formula

$$\neg x < y \Rightarrow \Big\{x \dot{-} y = \varkappa\Big[x < y * [\mathbf{1}[\ ]] \circ \big[[u/y\ z/0] * \big[\neg u = x[u/\mathbf{S}(u)\ z/\mathbf{S}(z)]\big]\big]\Big]z\Big\}$$

is the conditional definition of subtraction $x \dot{-} y$ which is feasible iff $x \geqslant y$.

7.2. *The following formula gives the definition of division x/y in the system $\langle \mathscr{N}, 0, \mathbf{S}, +, -, <, = \rangle$ of nonnegative integers*

$$\neg y = 0 \vee [t/0] \bigcup [t/t+y](x = t) \Rightarrow \Big\{x/y = \varkappa\Big[y = 0 * [x = x[\ ]] \circ \Big[[q/0\ u/x] * \big[\neg u = 0\, \varkappa\big[u < y * [x = x[\ ]][u/u-y\ q/q+1]]\big]\Big]\Big]q\Big\}.$$

§ 8. Substitutions

Let s be a substitution of the form

$$[x_1/\tau_1 \ \ldots\ x_n/\tau_n\ a_1/\alpha_1 \ \ldots\ a_m/\alpha_m];$$

we can conceive this as the diagram of the mapping

$$s\colon V_i \cup V_0 \to \boldsymbol{T} \cup \boldsymbol{F}^o$$

such that

$$s(x_i) = \tau_i \quad \text{for } i = 1, \ldots, n,$$
$$s(a_j) = \alpha_j \quad \text{for } j = 1, \ldots, m,$$
$$s(z) = z \quad \text{for the remaining variables.}$$

From now on we shall make no distinction between substitutions and mappings which satisfy the following conditions:

(1) $s\colon V_i \cup V_0 \to \boldsymbol{T} \cup \boldsymbol{F}^o$,

(2) if $z \in V_i$ then $s(z) \in \boldsymbol{T}$,

(3) if $z \in V_0$ then $s(z) \in \boldsymbol{F}^o$,

(4) the set $\{z \in V_i \cup V_0 \colon s(z) \neq z\}$ is finite.

Let us observe that every mapping s can be extended in a unique way to the set $\boldsymbol{T} \cup \boldsymbol{F}^o$ if we put

$$s(\omega) = \omega\big(z_1/s(z_1), \ldots, z_n/s(z_n)\big)$$

where

(a) $z_1, \ldots, z_n$ are all variables occurring in the expression $\omega \in \boldsymbol{T} \cup \boldsymbol{F}^o$,

(b) $\omega\big(z_1/s(z_1), \ldots, z_n/s(z_n)\big)$ denotes an expression obtained from ω by simultaneous replacement of all occurrences of the variables $z_1, \ldots, z_n$ by corresponding expressions $s(z_1), \ldots, s(z_n)$. We shall use the denotation $\overline{s\omega}$, also.

It can be proved that the resulting expression $\omega\big(z_1/s(z_1), \ldots, z_n/s(z_n)\big)$ is a term (an open formula) provided that $\omega \in \boldsymbol{T}$ ($\omega \in \boldsymbol{F}^o$).

Let s_1, s_2 be two substitutions. We can define another substitution s, letting

$$s(z) = s_1\big(s_2(z)\big) = (s_2 \circ s_1)(z).$$

It is obvious that the substitutions considered with the superposition operation form a semigroup $\boldsymbol{S}$, the identity mapping (or empty substitution) being the unit of the semigroup.

8.1. *For every term τ, every open formula α, every substitution s, every realization R, and for every valuation v the following equalities hold* ([49]):

$$(\overline{s\tau})_R(v) = \tau_R\big(s_R(v)\big),$$
$$(\overline{s\alpha})_R(v) = \alpha_R\big(s_R(v)\big).$$

This lemma leads to the following theorem:

8.2.

$$(s_2 \circ s_1)_R = s_{2R} \circ s_{1R},$$

i.e., the realization is a homomorphism from the semigroup of substitutions into the semigroup of transformations of the set of valuations W into itself.

§ 9. Generalized terms

Generalized terms from the set ***FST*** possess the following useful property:

9.1. *For every generalized term τ there exist a program K and a classical term τ^* such that for every realization R and every valuation v*

$$\tau_R(v) = (K\tau^*)_R(v)$$ ([17]).

EXAMPLE. Let us consider

$$\tau\colon\ *[x \geqslant y\ [x/x-y]]x+\big(\circ[[i/0]*[x \geqslant y\ [x/x-y\ i/i+1]]]i\big)\cdot y.$$

This term is equivalent to the term

$$\circ\Big[[z/x]\circ\Big[*[x \geqslant y\ [x/x-y]]\circ\Big[[t/x\ x/z]\circ[[i/0]*[x \geqslant y\ [x/x-y\ i/i+1]]]\Big]\Big]\Big](t+i\cdot y).$$

Lemma 9.1 implies the following important fact:

9.2. *There exists an effective transformation χ defined for every formula of the form $\varrho(\tau_1 \dots \tau_n)$ and such that*

(1) $\chi\big(\varrho(\tau_1, \dots, \tau_n)\big) = K\varrho(\tau_1^*, \dots, \tau_n^*)$ *where* $\tau_1^*, \dots, \tau_n^*$ *are classical terms from* $\boldsymbol{T}$,

(2) *for every realization R and every valuation v*

$$\big(\varrho(\tau_1, \dots, \tau_n)\big)_R(v) = \big(K\varrho(\tau_1^*, \dots, \tau_n^*)\big)_R(v).$$

Compare this with axiom (T16) in Section 11. ■

§ 10. Semantic consequence operation

Let α be a formula; if it is valid in a realization R, we shall say that the *realization R is a model of α*.

The given realization R is associated with a relational system. We shall say that $\mathfrak{A}$ *is a model of* α if it is valid in $\mathfrak{A}$.

Notation: $\underset{R}{\models} \alpha$ ($\underset{\mathfrak{A}}{\models} \alpha$).

By a *model of the set $\mathscr{A}$ of formulas* we shall understand a realization R such that all formulas $\alpha \in \mathscr{A}$ are valid in R. Notation: $\underset{R}{\models} \mathscr{A}$.

A formula α is a semantic consequence of the set $\mathscr{A}$ of formulas iff every model of $\mathscr{A}$ is also a model for α.

Notation: $\mathscr{A} \models \alpha$.

The set of all formulas that are semantic consequences of the set $\mathscr{A}$ of formulas will be denoted by $Cn(\mathscr{A})$, i.e.

$$Cn(\mathscr{A}) = \{\alpha \in \boldsymbol{FSF}\colon \mathscr{A} \models \alpha\}.$$

The set of all tautologies is equal to $Cn(\emptyset)$.

EXAMPLES.

$$\{K^i\alpha \Rightarrow \beta\}_{i\in\mathcal{N}} \models \big(\bigcup K\alpha \Rightarrow \beta\big)$$

where K is a program, α and β are formulas.

In fact, let R be a model of every formula $(K^i\alpha \Rightarrow \beta)$ $(i \in \mathcal{N})$. By definition, this means that, for every valuation v, $(K^i\alpha \Rightarrow \beta)_R(v) = \bigvee$. Hence, for every valuation v,

$$\big(\bigcup K\alpha \Rightarrow \beta\big)_R(v) = \underset{i\in\mathcal{N}}{\text{l.u.b.}}\ (K^i\alpha)_R(v) \to \beta_R(v) = \underset{i\in\mathcal{N}}{\text{g.l.b.}}\ (K^i\alpha \Rightarrow \beta)_R(v) = \bigvee.$$

Contrary to classical consequence operations, Cn is not finitistic, i.e. it is not true in general that if $\alpha \in Cn(\mathscr{A})$ then $\alpha \in Cn(\mathscr{A}_0)$ for some finite subset $\mathscr{A}_0 \subset \mathscr{A}$. This problem will be examined in Chapters II and III.

§ 11. Algorithmic formalized theories

The set $\boldsymbol{Ax}$ of all logical axioms consists of all formulas of the following forms:

T1. $\big((\alpha \Rightarrow \beta) \Rightarrow ((\beta \Rightarrow \gamma) \Rightarrow (\alpha \Rightarrow \gamma))\big)$,

T2. $(\alpha \Rightarrow (\alpha \vee \beta))$,

T3. $(\beta \Rightarrow (\alpha \vee \beta))$,

T4. $\big((\alpha \Rightarrow \gamma) \Rightarrow \big((\beta \Rightarrow \gamma) \Rightarrow ((\alpha \vee \beta) \Rightarrow \gamma)\big)\big)$,

T5. $((\alpha \wedge \beta) \Rightarrow \beta)$,

T6. $((\alpha \wedge \beta) \Rightarrow \alpha)$,

T7. $\big((\gamma \Rightarrow \alpha) \Rightarrow \big((\gamma \Rightarrow \beta) \Rightarrow (\gamma \Rightarrow (\alpha \wedge \beta))\big)\big)$,

T8. $\big((\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \wedge \beta) \Rightarrow \gamma)\big)$,

T9. $\big(((\alpha \wedge \beta) \Rightarrow \gamma) \Rightarrow (\alpha \Rightarrow (\beta \Rightarrow \gamma))\big)$,

T10. $((\alpha \wedge \neg\alpha) \Rightarrow \beta)$,

T11. $\big((\alpha \Rightarrow (\alpha \wedge \neg\alpha)) \Rightarrow \neg\alpha\big)$,

T12. $(\alpha \vee \neg\alpha)$,

T13. $(\mathbf{1} \wedge \neg\mathbf{0})$,

T14. $(s\delta \Leftrightarrow \overline{s\delta})$,

T15. $\big(K\varrho(\tau_1, \dots, \tau_n) \Leftrightarrow \varrho(K\tau_1, \dots, K\tau_n)\big)$,

T16. $\big(\varrho(\tau_1, \dots, \tau_n) \Leftrightarrow \chi\big(\varrho(\tau_1, \dots, \tau_n)\big)\big)$,

T17. $\big(K(\alpha \vee \beta) \Leftrightarrow (K\alpha \vee K\beta)\big)$,

T18. $\big(K(\alpha \wedge \beta) \Leftrightarrow (K\alpha \wedge K\beta)\big)$,

T19. $(K\neg\alpha \Rightarrow \neg K\alpha)$,

T20. $\big(K\mathbf{1} \Rightarrow (\neg K\alpha \Rightarrow K\neg\alpha)\big)$,

T21. $\big(K(\alpha \Rightarrow \beta) \Rightarrow (K\alpha \Rightarrow K\beta)\big)$,

T22. $\big(K\mathbf{1} \Rightarrow ((K\alpha \Rightarrow K\beta) \Rightarrow K(\alpha \Rightarrow \beta))\big)$,

T23. $\big(M \bigcup K\alpha \Leftrightarrow (M\alpha \vee M \bigcup K(K\alpha))\big)$,

T24. $\big(M \bigcap K\alpha \Leftrightarrow (M\alpha \wedge M \bigcap K(K\alpha))\big)$,

T25. $(\circ[KM]\alpha \Leftrightarrow KM\alpha)$,

T26. $\big(\varkappa[\delta KM]\alpha \Leftrightarrow ((\delta \wedge K\alpha) \vee (\neg\delta \wedge M\alpha))\big)$,

T27. $\big(*[\delta K]\alpha \Leftrightarrow \bigcup \varkappa[\delta K[\ ]](\neg\delta \wedge \alpha)\big)$,

where α, β, γ denote any formulas from ***FSF***, δ an open formula, s a substitution, and K, M programs.

We admit four rules of inference

(r1) $$\frac{\alpha, (\alpha \Rightarrow \beta)}{\beta},$$

(r2) $$\frac{\alpha, K\mathbf{1}}{K\alpha},$$

(r3) $$\frac{\{(\gamma \Rightarrow MK^i\alpha)\}_{i\in\mathcal{N}}}{(\gamma \Rightarrow M\bigcap K\alpha)},$$

(r4) $$\frac{\{(MK^i\alpha \Rightarrow \gamma)\}_{i\in\mathcal{N}}}{(M\bigcup K\alpha \Rightarrow \gamma)}.$$

Let $\mathscr{A}$ be a subset of ***FSF***. By $C(\mathscr{A})$ we denote the set of all syntactic consequences of $\mathscr{A}$, i.e. the set of all theorems derivable from the set $\mathscr{A}\cup\mathrm{Ax}$ by means of the rules (r1)–(r4). The system $\mathscr{T} = \langle \mathscr{L}, \mathscr{C}, \mathscr{A}\rangle$ is called a *formalized algorithmic theory*. The system $\langle \mathscr{L}, \mathscr{C}, \emptyset\rangle$ will be called the *deductive system of algorithmic logic*.

A realization R is said to be a *model for an algorithmic theory* $\mathscr{T} = \langle \mathscr{L}, \mathscr{C}, \mathscr{A}\rangle$ provided every formula in $\mathscr{A}$ is valid in R.

All the theories studied in the sequel are theories with equality, i.e., they contain the sign of equality in their languages and the following axioms for the predicate of equality:

T28. $$x = x,$$

T29. $$(x = y \Rightarrow y = x),$$

T30. $$(x = y \wedge y = z \Rightarrow x = z),$$

for every natural number m and every m-argument functor φ;

T31. $$\big((x_1 = y_1)\wedge \ldots \wedge (x_m = y_m) \Rightarrow (\varphi(x_1, \ldots, x_n) = \varphi(y_1, \ldots, y_m))\big),$$

for every natural number m and every m-argument predicate;

T32. $$\big((x_1 = y_1)\wedge \ldots \wedge (x_m = y_m) \Rightarrow \varrho(x_1, \ldots, x_m) \Leftrightarrow \varrho(y_1, \ldots, y_m)\big),$$

where $x, y, z, x_1, \ldots, x_m, y_1, \ldots, y_m$ are individual variables.

EXAMPLES. 1. The algorithmic theory of natural numbers is a theory with the following specific axiom:

$$\mathrm{Ax}\mathfrak{N}\mathfrak{r}\colon \mathrm{S}x = 0 \wedge (\mathrm{S}x = \mathrm{S}y \Rightarrow x = y) \wedge [y/0]\bigcup[y/y+1](x = y).$$

This theory characterizes categorically the standard model of classical arithmetic.

2. The algorithmic theory of fields of characteristic zero is the theory containing the axioms of a field and the axiom 2 from § 4.

3. The algorithmic theory of Archimedean ordered fields is a theory containing the axioms of an ordered field and axiom 3 from § 4.

§ 12. Completeness theorem

12.1. *$C(\mathscr{A}) = Cn(\mathscr{A})$ for every $\mathscr{A} \subset$ **FSF**.*

Theorem 12.1 shows that algorithmic logic constitutes an appropriate basis for proving properties of programs. Not only does it allow us to carry out correct proofs but also if an algorithmic fact is valid then a proof of it exists.

As a consequence we obtain:

12.2. *Formula α is a tautology iff $\alpha \in C(\emptyset)$* ([17]).

An algorithmic theory $\mathscr{T}$ is said to be consistent if there is no formula α with the property that both α and $\neg\alpha$ are theorems of $\mathscr{T}$.

12.3. *An algorithmic theory $\mathscr{T}$ is consistent iff it possesses a model* ([17]).

§ 13. An example of proving a property of a program

We shall prove here that addition is a strongly programmable function in the relational system $\mathfrak{N} = \langle \mathcal{N}, 0, \mathrm{S}, =\rangle$.

Let us assume

(d) $$x+y \overset{\mathrm{df}}{=} \circ\big[[t/x\ u/0] * [\neg u = y\ \ [t/\mathrm{S}(t)\ \ u/\mathrm{S}(u)]]\big]t.$$

The program used here will be denoted by K.

We are going to prove that

(a) $\models \mathrm{Ax}\mathfrak{N}\mathfrak{r} \Rightarrow K\mathbf{1}$,

(b) $\models \mathrm{Ax}\mathfrak{N}\mathfrak{r} \wedge d \Rightarrow (x+0 = x)$,

(c) $\models \mathrm{Ax}\mathfrak{N}\mathfrak{r} \wedge d \Rightarrow \big(x+\mathrm{S}(y) = \mathrm{S}(x+y)\big)$.

Ad (a). For every natural i the formula $\mathrm{S}^i(0) = y \Rightarrow \mathrm{S}^i(0) = y$ is a tautology: $\models \mathrm{S}^i(0) = y \Rightarrow \mathrm{S}^i(0) = y$.

Making use of axioms (T14), (T26) and of axioms (T1)–(T12) we can prove that, for every natural i,

$$\models [u/0][u/\mathrm{S}(u)]^i u = y \Rightarrow [t/x\ \ u/0]\preceq[\neg u = y\ \ [t/\mathrm{S}(t)\ u/\mathrm{S}(u)][\ ]]^i u = y.$$

Simple induction based on (T23) leads to the following conclusion: for every natural i

$$\models [u/0][u/\mathrm{S}(u)]^i u = y \Rightarrow [t/x\ u/0]\bigcup\preceq[\neg u = y\ \ [t/\mathrm{S}(t)\ u/\mathrm{S}(u)][\ ]]u = y.$$

By (T27) and (T25) we obtain for every natural i

$$\models [u/0][u/\mathrm{S}(u)]^i u = y \Rightarrow \circ\big[[t/x\ u/0] * [\neg u = y\ \ [t/\mathrm{S}(t)\ u/\mathrm{S}(u)]]\big]\mathbf{1}.$$

Now, by r4 we obtain

$$\models [u/0]\bigcup[u/\mathrm{S}(u)]u = y \Rightarrow \circ\big[[t/x\ u/0] * [\neg u = y\ \ [t/\mathrm{S}(t)\ u/\mathrm{S}(u)]]\big]\mathbf{1},$$

i.e. the program K stops when interpreted in natural numbers.

Ad (b). This is a simple consequence of (T26).

Ad (c). In proving (c) we shall make use of two more general lemmas (the proofs are left to the reader).

13.1. *Let K and M be programs written in the language of arithmetic. Let the following formulas*

$$K(u = 0),$$

$$(u = v) \Rightarrow M\big(u = \mathbf{S}(v)\big)$$

be valid in natural numbers. Then the following programs are equivalent:

$$\circ\big[K * [\neg u = \mathbf{S}(y)\ M]\big] \quad \text{and} \quad \circ\big[K \circ [* [\neg u = y\ M] M]\big].$$

We can state this also as follows: for every formula α the following formula is valid

$$\circ\big[K * [\neg u = \mathbf{S}(y)\ M]\big]\alpha \Leftrightarrow \circ\big[K \circ [* [\neg u = y\ M] M]\big]\alpha.$$

The next lemma used in the proof is as follows:

13.2. *For every program K, for every 1-argument functor φ, for every realizations R and for every valuation v the following equality holds*

$$\big(K\varphi(\tau)\big)_R(v) = \big(\varphi(K\tau)\big)_R(v).$$

Indeed,

$$\begin{aligned}\big(K\varphi(\tau)\big)_R(v) &= \varphi(\tau)_R\big(K_R(v)\big) = \varphi_R\big(\tau_R\big(K_R(v)\big)\big)\\ &= \varphi_R\big((K\tau)_R(v)\big) = \big(\varphi(K\tau)\big)_R(v).\end{aligned}$$

Making use of these facts we proceed as follows:

$$x+\mathbf{S}(y) = \circ\big[[t/x\ u/0] * [\neg u = \mathbf{S}(y)[t/\mathbf{S}(t)\ u/\mathbf{S}(u)]]\big]t$$

from the definition of $+$

$$= \circ\big[\circ\big[[t/x\ u/0] * [\neg u = y[t/\mathbf{S}(t)\ u/\mathbf{S}(u)]]\big][t/\mathbf{S}(t)\ u/\mathbf{S}(u)]\big]t$$

from lemma 13.1

$$= \circ\big[[t/x\ u/0] * [\neg u = y[t/\mathbf{S}(t)\ u/\mathbf{S}(u)]]\big]\mathbf{S}(t)$$

from (T14)

$$= \mathbf{S}\big(\big[\circ\,[t/x\ u/0] * [\neg u = y[t/\mathbf{S}(t)\ u/\mathbf{S}(u)]]\big]t\big)$$

from lemma 13.2

$$= \mathbf{S}(x+y).$$

from the definition of $+$.

This simple example shows certain tools that can be used in the proofs of properties of programs.

Chapter II

METAMATHEMATICAL INVESTIGATIONS IN ALGORITHMIC LOGIC

§ 1. Properties of the semantic consequence operation

In this section we shall investigate the semantic consequence operation Cn as defined in I, § 10.

Let $\mathscr{L}$ be an established algorithmic language.

1.1. *For every set $Z \subset \boldsymbol{FSF}$ and every formula $\alpha \in \boldsymbol{FSF}$, if $Z \models \alpha$ then the set $Z \cup \{\neg\alpha\}$ has no model* [17].

In the case where the formula α is closed, i.e., where for every realization of the language $\mathscr{L}$ the value of the formula α does not depend on the adopted valuation, the above lemma can be strenghthened:

$Z \models \alpha$ if and only if the set $Z \cup \{\neg\alpha\}$ has no model.

1.2. *For every set $Z \subset \boldsymbol{FSF}$ and any formulas $\alpha, \beta \in \boldsymbol{FSF}$,*

$$\text{if } Z \models (\alpha \Rightarrow \beta) \text{ then } Z \cup \{\alpha\} \models \beta \text{ [17].}$$

Let us observe that the inverse theorem is, in general, not true. Consider the following example: $Z = \emptyset$ and $\beta = (s\alpha)$. Certainly, $\alpha \models \beta$ but if α and $\neg\alpha$ are not tautologies then there exist a realization R, a valuation v and a substitution s such that $\alpha_R(v) = \mid$ and $\alpha_R\big(s_R(v)\big) = /$.

1.3. *The consequence operation Cn does not have the following property: if $Z \models \alpha$ then there exists a finite subset Z_0 of the set Z such that $Z_0 \models \alpha$.*

To prove 1.3 we shall give an example of the set Z and formula α such that $Z \models \alpha$ but for every finite set $Z_0 \subset Z$ there exists a model for Z_0 that is not a model for the formula α.

Let

$$Z = \big\{\big([x/0]([x/\mathbf{S}x]^i\ 0 \leqslant x)\big)\big\}_{i \in \mathscr{N}}, \qquad \alpha = [x/0] \cap [x/\mathbf{S}x]\ \ 0 \leqslant x,$$

where 0 is a constant ($0 \in \Phi_0$), $\mathbf{S}$ is a one-argument operation and $0 \leqslant$ is a one-argument relation.

Let R be a model for the set Z. Then we have $([x/0] \cap [x/\mathbf{S}x]\ 0 \leqslant x)_R(v) = \mid$ for every valuation v. Thus R is a model for the formula α, and $Z \models \alpha$. Let us consider any finite subset Z_0 of the set Z,

$$Z_0 = \{[x/0]([x/\mathbf{S}x]\ 0 \leqslant x)\}_{i \in I}$$

where I is any finite sequence of natural numbers.

Now we define a realization $\bar{R}$ in the set of natural numbers $\mathscr{N}$ as follows: the constant 0 is a zero in the set $\mathscr{N}$, the operation $\mathbf{S}$ is a consequent in $\mathscr{N}$, the relation $0 \leqslant$ is the characteristic function of the set I, i.e.,

$$0 \leqslant n = \begin{cases} \mid & \text{if} \quad n \in I, \\ / & \text{if} \quad n \notin I. \end{cases}$$

The realization $\bar{R}$ defined in such a way is a model for the set Z_0 since for every $i \in I$ and for every v

$$\big([x/0]([x/\mathbf{S}x]^i\ 0 \leqslant x)\big)_{\bar{R}}(v) = 0 \leqslant \mathbf{S}^i 0 = \mid.$$

Nevertheless, if $i \notin I$ then the formula

$$\big([x/0]([x/\mathbf{S}x]^i\ 0 \leqslant x)\big)$$

has value $/$ for every valuation v in the realization $\bar{R}$. So $\alpha_{\bar{R}}(v) = /$.

We can prove an analogue of the downward Skolem–Löwenheim theorem.

1.4. *If the theory* $\mathscr{T} = \langle \mathscr{L}, \mathscr{C}, \mathscr{A} \rangle$ *has an infinite model then it has a denumerable model.*

Let us observe that the upward theorem of Skolem–Löwenheim is not true in the class of all ordinary semantic models. One can prove that all models of the theory of arithmetic with the axioms

$$\neg(\mathrm{S}x = 0), \quad [x/0] \bigcup [x/\mathrm{S}x]\, x = y, \quad (\mathrm{S}x = \mathrm{S}y \Rightarrow x = y)$$

are isomorphic with the standard model $\langle \mathscr{N}, \mathrm{S}, 0, = \rangle$ of arithmetic.

§ 2. Diagrams of formulas

In this section we shall consider another formalization of the set of tautologies of the algorithmic language $\mathscr{L}$. We shall follow Gentzen's ideas. We first recall some auxiliary notions.

By a *tree* we shall understand a set D of finite sequences of natural numbers such that if any sequence $c = (i_1, \ldots, i_n)$ is an element of D then every initial segment c_k of c, $c_k = (i_1, \ldots, i_k)$, $k \leqslant n$, is an element of the tree D also. The empty sequence of natural numbers, denoted by $\emptyset$, belongs to every tree.

For any element $c = (i_1, \ldots, i_n)$ of the tree D the number n is called the *level of the element* c *in* D.

By the *level of* D we shall mean the set of all elements that have the same level.

A subset of D such that its elements are linearly ordered with respect to the relation "to be an initial segment" is called a *branch of the tree* D.

Let Γ_1, Γ_2 denote finite sequences (the empty sequence is admitted) of formulas in $\mathscr{L}$. Every expression of the form $\Gamma_1 \to \Gamma_2$ will be called a *sequent*.

The sequent S of the form

$$\alpha_1, \ldots, \alpha_n \to \beta_1, \ldots, \beta_m$$

is called *indecomposable* if and only if every formula α_i, β_j $(i = 1, \ldots, n, j = 1, \ldots, m)$ is an atomic formula.

A sequent S is said to be an *axiom* if and only if there exist indices i and j $(1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m)$ such that α_i and β_j are identical or $\mathbf{1} \in \{\beta_1, \ldots, \beta_m\}$ or $\mathbf{0} \in \{\alpha_1, \ldots, \alpha_n\}$.

By a *scheme* we shall understand a pair $\{S, S_0\}$ of sequents, a triple $\{S, S_0; S_1\}$ of sequents or an enumerable sequence $\{S, S_0; S_1; S_2; \ldots\}$ of sequents, which will be written in the form

$$\frac{S}{S_0} \quad \text{or} \quad \frac{S}{S_0;\ S_1} \quad \text{or} \quad \frac{S}{\{S_i\}_{i \in \mathscr{N}}}.$$

Sequent S is called the *conclusion*, and sequents: S_0 in the first case, S_0, S_1 in the second and $S_0, S_1, \ldots$ in the third case—the *premises*.

In the sequel we shall consider three groups of schemes:

(1A) $$\frac{\Gamma_1, s_1 \ldots s_k\alpha_0, \Gamma_2 \to \Gamma_3}{s_1 \ldots s_{k-1}\overline{s_k\alpha_0}, \Gamma_1, \Gamma_2 \to \Gamma_3},$$

(1B) $$\frac{\Gamma_1 \to \Gamma_2, s_1 \ldots s_k\alpha_0, \Gamma_3}{\Gamma_1 \to s_1 \ldots s_{k-1}\overline{s_k\alpha_0}, \Gamma_2, \Gamma_3},$$

where α_0 is an atomic formula and $k \in \mathscr{N}$;

(2A) $$\frac{\Gamma_1, s\varrho(\tau_1, \ldots, \tau_n), \Gamma_2 \to \Gamma_3}{s\chi(\varrho(\tau_1, \ldots, \tau_n)), \Gamma_1, \Gamma_2 \to \Gamma_3},$$

(2B) $$\frac{\Gamma_1 \to \Gamma_2, s\varrho(\tau_1, \ldots, \tau_n), \Gamma_3}{\Gamma_1 \to s\chi(\varrho(\tau_1, \ldots, \tau_n)), \Gamma_2, \Gamma_3},$$

(3A) $$\frac{\Gamma_1, s\neg\alpha, \Gamma_2 \to \Gamma_3}{\Gamma_1, \Gamma_2 \to s\alpha, \Gamma_3},$$

(3B) $$\frac{\Gamma_1 \to \Gamma_2, s\neg\alpha, \Gamma_3}{\Gamma_1, s\alpha \to \Gamma_2, \Gamma_3},$$

(4A) $$\frac{\Gamma_1, s(\alpha \wedge \beta), \Gamma_2 \to \Gamma_3}{s\alpha, s\beta, \Gamma_1, \Gamma_2 \to \Gamma_3},$$

(5B) $$\frac{\Gamma_1 \to \Gamma_2, s(\alpha \vee \beta), \Gamma_3}{\Gamma_1 \to s\alpha, s\beta, \Gamma_2, \Gamma_3},$$

(7A) $$\frac{\Gamma_1, s\bigcap K\alpha, \Gamma_2 \to \Gamma_3}{s\bigcap K(K\alpha), \Gamma_1, s\alpha, \Gamma_2 \to \Gamma_3},$$

(6B) $$\frac{\Gamma_1 \to \Gamma_2, s(\alpha \Rightarrow \beta), \Gamma_3}{s\alpha, \Gamma_1 \to s\beta, \Gamma_2, \Gamma_3},$$

(9A) $$\frac{\Gamma_1, s \circ [KM]\alpha, \Gamma_2 \to \Gamma_3}{sKM\alpha, \Gamma_1, \Gamma_2 \to \Gamma_3},$$

(8B) $$\frac{\Gamma_1 \to \Gamma_2, s\bigcup K\alpha, \Gamma_3}{\Gamma_1 \to s\bigcup K(K\alpha), \Gamma_2, s\alpha, \Gamma_3},$$

(10A) $$\frac{\Gamma_1, s\curlyvee[\delta KM]\alpha, \Gamma_2 \to \Gamma_3}{s((\delta \wedge K\alpha) \vee (\neg\delta \wedge M\alpha)), \Gamma_1, \Gamma_2 \to \Gamma_3},$$

(9B) $$\frac{\Gamma_1 \to \Gamma_2, s \circ [KM]\alpha, \Gamma_3}{\Gamma_1 \to sKM\alpha, \Gamma_2, \Gamma_3},$$

(11A) $$\frac{\Gamma_1, s*[\delta K]\alpha, \Gamma_2 \to \Gamma_3}{s\bigcup\curlyvee[\delta K[\,]](\neg\delta \wedge \alpha), \Gamma_1, \Gamma_2 \to \Gamma_3},$$

(10B) $$\frac{\Gamma_1 \to \Gamma_2, s\curlyvee[\delta KM]\alpha, \Gamma_3}{\Gamma_1 \to s((\delta \wedge K\alpha) \vee (\neg\delta \wedge M\alpha)), \Gamma_2, \Gamma_3},$$

(11B) $$\frac{\Gamma_1 \to \Gamma_2, s*[\delta K]\alpha, \Gamma_3}{\Gamma_1 \to s\bigcup\curlyvee[\delta K[\,]](\neg\delta \wedge \alpha), \Gamma_2, \Gamma_3}.$$

The second group

(5A) $$\frac{\Gamma_1, s(\alpha \vee \beta), \Gamma_2 \to \Gamma_3}{s\alpha, \Gamma_1, \Gamma_2 \to \Gamma_3;\ s\beta, \Gamma_1, \Gamma_2 \to \Gamma_3},$$

(4B) $$\frac{\Gamma_1 \to \Gamma_2, s(\alpha \wedge \beta), \Gamma_3}{\Gamma_1 \to s\alpha, \Gamma_2, \Gamma_3; \Gamma_1 \to s\beta, \Gamma_2, \Gamma_3},$$

(6A) $$\frac{\Gamma_1, s(\alpha \Rightarrow \beta), \Gamma_2 \to \Gamma_3}{\Gamma_1, \Gamma_2 \to s\alpha, \Gamma_3; s\beta, \Gamma_1, \Gamma_2 \to \Gamma_3}.$$

The third group

(8A) $$\frac{\Gamma_1, s\bigcup K\alpha, \Gamma_2 \to \Gamma_3}{\{sK^i\alpha, \Gamma_1, \Gamma_2 \to \Gamma_3\}_{i \in \mathcal{N}}},$$

(7B) $$\frac{\Gamma_1 \to \Gamma_2, s\bigcap K\alpha, \Gamma_3}{\{\Gamma_1 \to sK^i\alpha, \Gamma_2, \Gamma_3\}_{i \in \mathcal{N}}}.$$

In all the above schemes $\Gamma_1, \Gamma_2, \Gamma_3$ denote any sequents, s denotes any sequence of substitutions, K, M—any programs, and α, β, δ any formulas $\alpha, \beta \in \boldsymbol{FSF}$, $\delta \in \boldsymbol{F^o}$.

By the *diagram of a formula* α_0 we shall mean an ordered pair (D, d) where D is a tree and d is a mapping which to every element of the tree D assigns a certain non-empty sequent. The tree D and the mapping d are defined by induction on the level l of the tree D as follows:

1. If $l = 0$ then the only element of this level is $\emptyset$ and $d(\emptyset)$ is equal to the sequent $\to \alpha_0$.

Suppose that we have defined all the elements of the tree D with the level not higher than n. Now we define the elements of the level $n+1$ of D. Let $c = (i_1, \ldots, i_n) \in D$ and let the sequent $d(c)$ be defined:

2. If $d(c)$ is an indecomposable sequent or an axiom, then none of the elements $c = (i_1, \ldots, i_n, k)$, $k \in \mathcal{N}$ belongs to D, and c and $d(c)$ are called an *end-element* and an *end-sequent* of the tree D.

3. The sequent $d(c)$: $\Gamma \to \nabla$ is neither indecomposable nor an axiom. We shall consider two cases

Case 1: *n is an even number.*

A. If the sequence ∇ contains only atomic formulas, then $(i_1, \ldots, i_n, 0) \in D$ and $d(i_1, \ldots, i_n, 0) = d(c)$.

B. If α is the first right side non-atomic formula in ∇, then we consider different forms of the formula α:

1. if the sequent $d(c)$ is the conclusion in a scheme of the group IB concerning the formula α, then $(i_1, \ldots, i_n, 0) \in D$ and $d(i_1, \ldots, i_n, 0)$ is equal to the only premise in that scheme,

2. if the sequent $d(c)$ is the conclusion in a scheme of the group IIB, then $(i_1, \ldots, i_n, 0)$ and $(i_1, \ldots, i_n, 1)$ belong to D and $d(i_1, \ldots, i_n, 0)$, $d(i_1, \ldots, i_n, 1)$ are the first and the second premise in that scheme,

3. if the sequent $d(c)$ is the conclusion in a scheme of the group IIIB, then $(i_1, \ldots, i_n, k)$ are in D for every $k \in \mathcal{N}$ and $d(i_1, \ldots, i_n, k)$ is the kth premise in that scheme.

Case 2: *n is an odd number.*

Points A and B in the above definition must be changed in the following way: the sequence ∇ is replaced by Γ and groups I, II, IIIB by I, II, IIIA.

From this definition it immediately follows that for every formula its diagram is defined in an unambiguous way.

Let S be a sequent $\alpha_1 \ldots \alpha_n \to \beta_1 \ldots \beta_n$; then by δ_S we shall denote the formula $\big((\alpha_1 \wedge (\alpha_2 \wedge \ldots \wedge \alpha_n) \ldots) \Rightarrow (\beta_1 \vee (\beta_2 \vee \ldots \beta_m) \ldots)\big)$.

2.1. *For every realization R of the language $\mathscr{L}$ and for every valuation v the following conditions hold:*

1. *if $\{S, S_0\}$ is a scheme of inference belonging to the first group, then* $\delta_{SR}(v) = \delta_{S_0R}(v)$;

2. *if $\{S, S_0, S_1\}$ is a scheme of inference belonging to group* II, *then* $\delta_{SR}(v) = \delta_{S_0R}(v) \cap \delta_{S_1R}(v)$;

3. *if $\{S, S_0, S_1, \ldots\}$ is a scheme of group* III, *then* $\delta_{SR}(v) = \text{g.l.b.}\,\{\delta_{S_iR}(v)\}_{i \in \mathcal{N}}$.

The diagram of a formula α is said to be *finite* if and only if its every branch is a finite set.

2.2. *A formula α_0 is a tautology if and only if its diagram is finite and every end sequent is an axiom.*

§ 3. Inessentiality of definitions

Let $\mathscr{L}$ be an arbitrary fixed algorithmic language. Let $Z_1 \subset \boldsymbol{FSF}$ be a set of formulas and let $Z_2 \subset \boldsymbol{FST}$ be a set of terms. By $FV(\alpha)$ or $FV(\tau)$ we shall denote the set of all free variables that occur in formula α or in the term τ, respectively. The precise definition of FV can be found in [28], here we limit ourselves to the assertion that a variable z is free in α (τ) if for any realization R and any valuation v the value $\alpha_R(v)$ ($\tau_R(v)$) of expression depends essentially on the value $v(z)$ of the variable z.

We assume that the free variables of the formulas of Z_1 and of the terms of Z_2 are all individual.

With every formula $\alpha \in Z_1$ we associate a predicate ϱ_α not belonging to $\mathscr{L}$; the number of arguments of ϱ_α is equal to the number of free variables in α. All predicates ϱ_α are different. For every term $\tau \in Z_2$ we introduce a functor φ_τ not belonging to $\mathscr{L}$; the number of arguments of φ_τ is equal to the number of free variables in τ.

Let $\mathscr{L}'$ be an extension of the language $\mathscr{L}$ obtained by adding new predicates $\{\varrho_\alpha\}_{\alpha \in Z_1}$ and new partial-functors $\{\varphi_\tau\}_{\tau \in Z_2}$. Let $\mathscr{T} = \langle \mathscr{L}, \mathscr{C}, \mathscr{A} \rangle$ be a theory with

the set $\mathscr{A}$ of specific axioms. Let $\mathscr{A}'$ be the set of formulas composed of all the formulas from $\mathscr{A}$ and of all formulas of one of the following forms:

(1) $(\varrho_\alpha(x_1, \ldots, x_n) \Leftrightarrow \alpha)$ where $FV(\alpha) = \{x_1, \ldots, x_n\}$,

(2) $(E(\tau) \Rightarrow \varphi_\tau(x_1, \ldots, x_n) = \tau)$ where $FV(\tau) = \{x_1, \ldots, x_n\}$.

$E(\tau)$ is a formula with the following property: for any R and v, $E(\tau)_R(v) = |\!/$ if and only if $\tau_R(v)$ is defined, see I.7.

3.1. *For every realization R which is a model for the theory $\mathscr{T}$ there exists an extension R' of R such that R is a model for the theory $\mathscr{T}' = \langle \mathscr{L}', \mathscr{C}, \mathscr{A}' \rangle$* [31].

An introduction of new predicates and partial-functors is equivalent to passing to an extension of a given theory. By admitting definitions of new predicates and functors we cannot prove anything new about the predicates and functors in the already existing theory, as is stated by the following theorem:

3.2. *Let $\mathscr{T}' = \langle \mathscr{L}', \mathscr{C}, \mathscr{A}' \rangle$ be an extension of the consistent theory $\mathscr{T} = \langle \mathscr{L}, \mathscr{C}, \mathscr{A} \rangle$ by assuming definitions* (1) *and* (2). *Then the theory $\mathscr{T}'$ is an inessential extension of $\mathscr{T}$, i.e., $\mathscr{C}(\mathscr{A}) = \mathscr{C}(\mathscr{A}') \cap \boldsymbol{FSF}$.*

§ 4. An analogue of the Herbrand theorem in algorithmic logic

In this part of our considerations we present a theorem of algorithmic logic that is analogous to the Herbrand theorem in classical logic. In spite of the fact that the theorem refers only to some narrow class of formulas, it can be applied to the solving of some decidability problems in algorithmic logic and in the programs theory based on algorithmic logic.

Let us consider the algorithmic logic $\langle \mathscr{L}, \mathscr{C} \rangle$ where $\mathscr{L}$ is a fixed algorithmic language with equality and the consequence operation $\mathscr{C}$ includes axioms of equality. Let n be any natural number, let K_i, M_i $(0 \leqslant i \leqslant n)$ be programs in which the sign $*$ does not appear and let α belong to the set of open formulas.

Under these assumptions we can formulate the following theorem.

4.1. *Any formula β of the form $M_0 \bigcup K_0 \ldots M_n \bigcup K_n \alpha$ is a tautology of algorithmic logic if and only if there exist natural numbers $m_0, \ldots, m_n$ such that the formula $M_0 \bigvee_{i=0}^{m_0} K_0^i \ldots M_n \bigvee_{j=0}^{m_n} K_n^j \alpha$ is a tautology of algorithmic logic* [17].

Let us denote by $\mathscr{K}$ the class of all formulas of the form $M_0 \bigcup K_0 \ldots M_n \bigcup K_n \alpha$ where K_i, M_i $(i = 0, 1, \ldots, n)$ are as above. From theorem 4.1 it follows that the subset of tautologies in $\mathscr{K}$ is recursively enumerable. Indeed, let β be a formula of class $\mathscr{K}$. By 4.1, β is a tautology if and only if there exists a formula in which signs of quantifiers do not appear and which is a tautology of algorithmic logic. Now observe that the following lemma holds:

4.2. *For every formula α without the symbols $\bigcup, \bigcap, *$ we can find in an effective way an open formula α_0 such that the formulas $(\alpha \Rightarrow \alpha_0)$ and $(\alpha_0 \Rightarrow \alpha)$ are both theorems in algorithmic logic* [17].

Hence for the formula β we can find an open formula β_0 such that $\models \beta$ if and only if $\models \beta_0$. Now, in a finite number of steps we can check whether β_0 is a substitution in a tautology of the propositional calculus.

§ 5. Algorithmic logic with classical quantifiers

In some applications we need an algorithmic logic extended by classical quantifiers $\exists$ and $\forall$. That logic will be called *extended algorithmic logic* (see [5]). In order to preserve the completeness property it is sufficient to add the following four logical axioms:

$$\left.\begin{array}{l} (s(\exists x\alpha) \Leftrightarrow \exists y(s([x/y]\alpha))) \\ (s(\forall x\alpha) \Leftrightarrow \forall y(s([x/y]\alpha))) \end{array}\right\} \text{where } y \text{ is an individual variable not occurring in } s\alpha;$$

$$([x/\tau]\alpha \Rightarrow \exists x\alpha),$$

$$(\forall x\alpha \Rightarrow [x/\tau]\alpha),$$

and the following two rules of inference:

$$\frac{[x/y]\alpha \Rightarrow \beta}{\exists x\alpha \Rightarrow \beta} \qquad \frac{\beta \Rightarrow [x/y]\alpha}{\beta \Rightarrow \forall x\alpha}$$

where y is an individual variable occurring neither in α nor in β.

Remark. On the grounds of Gentzen style formalization we need the following rules of inference concerning the universal quantifier

$$\frac{\Gamma \to s[x/y]\alpha, \Delta}{\Gamma \to \Delta, s(\forall x\alpha)} \qquad \frac{s(\forall x\alpha), \Gamma, s[x/\tau]\alpha \to \Delta}{\Gamma, s(\forall x\alpha) \to \Delta}$$

where y is an individual variable not occurring in the conclusion; for $\tau \in \boldsymbol{T}$.

In the above rules s denotes a finite sequence of substitutions (the empty one included).

The extension by classical quantifiers is essential. Let us consider the relational system $\mathfrak{U} = \langle J, P, = \rangle$ where $J = \{1, 2, 3\}$ and P is a binary relation defined as follows: $P(x, y)$ iff $x = 1$ and $y = 2$. In this system formula $\exists y\, P(x, y)$ cannot be equivalently reduced to a formula from the set $\boldsymbol{FSF}$.

In extended algorithmic logic every closed formula can be reduced to its prenex normal form $Q_1 Q_2 \ldots Q_n \alpha$, where α is an open formula and, for each $i = 1, \ldots, n$, Q_i either is a classical quantifier binding an individual variable or is of the form $s \bigcup K$ or $s \bigcap K$ where s is a substitution and K is a loop-free program. Thus, it is possible to classify properties of programs by means of a configuration of quantifiers $Q_1 Q_2 \ldots Q_n$ appearing in the prenex form of the formula expressing

the given property. For example, the formula $(\forall\vec{x}(\alpha \Rightarrow K\beta))$ of correctness has the prenex form: $\forall\vec{x}[a/\mathbf{1}] \bigcup M\gamma$ where a is a propositional variable and M is a loop-free program.

In Chapter IV we shall consider the following properties of a program K. Namely,

the strongest verifiable consequent of a formula (introduced by Floyd [33])— αK: $\exists\vec{y}\,(\alpha(\vec{y}) \wedge K(\vec{y})(\vec{y} = \vec{x}))$ where $\vec{x}$ is the sequence of all different variables occurring in $K\alpha$ and $\vec{y}$ is a copy of $\vec{x}$—$(\alpha K)_R(v) = \checkmark$ means that v is an output data of K for some initial data satisfying α;

iteration of the strongest verifiable consequent of a formula α— $\bigcup \alpha K$: $\exists\vec{y}\,(\alpha(\vec{y}) \wedge \bigcup K(\vec{y})(\vec{y} = \vec{x}))$—$(\bigcup \alpha K)_R(v) = \checkmark$ means that v is an output data of an iteration K^i for some initial data satisfying α;

adequacy with respect to an input formula α and an output formula β— $((\alpha \Leftrightarrow \beta) \wedge (\beta \Leftrightarrow \alpha K))$—$((\alpha \Leftrightarrow K\beta) \wedge (\beta \Leftrightarrow \alpha K))_R(v) = \checkmark$ means that

(1) the input v satisfies α iff K converges and the output satisfies β,

(2) the valuation v satisfies β iff v is the output for some initial data satisfying α.

The equality $(\bigcup \alpha K)_R(v) = \underset{i\in\mathcal{N}}{\text{l.u.b.}}(\alpha K^i)_R(v)$ shows the interconnection between the two constructions.

The strongest verifiable consequent cannot be defined without classical quantifiers. This follows from the undefinability of the existential quantifier and the definablity of the existential quantifier by means of the strongest verifiable consequent, i.e., $\exists x\alpha \Leftrightarrow [x/y](\alpha[x/y])$ is a tautology provided y and x are distinct individual variables.

Chapter III

EFFECTIVITY PROBLEMS OF ALGORITHMIC LOGIC

In this chapter we shall examine various algorithmic properties from the point of view of recursion theory. The first stage of our investigations will show that such natural notions as for example the strong and weak equivalence of programs, the correctness of a program, the halting problem for different classes of examined models, lie at the bottom of arithmetical hierarchy: strictly speaking in the class Π_2^0.

The last-mentioned theorem establishes that the whole elementary theory of programming has the same degree of unsolvability as the notion of truth in the first order arithmetic. Hence, there are elementary properties of programs on an arbitrarily high level of arithmetical hierarchy. In order to obtain these results we must introduce in the Gentzen formalization of algorithmic logic axioms for equality. We devote the first section of this chapter to this aim.

§ 1. Gentzen style formalization with equality

We start from the following definitions:

DEFINITION 1. Let X be a set of equations $t_i = t_j$ where t_i, t_j are terms. We shall say that terms t, u are X *equivalent* ($t = u[X]$) iff there exists a sequence of terms $t_1, \ldots, t_n$ such that t_1 is t, t_n is u and for $i \leqslant n$ either t_i is t_{i+1} or one of the equations $t_i = t_{i+1}$ and $t_{i+1} = t_i$ belongs to X.

DEFINITION 2. Let X be the same set as in Definition 1. We shall denote by $t \asymp u[X]$ the *closure* of relations $t = u[X]$ with respect to the extensionality of functional symbols, i.e.,

$$\text{if } t_i \asymp u_i[X],\ i \leqslant n, \text{ then } f(t_1, \ldots, t_n) \asymp f(u_1, \ldots, u_n)[X].$$

The relation $\asymp$ is of course a congruence in the algebra of terms.

The rules of inference of this system with equality are exactly the same as in Chapter II, § 5 for the system without it. But to the set of axioms we must add the following:

(1) *$\Gamma \to \Delta$ if for any term t the formula $t = t$ appears in Δ;*

(2) *$\Gamma \to \Delta$ if there exist a set of equations X in Γ and a set of terms t_i; u_i, $i \leqslant n$, such that $t_i \asymp u_i[X]$ for $i \leqslant n$ and for a certain predicate r, $r(t_1, \ldots, t_n) \in \Gamma$ and $r(u_1, \ldots, u_n) \in \Delta$.*

The schema of axioms of type (2) ensures the extensionality of relational symbols.

1.1. (*Completeness theorem for Gentzen axiomatization with equality*). *For every formula α of algorithmic logic with equality $\models \alpha$ iff $\vdash \to \alpha$.*

Outline of the proof. The implication from right to left is obvious. To show the converse, let us consider the case where the diagram of a sequent $\to \alpha$ has an infinite path. From this path we build a model in which formula α is not valid (for comparison see [43]). The universe of this model is a set of equivalence classes of the relation $\asymp$ defined in Definition 2, where X is a set of equalities lying on the antecedent of this path. Thus, an element of the model is a class

$$[t] = \{u\colon t \asymp u[X]\}.$$

Functional symbols are realized as corresponding classes:

$$f_R([t_1], \ldots, [t_n]) = [f(t_1, \ldots, t_n)].$$

Relational symbols are realized in the following way:

$$\varphi_R([t_1], \ldots, [t_n]) = \begin{cases} \text{true} & \text{if there exists a } t_i' \in [t_i] \text{ for every } i \leqslant n \text{ such that} \\ & \varphi(t_1', \ldots, t_n') \text{ belongs to the antecedent;} \\ \text{false} & \text{otherwise.} \end{cases}$$

The correctness of this realization for functional symbols follows from the extensionality of relation $\asymp$ and for relational symbols from the axioms of equality.

Let v be the following valuation:

$v(x) = [x]$ for individual variables,

$v(p) =$ true if p occurs in the antecedent, otherwise false.

Finally, let us suppose that $\alpha_R(v)$ is true. Analysing inference rules we can easily prove by induction on the complexity of formulas that there is an atomic formula in the consequent such that $\varphi_R(t_1, \ldots, t_n)(v)$ is true. This contradicts the definition of realization.

§ 2. Some elementary algorithmic properties

Let $\boldsymbol{A}$ be a class of similar models. In this section we shall consider the following properties of progress:

$\mathrm{Stop}_{\boldsymbol{A}} = \{K \in \boldsymbol{FS}$: for every model $A \in \boldsymbol{A} \models_A K\mathbf{1}\}$,

$\mathrm{Stop}'_{\boldsymbol{A}} = \{K \in \boldsymbol{FS}$: for every model $A \in \boldsymbol{A} \models_A \sim K\mathbf{1}\}$,

$\mathrm{Corr}_{\boldsymbol{A}} = \{\langle \alpha, K, \beta\rangle \in \boldsymbol{F}\times\boldsymbol{FS}\times\boldsymbol{F}$: for every model $A \in \boldsymbol{A} \models_A \alpha \Rightarrow K\beta\}$,

$\mathrm{Eq}_{\boldsymbol{A}} = \{\langle K, M\rangle \in \boldsymbol{FS}\times\boldsymbol{FS}$: for every model $A \in \boldsymbol{A} \models_A K \equiv M\}$.

In the case where $\boldsymbol{A}$ contains all similar models we shall omit the symbol $\boldsymbol{A}$. So, for instance, $\mathrm{Stop} = \{K \in \boldsymbol{FS}: \models K\mathbf{1}\}$.

If $\boldsymbol{A}$ is a class of models isomorphic to the model of natural numbers with successor $N = \langle \mathcal{N}, 0, \mathbf{S}, =\rangle$, then the above properties are those of partial recursive functions, since programs in this model and partial recursive functions are recursively isomorphic.

In what follows we shall try to place in an arithmetical hierarchy these properties for other classes of models.

§ 3. Model-independent properties

3.1. Stop, Corr $\in \Sigma^0_1 - \Pi^0_1$, Stop' $\in \Pi^0_1 - \Sigma^0_1$, Eq $\in \Pi^0_2 - \Sigma^0_2$.

Proof.

1. Stop $\in \Sigma^0_1$. This fact we obtain immediately from axiomatization. In diagrams of sequence of the form $\to K\mathbf{1}$ only finite rules of inference are used.

2. Stop $\notin \Pi^0_1$. For the proof see [44].

3. Stop' $\in \Pi^0_1$. In fact, $\models \sim K\mathbf{1}$ iff $\models \sim [K_1 * [\beta K_2]]\mathbf{1}$ where $\circ[K_1 * [\beta K_2]]$ is the normal form of K. Since K_1 is a loop-free program in the diagram if the sequent

$$\vdash \circ[K_1 * [\beta K_2]]\mathbf{1} \to,$$

after a finite number of steps we pass to a finite sequence of assertions:

$$\vdash \Gamma, s * [\beta K_2]\mathbf{1} \to \Delta,$$

where all the formulas in Γ and Δ are open. This assertion is in the class Π^0_1 since after applying the infinitistic rule only once we come to sequents of open formulas exclusively.

4. Stop' $\notin \Sigma^0_1$. For the proof see [44].

5. Eq $\in \Pi^0_2$. To begin with, consider an assertion $\vdash K\mathbf{1} \to M\mathbf{1}$. In this diagram, as in point 3, after a finite number of steps we obtain assertions $\vdash \Gamma, s * [\beta K_2] \to \Delta$ where M appears in Δ. After the application of the infinitistic rule, the resulting sequents will require in their proofs only finite rules of inference. Thus, this assertion is in the class Π^0_2. Since $K \equiv M$ is equivalent to the conjunction of three formulas of the form $K\mathbf{1} \Rightarrow M\mathbf{1}$, Eq is in the same arithmetical class.

6. Eq $\notin \Sigma^0_2$. Let us suppose the contrary. We first limit the set of non-logical constants to the functional symbols 0 (zero argument) and S (one argument). Let $K_i(x)$ denote a program in that language which in the model of natural numbers computes the partial recursive function of number i in the Gödel enumeration $(\varphi_i(x))$. Now, let us consider two programs:

$$M: \circ[[y/0] * [y \neq x\ [y/S(y)]]][y/0],$$

$$P_i: \circ[\circ[MK_i]\ [y/0]].$$

Then $P_i \equiv M$ iff $P_i\mathbf{1} \Leftrightarrow M\mathbf{1}$ because the output value of variable y in both programs is the same. So, from our assumption, $\{i: \models P_i\mathbf{1} \Leftrightarrow M\mathbf{1}\} \in \Sigma^0_2$. By the completeness theorem

$$\{i: \vdash P_i\mathbf{1} \to M\mathbf{1} \wedge \vdash M\mathbf{1} \to P_i\mathbf{1}\} \in \Sigma^0_2.$$

We add two axioms:

$\Gamma \to \Delta$ if formula $\mathbf{S}(x) = 0$ is in Γ or if $\mathbf{S}(x) = \mathbf{S}(y)$ in Γ and $x = y$ in Δ.

This will not change the arithmetical class, so again by the completeness theorem, the set $U = \{i: \alpha \models P_i\mathbf{1} \Leftrightarrow M\mathbf{1}\}$ where α: $\sim (S(x) = 0) \wedge (S(x) = S(y) \Leftrightarrow (x = y))$ belongs to Σ^0_2.

Now, if $A = \langle \mathscr{A}, o, \mathbf{s}\rangle$ is a model of α then a subsystem $B = \langle \mathscr{B}, o, \mathbf{s}\rangle$ where $\mathscr{B} = \{s^i(o): i \in \mathcal{N}\}$ is isomorphic to N. So, for every model A of α we have:

for every $a \in \mathscr{A}$, $M(a)$ is defined iff $P_i(a)$ is defined.

But $M(a)$ is defined iff $a \in \mathscr{B}$. We obtained the following chain of equivalences:

1. $i \in U$,
2. for every model A of α and every $a \in \mathscr{A}$, $M(a)$ is defined iff $P_i(a)$ is defined,
3. for every model A of α and every $a \in \mathscr{A}$, $a \in \mathscr{B}$ iff $P_i(a)$ is defined,
4. for every model A of α and every $a \in \mathscr{A}$, $a \in \mathcal{N}$ iff $P_i(a)$ is defined,
5. φ_i is total.

Since P_i is obtained recursively from i, the set $\{i: \varphi_i \text{ total}\}$ is recursively reducible to U. This is a contradiction because the set $\{i: \varphi_i \text{ total}\}$ is Π^0_2-complete (see [52]).

7. Corr $\in \Sigma^0_1 - \Pi^0_1$. It follows from the equivalence

$$K \in \mathrm{Stop} \quad \text{iff} \quad \langle \mathbf{1}, K, \mathbf{1}\rangle \in \mathrm{Corr}.$$

§ 4. Properties of programs in the fields of reals

We shall denote the class of models isomorphic to the model of reals $\langle \mathscr{R}, +, -, \cdot, ^{-1}, 0, 1\rangle$ by $\boldsymbol{R}$.

4.1. $\mathrm{Stop}_{\boldsymbol{R}}$, $\mathrm{Corr}_{\boldsymbol{R}} \Sigma_1^0 - \Pi_1^0$, $\mathrm{Stop}'_{\boldsymbol{R}} \Pi_1^0 - \Sigma_1^0$, $\mathrm{Eq}_{\boldsymbol{R}} \in \Pi_2^0 - \Sigma_2^0$.

Proof. In the proof we shall need the following lemma, due to E. Engeler ([36]):

4.2. *For every formula α which is a Boolean combination of formulas of the form $K\beta$ where β is open*

$$\models_{\boldsymbol{R}} \alpha \quad \textit{iff} \quad \mathrm{Char}\,0 \models \alpha$$

where Char 0 *denotes the set of algorithmic axioms of fields with characteristic zero.*

(1) $\mathrm{Stop}_{\boldsymbol{R}} \in \Sigma_1^0$. From 4.2 we obtain an equivalence:

$$\models_{\boldsymbol{R}} K\mathbf{1} \quad \text{iff} \quad \vdash [x/0] \cap [x/x+1](x \neq 0) \to K\mathbf{1}.$$

Examining diagrams of this kind of sequents, we see that finite rules are used. This proves point (1).

(2) $\mathrm{Stop}_{\boldsymbol{R}} \notin \Pi_1^0$. Let $K_i(x)$ be a program which computes in the system $\boldsymbol{R}$ a partial recursive function $\varphi_i(x)$ (the construction of this program is the same as in N). Let N_i be a program $\circ[[x/1 + \ldots + 1]K_i]$ where the sum is equal to i. So, $\varphi_i(i)$ is defined iff $\models_{\boldsymbol{R}} N_i\mathbf{1}$.

(3) $\mathrm{Stop}'_{\boldsymbol{R}} \in \Pi_1^0$. We have the following chain of equivalences:

1. $K \in \mathrm{Stop}'_{\boldsymbol{R}}$ iff
2. $\models_{\boldsymbol{R}} \sim K\mathbf{1}$ iff
3. $\models_{\boldsymbol{R}} \sim \circ[K_1 * [\alpha K_2]]\mathbf{1}$ iff
4. $(\forall r_1, \ldots, r_n \in \mathscr{R})(\forall i \in \mathscr{N})(\models_{\boldsymbol{R}} K_1 K_2^i \alpha)$ iff
5. $(\forall i \in \mathscr{N}) \models_{\boldsymbol{R}} (\forall x_1, \ldots, x_n)\, K_1 K_2^i \alpha$ where $x_1, \ldots, x_n$ are all input variables of K.

Notice that the formula $K_1 K_2^i \alpha$ is effectively equivalent to an open formula $\beta_i(x_1, \ldots, x_n)$. From Tarski's ([55]) theorem we infer that the relation $\models_{\boldsymbol{R}} (\forall x_1, \ldots, x_n) K_1 K_2^i \alpha$ is recursive.

(4) $\mathrm{Stop}_{\boldsymbol{R}} \notin \Sigma_1^0$. $\varphi_i(i)$ is undefined iff $N_i \in \mathrm{Stop}'_{\boldsymbol{R}}$.

(5) $\mathrm{Eq}_{\boldsymbol{R}} \in \Pi_2^0$. $K \equiv M$ is equivalent to a Boolean combination of formulas of the form $K\alpha$. So, by applying Engeler's lemma and examining the diagrams of corresponding sequents we easily prove (5).

(6) $\mathrm{Eq}_{\boldsymbol{R}} \notin \Sigma_2^0$. Let M denote a program:

$$\circ\Big[\circ[[y/0] * [(y \neq x)[y/y+1]]][y/0]\Big]$$

and let P_i: $\circ[K_i[y/0]]$ where y is for both programs a unique output variable. Now, we have the equivalence:

$$\langle M, P_i\rangle \in \mathrm{Eq}_{\boldsymbol{R}} \quad \text{iff} \quad \varphi_i \text{ is total.}$$

(7) $\mathrm{Corr}_{\boldsymbol{R}} \in \Sigma_1^0 - \Pi_1^0$. As usual, $K \in \mathrm{Stop}_{\boldsymbol{R}}$ iff $\langle \mathbf{1}, K, \mathbf{1}\rangle \in \mathrm{Corr}_{\boldsymbol{R}}$.

§ 5. Properties of programs in the ordered field of reals

Let $\boldsymbol{R}$ denote a class of models isomorphic to the ordered field of reals $\langle \mathscr{R}, +, -, \cdot, ^{-1}, <, 0, 1\rangle$.

5.1. $\mathrm{Stop}_{\boldsymbol{R}<}$, $\mathrm{Corr}_{\boldsymbol{R}<}$, $\mathrm{Eq}_{\boldsymbol{R}<} \in \Pi_2^0 - \Sigma_2^0$, $\mathrm{Stop}'_{\boldsymbol{R}<} \in \Pi_1^0 - \Sigma_1^0$.

Proof. We shall need Engeler's lemma ([36]):

5.2. *For every formula α which is a Boolean combination of formulas of form $K\beta$ where β is open,*

$$\models_{\boldsymbol{R}<} \alpha \quad \textit{iff} \quad \mathrm{Arch} \models \alpha,$$

where Arch *denotes the set of algorithmic axioms of Archimedean ordered fields.*

(1) $\mathrm{Stop}_{\boldsymbol{R}<} \in \Pi_2^0$. By 4.2, $K \in \mathrm{Stop}_{\boldsymbol{R}<}$ iff $\mathrm{Arch} \models K\mathbf{1}$ but Arch is the conjunction of open formulas and the formula

$$(x > 0 \wedge y > 0) \Rightarrow [z/y] \bigcup [z/z+y] \quad (x < z).$$

The diagram of the sequent $\mathrm{Arch} \to K\mathbf{1}$ is in the same class as that of the sequent $M\mathbf{1} \to K\mathbf{1}$. So,

$$\mathrm{Stop}_{\boldsymbol{R}<} \in \Pi_2^0.$$

(2) $\mathrm{Stop}_{\boldsymbol{R}<} \notin \Sigma_2^0$. Let us consider the program

$$M_i(y)\colon \circ\Big[\circ\big[[x/0] * [x < y\ \ [x/x+1]]\big]\ K_i(x)\Big]$$

where $K_i(x)$ is the same as in § 4, (6). Hence, $\models_{\boldsymbol{R}<} M_i\mathbf{1}$ iff $(\forall n \in \mathscr{N}) \models_{\boldsymbol{R}<} K_i(n)\mathbf{1}$ iff φ_i is total.

(3) $\mathrm{Eq}_{\boldsymbol{R}<} \in \Pi_2^0 - \Sigma_2^0$. Proof as in § 4.

(4) $\mathrm{Stop}'_{\boldsymbol{R}<} \Pi_1^0 - \Sigma_1^0$. Exactly the same situation as for $\mathrm{Stop}'_{\boldsymbol{R}}$.

(5) $\mathrm{Corr}_{\boldsymbol{R}<} \in \Pi_2^0 - \Sigma_2^0$. Since $K \in \mathrm{Stop}_{\boldsymbol{R}<}$ iff $\langle \mathbf{1}, K, \mathbf{1}\rangle \in \mathrm{Corr}_{\boldsymbol{R}<}$.

Remark. The idea used in the case of point (2) for $\mathrm{Stop}_{\boldsymbol{R}<}$ cannot be copied in the case of $\mathrm{Stop}_{\boldsymbol{R}}$. If we have taken

$$M_i(y)\colon \circ\Big[\circ\big[[x/0] * [x \neq y\ \ [x/x+1]]\big] K_i\Big]$$

we should obtain non $\models_{\boldsymbol{R}} M_i\mathbf{1}$ for every i.

COROLLARY. *The relation $<$ is not programmable in the model of reals without ordering $\boldsymbol{R}$.*

Proof follows immediately from the fact that $\mathrm{Stop}_{\boldsymbol{R}<} \notin \Sigma_2^0$ and $\mathrm{Stop}_{\boldsymbol{R}} \in \Sigma_1^0$.

§ 6. Degree of recursive unsolvability of algorithmic logic

Let V denote the set of all sentences of the first order arithmetic valid in the standard model. Let W denote the set of all tautologies of algorithmic logic. In [11] the following theorem is proved:

6.1. (i) *V and W are recursively isomorphic.*

(ii) *W is recursively isomorphic to 0^{ω}.*

(iii) *W is not an arithmetical set.*

(iv) *It is impossible to replace infinitistic inference rules of any kind of axiomatization by finitistic ones* [12].

Chapter IV

DESCRIPTIONS AND THE MODULAR STRUCTURE OF PROGRAMS

§ 1. Verification of program correctness and the modular method

In up-to-date practice programs are checked for some simple input data for which the solution of the problem is known. If the test happens to be positive for a program, the latter is considered to be correct and is passed for exploitation. However, after some time one finds an input data for which the results are incorrect. Therefore the question what one should require from a programmer designing a program in order to be certain of the correctness of the program is of great importance.

One of the possibilities consists in demanding that a programmer should supply the proof of correctness in the appropriate, formalized, algorithmic theory. The task of the machine would be reduced to the examination whether the proof contains any errors.

A demand which is easier to fulfil is that the programmer should supply a complete net of subtasks for some segments of the program, i.e. the so-called description or documentation of the program. This approach to program verification is called Floyd's method [37]. Attempts at mechanical verification of programs based on Floyd's method have been presented in [1], [40] and [41].

The describing of programs corresponds to the modular method of their design. The process of designing a program begins with the elaboration of its logical structure. This consists in splitting the overall task into a net of subtasks in such a way that having obtained programs which accomplish the subtasks, we can put them together in an appropriate manner so as to obtain a program correct with respect to the overall task.

§ 2. Properties of programs in algorithmic theories

In this chapter we shall use extended algorithmic logic (see II, § 5), i.e., algorithmic logic with classical quantifiers.

According to I, § 4 we admit the following definitions of properties of programs in an algorithmic theory $\mathscr{T} = \{\mathscr{L}, \mathscr{C}, \mathscr{A}\}$.

Program K is said to be *correct* with respect to formulas α and β in theory $\mathscr{T}$ provided the formula $(\alpha \Rightarrow K\beta)$ is a theorem of theory $\mathscr{T}$.

Program K is said to be *partially correct* with respect to formulas α and β in theory $\mathscr{T}$ provided the formula $(\mathbf{K1} \wedge \alpha \Rightarrow K\beta)$ is a theorem of that theory.

Program K is said to be adequate with respect to formulas α and β in theory $\mathscr{T}$ provided the formula $\big((\alpha \Leftrightarrow K\beta) \wedge (\alpha K \Leftrightarrow \beta)\big)$ is a theorem of that theory.

EXAMPLE. Let us consider program M: $\circ\big[[t/0\ z/x] * [z \geqslant y\ [z/z-y\ t/t+1]]\big]$ and algorithmic theory of integers (see I, § 11). This theory is categorical. Hence every question about a property of a program in the algolic realization of integers is equivalently reduced to the problem whether an appropriate formula is derivable in algorithmic theory of integers. One can prove that:

M is partially correct and not correct with respect to the formulas $x \geqslant 0 \wedge y \geqslant 0$ and $x = y \cdot t + z \wedge 0 \leqslant z \wedge z < y$. M is correct and not adequate with respect to $x \geqslant 0 \wedge y > 0$ and $x = y \cdot t + z \wedge 0 \leqslant z \wedge z < y$. M is adequate (hence partially correct and correct as well) with respect to $x \geqslant 0 \wedge y > 0$ and $x = y \cdot t + z \wedge 0 \leqslant z \wedge z < y \wedge \wedge t \geqslant 0$.

The following lemma is useful in practice.

2.1. *Program K is partially correct with respect to α and β in an algorithmic theory $\mathscr{T}$ iff $(\alpha K \Rightarrow \beta)$ is a theorem of $\mathscr{T}$. Program K is correct with respect to α and β in an algorithmic theory $\mathscr{T}$ iff K is partially correct with respect to α and β in $\mathscr{T}$ and additionally $(\alpha \Rightarrow K\mathbf{1})$ is a theorem of $\mathscr{T}$.* [6], [7]

§ 3. Compatibility of the modular structure of programs and descriptions

We shall investigate the possibilities of deriving properties of programs from the appropriate properties of their modular structures.

By a *module* of a ***FS***-program we shall understand any subexpression of K which is also a program. The set of all modules of the program K will be denoted by $\mathrm{Mod}(K)$.

A pair $H = (I, \hat{K})$ is said to be a *tree* of the program K if $I \subset \{1,2\}^*$, $\hat{K}$: $I \underset{\text{onto}}{\rightarrow} \mathrm{Mod}(K)$ and the following conditions are fulfilled:

(1) the empty sequence e belongs to I and $\hat{K}_e$: K;

(2) if i is in I and $\hat{K}_i$: $\circ[L\,M]$ or $\hat{K}_i$: $\curlywedge\,[\gamma L\,M]$ then $i1$, $i2$ are in I and $\hat{K}_{i1}$: L, $\hat{K}_{i2}$: M;

(3) if i is in I and $\hat{K}_i$: $*[\gamma\,M]$ then $i1$ is in I and $\hat{K}_{i1}$: M;

(4) every element in I can be obtained from the vertice e by means of rules (2) and (3).

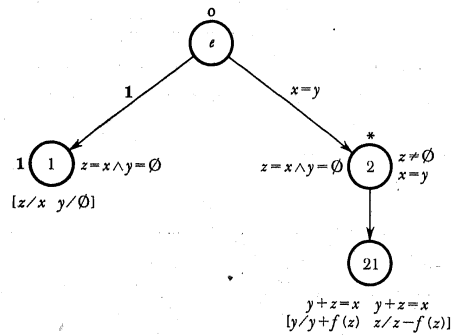
The function $\hat{K} = \{\hat{K}_i\}_{i \in I}$ is said to be the *modular structure* of the program K.

By a *description* of the program K we shall understand any sequence $A = \{(\alpha_i, \beta_i)\}_{i \in I}$ of pairs of formulas. For every i in I, the pair $A(i) = (\alpha_i, \beta_i)$ defines a subtask of the module $\hat{K}_i$. The formulas α_i and β_i are called an *input formula* and an *output formula* of the module $\hat{K}_i$, respectively. In particular, the overall task is defined by the pair $A(e) = (\alpha_e, \beta_e)$.

EXAMPLE.

$$M\colon \circ\,[[z/x\ y/\emptyset] * [z \neq \emptyset\ [y/y+f(z)\ z/z-f(z)]]].$$

Below there is a tree of M with an associated description.

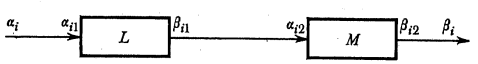


According to the definition, the modular structure of a program is determined uniquely by the program alone. However, in practice the choice of the partition of a program into modules depends also on the programmer. The results of this chapter will not change if we allow some loop-free programs to stand at the terminal vertices of the program tree. It was this approach that was applied in [1].

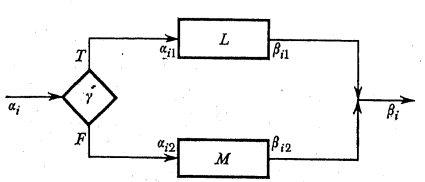
By a *verification condition* of a vertice i in I with respect to a description $A = \{(\alpha_i, \beta_i)\}_{i\in I}$ we shall understand a formula VC_i defined as follows:

(1) if $\hat{K}_i$: s is a substitution then VC_i: $(\alpha_i \Rightarrow (\overline{s\beta_i}))$;

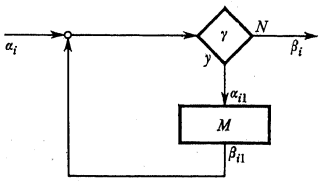
(2) if $\hat{K}_i$: $\circ[L\ M]$ then VC_i: $((\alpha_i \Rightarrow \alpha_{i1}) \wedge (\beta_{i1} \Rightarrow \alpha_{i2}) \wedge (\beta_{i2} \Rightarrow \beta_i))$, i.e.,



(3) if $\hat{K}_i$: $\curlyvee$ $[\gamma L\ M]$ then VC_i: $((\alpha_i \wedge \gamma \Rightarrow \alpha_{i1}) \wedge (\alpha_i \wedge \neg\gamma \Rightarrow \alpha_{i2}) \wedge (\beta_{i1} \vee \beta_{i2} \Rightarrow \beta_i))$, i.e.,



(4) if $\hat{K}_i$: $*[\gamma M]$ then VC_i: $\big(((\alpha_i \vee \beta_{i1}) \wedge \gamma \Rightarrow \alpha_{i1}) \wedge ((\alpha_i \vee \beta_{i1}) \wedge \neg\gamma \Rightarrow \beta_i)\big)$, i.e.,



By a *verification condition* of a program K with respect to a description A we shall understand the formula VC: $\bigwedge_{i\in I} VC_i$. The description A is called *compatible* with the modular structure $\hat{K}$ in an algorithmic theory $\mathscr{T} = \{\mathscr{L}, \mathscr{C}, \mathscr{A}\}$ provided $VC \in \mathscr{C}(A)$.

Let $H = (I, \hat{K})$ be the tree of a program K, let $A = \{(\alpha_i, \beta_i)\}_{i\in I}$ be a description of K and let $\mathscr{T} = \{\mathscr{L}, \mathscr{C}, \mathscr{A}\}$ be an algorithmic theory. The modular structure $\hat{K}$ is said to be *correct* (partially correct, adequate) with respect to A in $\mathscr{T}$ provided that

(1) A is compatible with $\hat{K}$ in $\mathscr{T}$;

(2) for every i in I, the ith module $\hat{K}_i$ is correct (partially correct, adequate) with respect to α_i and β_i in $\mathscr{T}$.

The following theorem explains the meaning of compatible descriptions.

3.1. *If A is compatible with the modular structure $\hat{K}$ in $\mathscr{T}$, then $\hat{K}$ is partially correct with respect to A in $\mathscr{T}$.*

As a corollary we obtain

3.2. *If A is compatible with the modular structure $\hat{K}$ in $\mathscr{T}$, then the program K is partially correct with respect to α_ε and β_ε in $\mathscr{T}$.*

The above lemma justifies Floyd's method of proving the partial correctness of a program (see also [32], [37], [14], [45], [47]).

EXAMPLE. Let Boolf be the formal algorithmic theory of Boolean algebras with the following additional axiom:

(Af) $x + f(x) = x$

where f is an additional 1-argument functor.

Let us consider the program M and its description from the preceding example in this section. In order to prove the partial correctness of M with respect to formulas $\mathbf{1}$ and $x = y$ in the Boolf theory, by 3.2 it is sufficient to show that the verification condition VC is a theorem of the Boolf theory. Let us observe that VC is the conjunction $VC_\varepsilon \wedge VC_1 \wedge VC_2 \wedge VC_{21}$ where

VC_ε: $((\mathbf{1} \Rightarrow \mathbf{1}) \wedge (z = x \wedge y = \emptyset \Rightarrow z = x \wedge y = \emptyset) \wedge (x = y \Rightarrow x = y))$,

VC_1: $(\mathbf{1} \Rightarrow x = x \wedge \emptyset = \emptyset)$,

VC_2: $\Big(\big(\big((z = x \wedge y = \emptyset) \vee y + z = x\big) \wedge z = \emptyset \Rightarrow x = y\big) \wedge \big(\big((z = x \wedge y = \emptyset) \vee y + z = x\big) \wedge z \neq \emptyset \Rightarrow y + z = x\big)\Big)$,

VC_{21}: $\big(y + z = x \Rightarrow \big(y + f(z) + \big(z - f(z)\big)\big) = x\big)$.

Evidently VC_e, VC_1 and VC_2 are tautologies and it remains to prove that VC_{21} is a theorem of the Boolf theory. To begin with, $\big(y + f(z)\big) + \big(z - f(z)\big) = y + z + f(z)$ is a theorem of the theory of Boolean algebras. Moreover, by the axiom (Af) the formula $z + f(z) = z$ is a theorem of Boolf. Hence VC_{21} is also a theorem.

§ 4. An extension of a task of a program to a description

The properties of correctness, partial correctness and adequacy have the common feature of extendability of a program task to the whole description.

4.1. *For every algorithmic theory $\mathscr{T}$ for every program K and for any formulas α, β, if K is correct (partially correct, adequate) with respect to α and β in $\mathscr{T}$ then there exists a description A of K such that the modular structure $\hat{K}$ is correct (partially correct, adequate) with respect to A in $\mathscr{T}$ and $A(e) = (\alpha, \beta)$.*

Strong correctness, expressible by $\big(K\mathbf{1} \wedge (\alpha \Rightarrow K\beta)\big)$, is a property for which 4.1 fails to hold.

By means of 3.1 and 4.1 one can build (see [6]) formal syntactic systems for proving the partial correctness of a program (or for the synthesis of programs partially correct with respect to a given task). These systems resemble Hoare's approach to proving partial correctness (see [38]). Contrary to [38] the systems based on 3.1 and 4.1 possess the completeness property of syntactic derivations.

§ 5. Open descriptions

In practice we usually meet descriptions consisting solely of open formulas. Such descriptions will be called *open*. The following theorem establishes the complexity degrees of properties of a modular structure with respect to open descriptions.

5.1. *In every algorithmic theory $\mathscr{T}$*

(1) *the properties of partial correctness of a modular structure with respect to open descriptions and of the provability of open formulas are recursively reducible to each other*;

(2) *the properties of correctness of a modular structure with respect to open descriptions and of correctness of a program with respect to open formulas are recursively reducible to each other.*

Hence supplying a program with a description simplifies the examination only in the case of partial correctness. However, in order to verify a program fully, we additionally have to check the stop property of that program.

§ 6. Properties of programs and second order logic

By $\boldsymbol{F}_{II}$ we shall denote the set of all formulas of the second order predicate calculus (i.e., quantifiers can bind also predicate variables).

We say that formula α is $\boldsymbol{F}_{II}$*-existential* ($\boldsymbol{F}_{II}$*-universal*) provided there exists a formula γ of the first order predicate calculus and a sequence of predicates $\vec{u}$ such that $\alpha \Leftrightarrow \exists \vec{u} \gamma$ ($\alpha \Leftrightarrow \forall \vec{u} \gamma$) is a tautology.

By means of 3.1 and 4.1 one can prove that

6.1. *For any programs K, M and for any formulas α, β of the first order predicate calculus*

(1) *the following formulas ate $\boldsymbol{F}_{II}$-existential*:

a. $K\mathbf{1} \wedge \alpha \Rightarrow K\beta$
b. $\alpha K \Rightarrow \beta$

} formulas of partial correctness;

(2) the following formulas are $\boldsymbol{F}_{II}$-universal:

a. $\alpha \Rightarrow K\beta$ formula of correctness,
b. $K\mathbf{1}$ halting formula,
c. $K\mathbf{1} \wedge M\mathbf{1} \wedge (K\alpha \Leftrightarrow M\alpha)$ formula of equivalence of total programs,
d. $\mathbf{1}K$ formula of counter-domain.

Manna [45] has proved 6.1 for go-to programs (except the points (1)b and (2)d). The point 2 of this theorem permits in some cases translations of problems for programs into better known problems of first order logic (like the Herbrand theorem). The question about the translation of the correctness problem into first order logic can be formulated as follows:

HYPOTHESIS 6.2. *Let A be a recursive set of formulas of the first order predicate calculus, let $\mathscr{T} = \{\mathscr{L}, \mathscr{C}, \mathscr{A}\}$ be the algorithmic theory and let α, β be formulas of the first order predicate calculus. The question whether $(\alpha \Rightarrow K\beta) \in \mathscr{C}(\mathscr{A})$ is recursively reducible to the problem whether one of the formulas from an effectively defined sequence of classical formulas does not possess a model.*

By 6.1 and the theorem on deduction II, § 1.2 the fact 6.2 is valid when A is a finite set. Let us observe that if we put "classical theory" instead of "algorithmic theory" and "classical formula" instead of "$(\alpha \Rightarrow K\beta)$" then 6.2 becomes true on account of the compactness theorem. This way of reasoning cannot be applied to algorithmic theories because the compactness theorem fails to hold in general in algorithmic logic II, § 1.

Chapter V

PROCEDURES

The semantics of procedures is the main topic of this chapter. The first attempt to explain procedures consists in treating them as subprograms—fragments of a program. In other words, one treats a procedure as a recipe for computing. Even this approach entails several difficulties, mainly due to the various ways of parameter

transmission. The notion of formal computation, introduced here, has been proposed in order to obtain a formalized tool for investigations of computations in in the presence of procedures.

Another aspect of programming with procedures is the structural way of thinking and solving problems. Procedures can be viewed as a tool for supplying the existing computers with new capabilities. This intuitive, informal view of procedures correspond to conceiving them as axioms of an algorithmic theory (implicit definitions). Then the question of models arises in a natural way. This embodies the fixed point approach to procedures and generalizes it. It will be shown that computations lead to models of procedures. Using the Gentzen-style formalization of algorithmic logic, we can indicate another way of constructing models, usually greater than those defined by computations. The theory proposed here does not require any additional constructions and is straightforward also in proving properties of procedures.

§ 1. Procedures, formal computations

Let $\varphi_1, \ldots, \varphi_p$ and $\varrho_1, \ldots, \varrho_l$ be partial-functors and predicates which do not belong to the language $\mathscr{L}$. We shall assume that the partial-functor φ_j is m_j-ary ($j = 1, \ldots, p$) and that the predicate ϱ_i is n_i-ary ($i = 1, \ldots, l$). By $\mathscr{L}'$ we shall denote the extension of $\mathscr{L}$ obtained by adding the partial-functors $\varphi_1, \ldots, \varphi_p$ and the predicates $\varrho_1, \ldots, \varrho_l$ to the alphabet of $\mathscr{L}$.

Let $K_1, \ldots, K_l, M_1, \ldots, M_p$ be programs, i.e., *FS*-expressions, $\alpha_1, \ldots, \alpha_l$ open formulas, $\tau_1, \ldots, \tau_p$ terms of the language $\mathscr{L}'$ such that the free variables of formulas $K_i\alpha_i$ are $x_1, \ldots, x_{n_i}$ (for $i = 1, \ldots, l$) and the free variables of terms $M_j\tau_j$ are $x_1, \ldots, x_{m_j}$ (for $j = 1, \ldots, p$); then the following system of equations and equivalences

$$(*)\quad \begin{cases} \varphi_1(x_1, \ldots, x_{m_1}) = M_1\tau_1, \\ \ldots\ldots\ldots\ldots \\ \varphi_p(x_1, \ldots, x_{m_p}) = M_p\tau_p, \\ \varrho_1(x_1, \ldots, x_{n_1}) = K_1\alpha_1, \\ \ldots\ldots\ldots\ldots \\ \varrho_l(x_1, \ldots, x_{n_l}) = K_l\alpha_l \end{cases}$$

will be called a *system of procedures defining the notions* $\varphi_1, \ldots, \varphi_p, \varrho_1, \ldots, \varrho_l$.

Note that each procedure can be translated into an ALGOL-like programming language as follows: the equation

$$\varphi(x_1 \ldots x_m) = M\tau$$

turns into the following procedure:

real procedure $\varphi(x_1, \ldots, x_m)$; **value** $x_1, \ldots, x_m$;
real $x_1, \ldots, x_m$;
begin "the program M translated into an ALGOL-like language"; $\varphi := \tau$
end of φ.

Similarly, the equivalences of the (*)-system can be translated into Boolean procedures.

Let R be a realization of language $\mathscr{L}$ in a nonempty set $\mathscr{J}$ and a two-element Boolean algebra B_0. Given a certain expression $\omega \in \mathscr{L}'$, we shall define its value at a valuation v in the realization R. Obviously, the value of a term will be an element of the set J, the value of a formula will be an element of the algebra B_0 of the logical values truth and falsity, the value of a program will be a valuation of variables.

We introduce the notion of computation. By a *computation* we shall understand a finite sequence of ordered triplets $\langle v, \omega, w\rangle$ where

v is a valuation of variables,
ω is an expression of the language $\mathscr{L}'$,
w is a value associated with the expression at the valuation v.

We assume that R is a semantic realization [49]. We shall adopt the following set of computing rules. Each rule has at least one triplet as a premise and exactly one triplet called the conclusion or result.

(F) $\dfrac{\{\langle v, \tau_i, j_i\rangle\}_{i=1}^n}{\langle v, \varphi(\tau_1 \ldots \tau_n), j\rangle}$ where φ is an n-ary functor of $\mathscr{L}$, $\varphi_R(j_1, \ldots, j_n) = j$,

(R) $\dfrac{\{\langle v, \tau_i, j_i\rangle\}_{i=1}^n}{\langle v, \varrho(\tau_1 \ldots \tau_n), w\rangle}$ where ϱ is an n-ary predicate of $\mathscr{L}$, $\varrho_R(j_1, \ldots, j_n) = w$,

(D1) $\dfrac{\langle v, \alpha, /\rangle\langle v, \beta, /\rangle}{\langle v, (\alpha \vee \beta), /\rangle}$,

(D2) $\dfrac{\langle v, \alpha, |/\rangle}{\langle v, (\alpha \vee \beta), |/\rangle}$, (D3) $\dfrac{\langle v, \beta, |/\rangle}{\langle v, (\alpha \vee \beta), |/\rangle}$,

(N1) $\dfrac{\langle v, \alpha, /|\rangle}{\langle v, \neg\alpha, |/\rangle}$, (N2) $\dfrac{\langle v, \alpha, |/\rangle}{\langle v, \neg\alpha, /|\rangle}$,

(I1) $\dfrac{\langle v, \alpha, |/\rangle\langle v, \beta, /|\rangle}{\langle v, (\alpha \Rightarrow \beta), /|\rangle}$, (I2) $\dfrac{\langle v, \alpha, /|\rangle}{\langle v, (\alpha \Rightarrow \beta), |/\rangle}$,

(I3) $\dfrac{\langle v, \beta, |/\rangle}{\langle v, (\alpha \Rightarrow \beta), |/\rangle}$, (C1) $\dfrac{\langle v, \alpha, |/\rangle\langle v, \beta, |/\rangle}{\langle v, (\alpha \wedge \beta), |/\rangle}$,

(C2) $\dfrac{\langle v, \alpha, /|\rangle}{\langle v, (\alpha \wedge \beta), /|\rangle}$, (C3) $\dfrac{\langle v, \beta, |/\rangle}{\langle v, (\alpha \wedge \beta), /|\rangle}$,

(P) $\dfrac{\{\langle v, \omega_i, w_i\rangle\}_{i=1}^n}{\langle v, [z_1/\omega_1 \ldots z_n/\omega_n], v'\rangle}$, where $v'(z) = \begin{cases} w_i & \text{if } z = z_i, \\ v(z) & \text{otherwise,} \end{cases}$

(S) $\dfrac{\langle v, K, v'\rangle\langle v', M, v''\rangle}{\langle v, \circ[K\,M], v''\rangle}$, (B1) $\dfrac{\langle v, \alpha, |/\rangle\langle v, K, v'\rangle}{\langle v, \leq [\alpha K\,M], v'\rangle}$,

(B2) $\dfrac{\langle v, \alpha, /|\rangle\langle v, M, v'\rangle}{\langle v, \preceq[\alpha K\, M], v'\rangle}$, (G1) $\dfrac{\langle v, \alpha, /|\rangle}{\langle v, *[\alpha K], v\rangle}$,

(G2) $\dfrac{\langle v, \alpha, |/\rangle\langle v, K, v''\rangle\langle v'', *[\alpha K], v'\rangle}{\langle v, *[\alpha K], v'\rangle}$,

(Kτ) $\dfrac{\langle v, K, v'\rangle\langle v', \tau, j\rangle}{\langle v, K\tau, j\rangle}$, (K$\alpha$) $\dfrac{\langle v, K, v'\rangle\langle v', \alpha, w\rangle}{\langle v, K\alpha, w\rangle}$,

(Fv) $\dfrac{\langle v, [x_1/\tau_1 \dots x_n/\tau_n]M\tau, w\rangle}{\langle v, \varphi(\tau_1 \dots \tau_n), w\rangle}$, where φ is an n-ary functor defined by the procedure, $\varphi(x_1 \dots x_n) = M\tau$,

(Rv) $\dfrac{\langle v, [x_1/\tau_1 \dots x_n/\tau_n]K\alpha, w\rangle}{\langle v, \varrho(\tau_1 \dots \tau_n), w\rangle}$, where ϱ is an n-ary predicate defined by the procedure $\varrho(x_1 \dots x_n) = K\alpha$.

Triplets of the form

$$\langle v, z, v(z)\rangle \quad \text{or} \quad \langle v, \varphi, \varphi_R\rangle,$$

where v is a valuation, z a variable, $v(z)$ the value of z at v, φ a zero-argument functor, φ_R its realization, i.e., a constant from the set J, will be called elementary.

Let ω be an expression of the language $\mathscr{L}'$, and v a valuation of variables. By a formal computation of the value w of the expression ω at the valuation v with the use of the system $(*)$ of procedures in the realization R we shall understand any finite sequence of ordered triplets $\{\langle v_i, \omega_i, w_i\rangle\}_{i=1}^{N_0}$ such that

(i) the last element of the sequence is identical with $\langle v, \omega, w\rangle$,

(ii) for every $i \leqslant N_0$ either the triplet $\langle v_i, \omega_i, w_i\rangle$ is elementary, or it is a result in a computing rule from some triplets among $\langle v_1, \omega_1, w_1\rangle \dots \langle v_{i-1}, \omega_{i-1}, w_{i-1}\rangle$ which are premises in that rule.

In the sequel we shall use the shorter form "a computation of a triplet $\langle v, \omega, w\rangle$".

Obviously, some triplets possess computations, others do not. Observe that there are some expressions ω of $\mathscr{L}'$ for which there exists no valuation v and no value w such that the triplet $\langle v, \omega, w\rangle$ has a computation. Moreover, some pairs of valuations and expressions of $\mathscr{L}$ have the same property.

EXAMPLE 1.1.

1. $\left\langle v: \frac{x}{2}\frac{y}{3}\frac{u}{2}\frac{z}{2}\frac{n}{5}, 1, 1\right\rangle$, (e)
2. $\left\langle v, [n/1], v': \frac{x}{2}\frac{y}{3}\frac{u}{2}\frac{z}{2}\frac{n}{1}\right\rangle$, (P, 1)
3. $\langle v', 1, 1\rangle$, (e)
4. $\langle v', n, 1\rangle$, (e)
5. $\langle v', n-1, 0\rangle$, (F, 3, 4)
6. $\left\langle v', [n/n-1], v_3: \frac{x}{2}\frac{y}{3}\frac{u}{2}\frac{z}{2}\frac{n}{0}\right\rangle$, (P, 5)

7. $\langle v_3, n, 0\rangle$, (e)
8. $\langle v_3, 0, 0\rangle$, (e)
9. $\langle v_3, n = 0, |/\rangle$ (R, 7, 8)
10. $\langle v_3, 1, 1\rangle$, (e)
11. $\left\langle v_3, [z/1], v_4: \frac{x}{2}\frac{y}{3}\frac{u}{2}\frac{z}{1}\frac{n}{0}\right\rangle$, (P, 10)
12. $\langle v_3, \preceq[n = 0\ [z/1][z/n\times f(n-1)]], v_4\rangle$, (B1, 9, 11)
13. $\langle v_4, z, 1\rangle$ (e)
14. $\langle v_3, \preceq[n = 0\ [z/1][z/n\times f(n-1)]]z, 1\rangle$, (K$\tau$, 13, 12)
15. $\langle v', [n/n-1]\preceq[n = 0\ [z/1][z/n\times f(n-1)]]z, 1\rangle$, (K$\tau$, 6, 14)
16. $\langle v', n, 1\rangle$, (e)
17. $\langle v', f(n-1), 1\rangle$. (Fv, 15)

In this example $\mathscr{L}$ is the language of arithmetic, and R its realization in the set of natural numbers; the system $(*)$ contains one procedure

$$f(n) = \preceq[n = 0\ [z/1][z/n\times f(n-1)]]z.$$

The sequence 1–17 is a computation of $\langle v', f(n-1), 1\rangle$.

Remark. Observe that the example is effective owing to the simplification we made when the notion of valuation was reduced to a finite sequence of values of those variables only that occur in $(*)$ and/or in the expression ω.

We can replace the rules (Fv) and (Rv) by the rules (Fn) and

(Rn) $\dfrac{\langle v, \overline{[x_1/\tau_1 \dots x_n/\tau_n]K\alpha}, w\rangle}{\langle v, \varrho(\tau_1, \dots, \tau_n), w\rangle}$

where $\overline{[x_1/\tau_1 \dots x_n/\tau_n]K\alpha}$ denotes the expression—result of the substitution of terms $\tau_1, \dots, \tau_n$ for variables $x_1, \dots, x_n$ in the expression $K\alpha$. The rule (Fn) is similar.

These rules can be applied only if the resulting expressions belong to the language $\mathscr{L}'$. In this way we obtain the second notion of computation. A computation using rules (Fn) and (Rn) will be called a *computation "by name"*. If it is necessary, we shall call computations of the first kind *computations "by value"*. Obviously, one can introduce different mixed types of computations. The following example asserts that the notions of computations "by value" and "by name" are different.

EXAMPLE 1.2. Let us consider the procedure

$$s(x, i) = \circ[[i/i+1][z/x]]z$$

and the realization in the set of integers. For any valuation v the computations of the term $s(n^3, n)$ will give different results. We obtain n^3 in the case of computations "by value" and $(n+1)^3$ in the case of computations "by name".

A triplet may possess a computation "by name" and no computation "by value", as can be seen from

EXAMPLE 1.3. Consider the procedure

$$f(x,y) = \varkappa[x = 0[z/2][z/f(x-1, f(x,y))]]z.$$

The triplet $\left\langle \frac{x}{1}\ \frac{y}{2}, f(x,y), 2 \right\rangle$ possesses a computation "by name" and does not possess any computation "by value".

§ 2. Basic properties of computations

The following lemma indicates that any two computations for a valuation v and an expression ω bring the same result w, even if they are different.

2.1. *If two triplets* $\langle v, \omega, w_1\rangle$ *and* $\langle v, \omega, w_2\rangle$ *possess computations, then* $w_1 = w_2$.

This lemma can be repeated for computations "by name":

2.1^n. *If two triplets* $\langle v, \omega, w_1\rangle$ *and* $\langle v, \omega, w_2\rangle$ *possess computations "by name", then* $w_1 = w_2$.

The following lemma shows that the notion of formal computation is an extension of the notion of the semantic of a formalized algorithmic language.

2.2. *For every expression* $\omega \in \mathscr{L}$, *if there exists a computation of* $\langle v, \omega, w\rangle$ *in the realization* R *then* $\omega_R(v) = w$.

§ 3. An example of an inconsistent procedure

Making use of the notion of computation, we can define the realization R^c, which is an extension of the realization R, by putting

$$\varphi_{R^c} = \varphi_R \quad \text{and} \quad \varrho_{R^c} = \varrho_R \quad \text{for} \quad \varphi, \varrho \in \mathscr{L}$$

and assuming that

$$\forall(j_1, \ldots, j_n \in J)\, \varphi_{R^c}(j_1, \ldots, j_n) = j$$

if there exists a computation of $\langle v, \varphi(x_1 \ldots x_n), j\rangle$ where $v(x_i) = j_i$ for $i = 1, \ldots, n$, undefined otherwise;

$$\forall(j_1, \ldots, j_n \in J)\ \varrho_{R^c}(j_1, \ldots, j_n) = \begin{cases} \swarrow & \text{if there exists a computation of} \\ & \langle v, \varrho(x_1, \ldots, x_n), \swarrow\rangle \text{ where } v(x_i) = j_i \\ & \text{for } i = 1, \ldots, n, \\ \nearrow & \text{otherwise;} \end{cases}$$

for the remaining functors and predicates, i.e., those defined by procedures.

One could expect that the realization R^c is a model for the system (*). The following example shows that it is not the case.

EXAMPLE 3.1. The system of procedures consists of one procedure

$$\varrho(x) \Leftrightarrow \neg\varrho(x).$$

Regardless of a given realization R its extension R^c satisfies $(\forall j \in J)\varrho_{R^c}(j) = \nearrow$. Coming back to our equivalence we see that ϱ_{R^c} is not a model for it. Obviously, this equivalence cannot possess a model owing to its inconsistency.

§ 4. Three examples illustrating the method

Our method of eliminating inconsistencies will be best illustrated by the following examples.

EXAMPLE 4.1. Let us consider the system of two procedures

$$E\varphi(x) \Leftrightarrow E\varphi(x),$$
$$\varphi(x) \Leftrightarrow \neg\varphi(x).$$

Obviously, $E\varphi_{R^c} = \varphi_{R^c}$ is again the empty set.

Consider the following two formulas

$$E\varphi(x) \Leftrightarrow E\varphi(x),$$
$$E\varphi(x) \Rightarrow \big(\varphi(x) \Leftrightarrow \neg\varphi(x)\big).$$

Now, the realization R^c is a model of these two formulas. The procedure $E\varphi$ "describes" in a sense the process of computation of the value of procedure φ. The procedure $E\varphi$ can be called the *halting procedure* of procedure φ. The second implication is valid in R^c since no computation exists. Before we describe the general construction of halting formulas and halting procedures, two more examples may be helpful in understanding the idea behind it.

EXAMPLE 4.2. Let us consider the procedure

$$f(n) = \varkappa[n = 0\ [z/1][z/n \times f(n-1)]]z$$

and the expressions

(α_1) $\quad f(2) = u,$

(α_2) $\quad f(2) = u \vee x = y.$

The realization is in the set of integers.

We first introduce a halting procedure Ef for procedure f

$$Ef(n) = \varkappa[n = 0\ [a/\mathbf{1}][a/Ef(n-1)]]a.$$

Observe that a computation of $\langle v, Ef(n), \mathbf{1}\rangle$ exists iff there exists an integer $w \in \mathscr{N}$ such that there exists a computation of $\langle v, f(n), w\rangle$.

Let us define

$E^0(\alpha_1)$: $Ef(2) \wedge f(2) \neq u$,
$E^1(\alpha_1)$: $Ef(2) \wedge f(2) = u$,
$E(\alpha_1)$: $E^0(\alpha_1) \vee E^1(\alpha_1) \Leftrightarrow Ef(2)$,
$E^0(\alpha_2)$: $\big(Ef(2) \wedge f(2) \neq u\big) \wedge x \neq y$,
$E^1(\alpha_2)$: $\big(Ef(2) \wedge f(2) = u\big) \vee x = y$,
$E(\alpha_2)$: $E^0(\alpha_2) \vee E^1(\alpha_2) \Leftrightarrow \mathbf{1}$.

Observe that a computation of $\langle v, \alpha_1, \mathbf{1}\rangle$ exists iff there exists a computation of $\langle v, E^1(\alpha_1), \mathbf{1}\rangle$.

In the construction given below we shall use procedures of specific forms to replace programs occurring in the procedures of the system (*). The following example is intended to give the idea.

EXAMPLE 4.3. Let K: $*[x < y\ [u/x\ y/y+1]]$ be a program. Let us denote by s the substitution $[u/x\ y/y+1]$. Let the construction

$$\text{if } \alpha \text{ then } \omega \text{ else } \omega'$$

be an equivalent replacing the expression $\measuredangle[\alpha[z/\omega][z/\omega']]$ where $z \notin V(\alpha)\cup V(\omega)\cup \cup V(\omega')$.

We define the system of twelve procedures

$$\begin{aligned}
&Eh^s_x(x,y,u) \Leftrightarrow \mathbf{1}, && h^s_x(x,y,u) = x,\\
&Eh^s_y(x,y,u) \Leftrightarrow \mathbf{1}, && h^s_y(x,y,u) = y+1,\\
&Eh^s_u(x,y,u) \Leftrightarrow \mathbf{1}, && h^s_u(x,y,u) = x,\\
&Eh^K_x(x,y,u) \Leftrightarrow \text{if } x<y \text{ then } \mathbf{1} \text{ else } Eh^K_x(x,y+1,x),\\
&Eh^K_y(x,y,u) \Leftrightarrow \text{if } x<y \text{ then } \mathbf{1} \text{ else } Eh^K_y(x,y+1,x),\\
&Eh^K_u(x,y,u) \Leftrightarrow \text{if } x<y \text{ then } \mathbf{1} \text{ else } Eh^K_u(x,y+1,x),\\
&h^K_x(x,y,u) = \text{if } E(K) \text{ then } \{\text{if } x<y \text{ then } x \text{ else } h^K_x(x,y+1,x)\} \text{ else } h^K_x(x,y,u);\\
&h^K_y(x,y,u) = \text{if } E(K) \text{ then } \{\text{if } x<y \text{ then } y \text{ else } h^K_y(x,y+1,x)\} \text{ else } h^K_y(x,y,u);\\
&h^K_u(x,y,u) = \text{if } E(K) \text{ then } \{\text{if } x<y \text{ then } u \text{ else } h^K_u(x,y+1,x)\} \text{ else } h^K_u(x,y,u).
\end{aligned}$$

Here $E(K)$ denotes the formula

$$Eh^K_x(x,y,u) \wedge Eh^K_y(x,y,u) \wedge Eh^K_u(x,y,u).$$

Observe the following equivalences, which hold for every variable x, y or u, for every valuation v and every value w:

1. conditions (i) and (ii) are equivalent

(i) there exists a computation of $\langle v, Kx, w\rangle$,

(ii) there exists a computation of $\langle v, h^K_x, w\rangle$.

2. conditions (iii) and (iv) are equivalent:

(iii) there exists a computation of $\langle v, Eh^K_x(x,y,u), \mathbf{1}\rangle$,

(iv) there exists a value w such that there exists a computation of $\langle v, Kx, w\rangle$.

3. conditions (v) and (vi) are equivalent

(v) the valuation $v' = K_R(v)$ is defined,

(vi) there exists a computation of $\langle v, E(K), \mathbf{1}\rangle$.

§ 5. Halting formulas and procedures

The method exemplified in the preceding section is general: namely, for every system (*) of procedures a new, normalized system (**) of procedures can be associated with (*).

Four goals are to be achieved by one simultaneous definition:

(1) an extension $\mathscr{L}''$ of the language $\mathscr{L}'$,

(2) a mapping E that associates with every expression ω of $\mathscr{L}'$ formulas denoted by $E(\omega)$, $E^1(\omega)$, $E^0(\omega)$,

(3) a mapping which with every program that occur in (*) associates a system of procedures that replace it,

(4) a mapping which with every procedure of the system associates its companion—a halting procedure.

Every procedure of the system (*) is replaced by a system of new procedures. Let the procedure $\varphi(x_1, \ldots, x_n) = M\tau$ have $y_1, \ldots, y_m$ as all the variables occurring in M and $z_1, \ldots, z_l$ as all the variables of the term τ. Then two procedures,

$$\varphi(x_1, \ldots, x_n) = \tau\big(z_1/h^M_{z_1}(y_1, \ldots, y_m), \ldots, z_l/h^M_{z_l}(y_1, \ldots, y_m)\big),$$
$$E\varphi(x_1, \ldots, x_n) \Leftrightarrow E(M\tau),$$

as well as the procedures $h^M_{y_1} \ldots h^M_{y_m}$, $Eh^M_{y_1} \ldots Eh^M_{y_m}$ associated with the program M, belong to the system (**).

The rather lengthy definition of the mapping E is omitted, see [31].

The aim of introducing normalized systems is explained by the following theorem:

5.1. *For every expression $\omega \in \mathscr{L}'$, a computation of $\langle v, \omega, w\rangle$ in the realization R with the use of the system (*) exists iff there exists a computation of $\langle v, \omega, w\rangle$ in the realization R with the use of the system (**).*

If a computation of $\langle v, \omega, w\rangle$ in the realization R with the use of (*) exists, then there exists a computation of $\langle v, E(\omega), \mathbf{1}\rangle$ and if there exists a computation of $\langle v, E(\omega), \mathbf{1}\rangle$ then for certain value w there exists a computation of $\langle v, \omega, w\rangle$.

§ 6. Computed model of procedures

We define the realization R^c as a computed extension of the realization R of the language $\mathscr{L}''$ in the set $\mathscr{J}$ and the two-element Boolean algebra B_0 in the way described in § 3 for $\mathscr{L}'$.

With the system (**) we associate the following system (w) of conditional recursive definitions (see examples 4.1, 4.2):

1. every equivalence of the form

$$E\varrho(x_1, \ldots, x_n) \Leftrightarrow \omega \quad \text{or} \quad E\varphi(x_1, \ldots, x_m) \Leftrightarrow \omega^1$$

which belongs to (**) is an element of the system (w),

2. all the remaining equalities and equivalences of the system (**) are replaced by implications according to the following scheme:

a. if the equality $\varphi(x_1, \ldots, x_m) = K\tau$ belongs to (**), then the implication $E\varphi(x_1, \ldots, x_m) \Rightarrow \{\varphi(x_1, \ldots, x_m) = K\tau\}$ is an element of (w);

b. if the equivalence $\varrho(x_1, \ldots, x_n) \Leftrightarrow M\alpha$ belongs to (**), then the implication $E\varrho(x_1, \ldots, x_n) \Rightarrow (\varrho(x_1, \ldots, x_n) \Leftrightarrow M\alpha)$ is an element of (w).

6.1. *The realization R^c is a model of the system (w) of conditional recursive definitions* (see [31]).

As was mentioned above, one can construct a system of procedures on the basis of the computations "by name". If (w^n) denotes the system of conditional recursive definitions obtained from (**) in the way indicated earlier, then after repeating the proofs we obtain

6.2. *The realization R^c_n (a computed "by name" extension of R) is a model of the system (w^n) of conditional recursive definitions* [31].

§ 7. Principle of recursion induction

We have proved that R^c is a model of the system (w) in the set $\mathscr{J}$ and the two-element Boolean algebra B_0. R^c is an extension of the realization R.

The example below shows that a system (w) can possess different models—extensions of R.

EXAMPLE 7.1. Let (w) be the following system of two formulas:

$$Ef(x, y) \Leftrightarrow \text{if } x = 0 \text{ then } \mathbf{1} \text{ else } Ef(x-1, f(x, y)) \wedge Ef(x, y)$$
$$Ef(x, y) \Rightarrow \big(f(x, y) = \text{if } x = 0 \text{ then } 2 \text{ else } f(x-1, f(x, y))\big).$$

Let R be a realization in the set of real numbers with the obvious meaning of the symbols 0, 1, 2 and $-$. The realization R^c is as follows:

$$Ef_{R^c}(j_1, j_2) = \begin{cases} \vee & \text{when } j_1 = 0, \\ \nearrow & \text{otherwise;} \end{cases} \qquad f_{R^c}(j_1, j_2) = \begin{cases} 2 & \text{when } j_1 = 0, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

It is not difficult to observe that the following extension R' of R is also a model of (w):

$$Ef_{R'}(j_1, j_2) = \begin{cases} \vee & \text{when } j_1 \in \mathscr{N}, \\ \nearrow & \text{otherwise;} \end{cases} \qquad f_{R'}(j_1, j_2) = \begin{cases} 2 & \text{when } j_1 \in \mathscr{N}, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In the sequel we shall consider the set of all extensions of the realization R which are models of (w). It will be denoted by Ext^w_R. The set Ext^w_R is ordered by the inclusion as follows. We assume that $R' \leqslant R''$ if both realizations are models of (w) and

a. for every functor $\varphi \in \mathscr{L}''$,

$\varphi_{R'} \subset \varphi_{R''}$, i.e., for all $j_1 \ldots j_m \in \mathscr{J}$ if $\varphi_{R'}(j_1 \ldots j_m)$ is defined and equal to j then $\varphi_{R''}(j_1 \ldots j_m) = j$;

b. for every predicate $\varrho \in \mathscr{L}''$,

$\varrho_{R'} \subset \varrho_{R''}$, i.e., for all $j_1 \ldots j_n \in \mathscr{J}$ $\varrho_{R'}(j_1 \ldots j_n) = \vee$ implies $\varrho_{R''}(j_1 \ldots j_n) = \vee$.

7.1. *The model R^c is the least of the models of (w) in* Ext^w_R [31].

Theorem 7.1 can be compared with the statements asserting that a function computed by a procedure is the least fixed point of that procedure treated as a functional equation.

7.2. (*Principle of recursion induction*). *Let (*) be a consistent system of procedures. Let (*) contain only functional (not relational) procedures. We shall treat (*) as a system of functional equations. Every solution R' of the system (*) (R' is a model of (*)) with the domain equal to the domain of the computed solution R^c is equal to R^c* (cf. [48]).

Here, by the domain of a model R' we obviously understand the family of sets $\{\mathrm{Dom}\,\varphi_{R'}\}_{\varphi \in \{\varphi_1, \ldots, \varphi\}}$ where

$$\mathrm{Dom}\,\varphi_{R'} = \{(j_1, \ldots, j_n) \in \mathscr{J}^n\colon \varphi_{R'}(j_1, \ldots, j_n) \text{ is defined}\}.$$

The following example shows that the assumption of the absence of relational procedures in (*) is essential.

EXAMPLE 7.2. Let us consider the following relational procedure:

$$\varrho(x, y, z) \Leftrightarrow \text{if } x = 0 \wedge z = 2 \text{ then } 1 \text{ else } \varrho(x-1, y, z) \wedge \varrho(x, y, z)$$

and two models, R^c and R', in the set of real numbers:

$$\varrho_{R^c}(j_1, j_2, j_3) = \begin{cases} \vee & \text{when } j_1 = 0 \text{ and } j_3 = 2, \\ \nearrow & \text{otherwise,} \end{cases}$$
$$\varrho_{R'}(j_1, j_2, j_3) = \vee \quad \text{for all } j_1, j_2, j_3 \in \mathscr{J}.$$

The domains of R^c and R' are equal, the models are different. This is caused by our definition of the realization R^c. From the standpoint of two-valued logic we put $\varrho_{R^c}(j_1 \ldots j_n) = \nearrow$ either if this value is computed or if no computation exists.

§ 8. Final remarks

It is not difficult to find greater or even maximal models of procedures. The following question arises in a natural way: how do we find the greatest of the effective models in Ext^w_x?

Another way to find a model of procedures is via Gentzen-style diagrams. We have no space here to give full details. The idea is as follows: (1) Mirkowska's Gentzen-style axiomatization of algorithmic logic (cf. II, § 6) is adopted and enriched. (2) Each procedure is transformed into a scheme according to the following example:

$$\varrho(\tau_1, \ldots, \tau_n) \Leftrightarrow K\alpha$$

is transformed into

$$\text{(Ra)} \quad \frac{\Gamma', s\varrho(\tau_1, \ldots, \tau_n), \Gamma'' \to \Delta}{\Gamma', s[x_1/\tau_1, \ldots, x_n/\tau_n]K\alpha, \Gamma'' \to \Delta}.$$

(3) The diagram of the relational structure described by the realization R is used to modify the notion of the fundamental sequent and also to generate further schemes.

Note. The notion of diagram used here has a different meaning from the notion of the diagram of a formula.

(4) The notion of the diagram of a formula is borrowed from [49].

Now the diagram of a formula of the form $\varphi(1, 2) = u$ can be used in order to compute the value of the procedure $\varphi(x, y)$ at $(1, 2)$. Namely, if in the diagram of $\varphi(1, 2) = u$ it is possible to turn all non-fundamental sequents into fundamental ones by one simultaneous replacement of the variable u by a constant, say c, then the value of $\varphi(1, 2)$ is assumed to be c.

It can be proved (cf. [31]) that the realization R^d defined in this way is a model of a given system of procedures. Moreover, there is no an algorithm which could produce models larger than R^d. Obviously, there are larger models but no general algorithmic procedure of constructing them exists. This can be seen from a theorem contained in [31].

From the example given by W. Dańko [7] it follows that there are functions definable by procedures which are not programmable. The relational system in the example is not a constructive one; on the contrary, in every constructive relational system, let us denote it by R, all recursively definable functions and relations, i.e., R^c, R^c_n, R^d models of the system (w), are programmable in R. From this an analogue of the Beth definability theorem can be proved:

Let all models of an algorithmic theory $\mathscr{T} = \langle \mathscr{L}, \mathscr{C}, \mathscr{A} \rangle$ be constructive relational systems. If a relation (a function) is implicitly definable in $\mathscr{T}$ by a system of procedures, then it is programmable in $\mathscr{T}$.

Another (the third) notion of computation, closer to the computer practice, can be introduced. We call those computations algolic (cf. [31]).

Chapter VI

CONSTRUCTIVE AND SEMICONSTRUCTIVE RELATIONAL SYSTEMS

Let us compare the notions of programmability and of recursiveness. It is intuitively clear that programmability is a generalization of the notion of recursiveness. This can be proved formally (Theorem 3.1). The question for which relational systems the notions of recursiveness and of programmability coincide will be solved by introducing the notion of constructive relational systems (CRS).

The aims to be achieved by introducing the notions of CRS are:

—a uniform approach to different theories of algorithms (recursive functions, Turing machines, normal algorithms of Markov, etc.).

—a generalization of the notion of a finite functionally complete relational system to the case with a denumerable universe.

—an exact estimation of recursiveness understood as a special case of programmability,

—a base for a theory of data structures relevant for (applicable) programming,

—a step toward an algebraic characterization of the notion of effectivity.

The last two sections deal with the notion of a semiconstructive relational system (SRS). In a SRS the stack mechanism can be implemented. The connection is shown between data structures an semiconstructive relational systems.

§ 1. Constructive equivalence of relational systems

A system $\mathfrak{B}$ is said to be a *strongly programmable extension* of a system $\mathfrak{A}$ if $\mathfrak{B}$ is an extension of $\mathfrak{A}$ and all the operations and relations of $\mathfrak{B}$ are strongly programmable in $\mathfrak{A}$. In other words, $\mathfrak{B}$ is a strongly programmable extension of $\mathfrak{A}$ if it arises from $\mathfrak{A}$ by adjoining certain functions and relations which are strongly programmable in $\mathfrak{A}$.

EXAMPLE 1.1. The system $\langle \mathscr{N}, 0, \mathbf{S}, +, \times, <, = \rangle$ is a strongly programmable extension of the system $\langle \mathscr{N}, 0, \mathbf{S}, = \rangle$ since the operations $+$ (addition) and $\times$ (multiplication) and the relation $<$ (less than) are strongly programmable in $\langle \mathscr{N}, 0, \mathbf{S}, = \rangle$.

We shall say that a system $\mathfrak{A}'$ is *definable within a system* $\mathfrak{A}$ iff it is a reduct of a strongly programmable extension of $\mathfrak{A}$.

EXAMPLE 1.2. The system $\langle \mathscr{N}, +, \times, < \rangle$ is definable within the system $\langle \mathscr{N}, 0, \mathbf{S}, = \rangle$.

A system $\mathfrak{A}$ is *strongly programmable on a system* $\mathfrak{B}$ iff it is isomorphic with a system $\mathfrak{A}'$ which is definable within $\mathfrak{B}$.

EXAMPLE 1.3. The system of integers $\langle \mathscr{Z}, 1, +, -, \times, = \rangle$ is strongly programmable on the system of natural numbers since it is possible to interpret even natural numbers as positive and odd natural numbers as negative and to give definitions of proper operations by non-looping programs.

Two systems $\mathfrak{A}$ and $\mathfrak{B}$ are *constructively equivalent* iff

(1) $\mathfrak{A}$ is strongly programmable on $\mathfrak{B}$ and

(2) $\mathfrak{B}$ is strongly programmable on $\mathfrak{A}$.

§ 2. Constructive relational systems

In this and the following two sections we are going to approach Church's thesis in a uniform algebraical way. The results obtained here can be interpreted also as arguments showing that the notion of computability in the sense of recursive functions theory is a restriction of the notion of function programmable in the corresponding relational system.

A relational system $\mathfrak{A}$ is said to be *constructive* iff it is constructively equivalent to the system $\langle \mathscr{N}, \mathbf{S}, 0, = \rangle$ of natural numbers.

In other words, a system $\mathfrak{A} = \langle \mathscr{J}, \{o_i\}_{i \in I}, \{r_k\}_{k \in K} \rangle$ is constructive iff

(1) there exist functions (a) g—a constant, $g \in \mathscr{J}$, (b) f: $\mathscr{J} \xrightarrow{1-1} \mathscr{J}$ such that $(\forall j \in \mathscr{J})(\exists i \in \mathscr{N})\ j = f^i(g)$;

(2) systems $\mathfrak{A}$ and $\mathfrak{B} = \langle \mathscr{J}, g, f, = \rangle$ are mutually definable in one another.

Examples of constructive relational systems.

2.1. The set of nonnegative integers with 0 (zero), S (successor), $=$ (identity).

2.2. The system of integers $\langle \mathscr{Z}, 1, +, -, = \rangle$.

2.3. The system of rational numbers $\langle \mathscr{R}, 1, +, -, \times, /, = \rangle$.

2.4. *Malcev's system.* Let $A = \{a_1, \ldots, a_m\}$ be a finite set called the *alphabet*. We shall consider the set A^* of finite sequences called *words* over the set A, together with 1-argument operations $\{f_i\}_{i=1}^m$, d and relations $\{r_i\}_{i=1}^m$, e in A. Let w denote a word from A; by λ we shall denote the empty word (the sequence of the length zero). The concatenation of words a and a' will be denoted by aa'. The primitive notions of the system $\mathfrak{M}$ are as follows:

$$f_i(w) = wa_i, \quad i = 1, \ldots, m,$$
$$d(wa) = w, \quad d(\lambda) = \lambda,$$
$$w \in r_i \quad \text{iff} \quad (\exists w' \in A^*)\, w = w'a_i, \quad i = 1, \ldots, m,$$
$$w \in e \quad \text{iff} \quad w = \lambda.$$

We first shall prove that the system $\langle A^*, \lambda, k, = \rangle$ is definable within $\mathfrak{M}$. The definitions are as follows:

$$\lambda \overset{\text{def}}{=} *\,[e(w)[w/d(w)]]\,w;$$

identity is defined by the recursive equivalence

$$w_1 = w_2 \overset{\text{def}}{\Leftrightarrow} e(w_1) \wedge e(w_2) \vee \bigvee_{i=1}^{m} r_i(w_1) \wedge r_i(w_2) \wedge d(w_1) = d(w_2),$$

which can be translated into an iterative one

$$w_1 = w_2 \Leftrightarrow \circ\big[[c/\mathbf{1}\ u_1/w_1\ u_2/w_2] * [e(u_1) \wedge e(u_2)$$
$$[c/c \wedge r_1(u_1) \wedge r_1(u_2) \vee r_2(u_1) \wedge r_2(u_2) \vee \ldots \vee r_m(u_1) \wedge r_m(u_2)\ \ u_1/d(u_1)\ \ u_2/d(u_2)]]\big]\, c.$$

The elements of A^* can be ordered in the following sequence:

$$\lambda, a_1, \ldots, a_m, a_1a_1, a_2a_2, \ldots, a_1a_m, a_2a_m, \ldots, a_ma_m, a_1a_1a_1, \ldots;$$

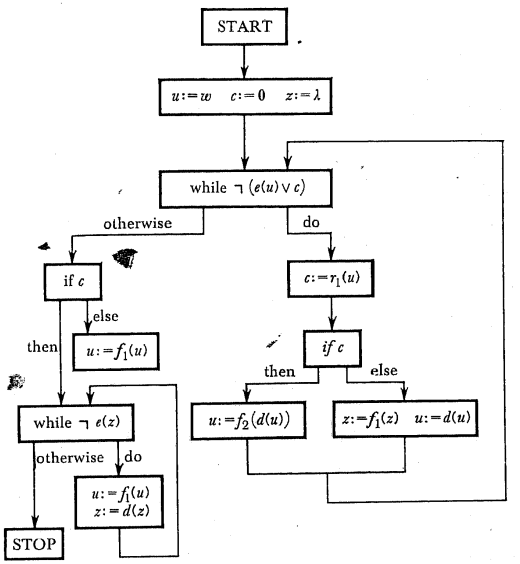
hence, the function k should satisfy the following conditions:

$$k(w) = \begin{cases} f_1(w) & \text{if} \quad w = \lambda, \\ f_{i+1}(d(w)) & \text{if} \quad w \in r_i,\ i = 1, \ldots, m-1, \\ f_1(k(d(w))) & \text{if} \quad w \in r_m. \end{cases}$$

We shall give the program defining the function k for the case $m = 2$; a generalization is easy.

$$k(w) = \circ\Big[[u/w\ c/\mathbf{0}\ z/\lambda] \circ \Big[*\Big[e(u) \vee c \circ \big[[c/r_1(u)]$$
$$\curlyvee [c[u/f_2(d(u))][z/f_1(z)\ u/d(u)]]\big]\Big]$$
$$\circ \big[\curlyvee [c[\ ][u/f_1(u)]] * [e(z)[u/f_1(u)\ z/d(z)]]\big]\Big]\, u.$$

The compatibility of this definition with the conditions established earlier can be proved from the axioms of Malcev's system. This, however, is beyond the scope of the paper. Instead, let us see the schematic representation of the program used.



2.5. *The Markov system.* The universe of a Markov system is the set $(A \cup \{\cdot\})^*$ of words over the alphabet $A \cup \{\cdot\}$. Given words w and w', we shall consider relation r_w and operation $f_w^{w'}$ (both unary):

for every word $v \in (A \cup \{\cdot\})$,

$v \in r_w$ if and only if there exist words $\alpha, \beta \in A^*$ such that $v = \alpha w \beta$,

$$f_w^{w'} v = \begin{cases} \alpha w' \beta & \text{when } v = \alpha w \beta \text{ and } \alpha \text{ is the shortest} \\ & \quad \text{word satisfying } v = \alpha w \beta, \\ v & \text{otherwise.} \end{cases}$$

A *Markov system* is thus a relational system with infinitely many relations and operations of the form described above.

Markov's limitations. We are able to use only one variable v in a program. Consequently, we shall write $f_w^{w'}$ to denote the substitution $[v/f_w^{w'}(v)]$ and r_w instead of $r_w(v)$.

All programs have to be of the following form:

$$\circ\left[*\left[r_{\cdot}\veebar\left[r_{w_1}f_{w_1}^{w_1'}\veebar\left[r_{w_1}f_{w_2}^{w_2'}\veebar[\ \dots\ [f_{\lambda}^{\lambda'}]]\right]\right]\right][f_{\cdot}^{\cdot}]\right].$$

Remark 1. It is easy to verify that such a program is in fact a normal algorithm of Markov.

Remark 2. Limitations are inessential since (1) any given n-tuple of words can be encoded as one word in a properly extended alphabet, (2) every program can be equivalently transformed to one of the normal form (cf. I, § 5).

2.6. *Turing's system.* Two alphabets are given: $A = \{a_1, \dots, a_p\}$ and $Q = \{q_0, \dots, q_n\}$. The universe of the Turing system is composed of all words w of the form $b_{q_i}b'$ where $b, b' \in A^*$, $i \in \{0, \dots, n\}$.

The relations are

$$w \in r_{km} \Leftrightarrow w = bq_ka_mb', \quad k = 0, 1, \dots, n,\ m = 1, \dots, p.$$

Let $w = ba_lq_ka_mb'$; the following operations are considered

$$R_i(w) = ba_la_mq_ib',$$
$$L_i(w) = bq_ia_la_mb',$$
$$P_{ij}(w) = ba_lq_ia_jb', \quad \text{where} \quad i = 1, \dots, n,\ j = 1, \dots, p.$$

It is not difficult to prove the constructivity of the Turing system.

Turing limitations. In a program we are able to use only one variable v. Consequently, we shall write r_{km}, R_i, L_i, P_{ij} to denote the formula $r_{km}(v)$ and the substitutions $[v/R_i(v)]$, $[v/L_i(v)]$, $[v/P_{ij}(v)]$, respectively. Let r_0 be the union of relations r_{0m}, $m = 1, \dots, p$, $r_0 = \bigcup_{m=1}^{p} r_{0m}$. Let I stand for one of the symbols R_i, L_i, P_{ij}. All Turing limited programs have to be of the following form:

$$*\left[r_0\veebar\left[r_{11}I_{11}\veebar\left[r_{12}I_{12}\veebar[\dots\veebar[r_{np}I_{np}[\ ]]\dots]\right]\right]\right].$$

The remarks from the previous example can be repeated. There exists a one-one correspondence between Turing machines and Turing limited programs. The restriction to one variable is inessential.

§ 3. Enumerated relational systems

Let $\mathfrak{A} = \langle \mathscr{J}, \{o_i\}_{i\in I}, \{r_k\}_{k\in I'}\rangle$ be a relational system. A mapping $\alpha\colon \mathscr{N} \xrightarrow{\text{onto}} \mathscr{J}$ will be called the *enumeration of* $\mathscr{J}$.

Let o be a k-argument operation. A k-argument function $g\colon \mathscr{N}^k \to \mathscr{N}$ will be called a *function representing the operation* o if the following equality holds

$$\circ(\alpha n_1, \dots, \alpha n_k) = \alpha(g(n_1, \dots, n_k))$$

for arbitrary $n_1, \dots, n_k$.

Let us notice the symmetry of the definition above; one can say that the operation o *represents the function* g *in* $\mathfrak{A}$.

The operation o is called *R-recursive in the enumeration* α if there exists an R-recursive function representing o. (R stands for one of the words: primitive-, total-, partial-).

A relation r is R-recursive in the enumeration α if there exists an R-recursive function g such that

$$g(n_1, \dots, n_k) = \begin{cases} 0 & \text{if } \langle \alpha n_1, \dots, \alpha n_k\rangle \in r, \\ 1 & \text{otherwise.} \end{cases}$$

In the case where R means *partial* we do not require

$$g(n_1 \dots n_k) = 1 \quad \text{when} \quad \langle \alpha n_1, \dots, \alpha n_k\rangle \notin r;$$

namely, relation r is *partial recursive* if there exists a partial recursive function g such that

$$g(n_1 \dots n_k) = \begin{cases} 0 & \text{if } \langle \alpha n_1 \dots \alpha n_k\rangle \in r, \\ 1 \text{ or undefined} & \text{in the opposite case.} \end{cases}$$

It is easily seen that partial-recursive relations are in fact recursively enumerable.

3.1. *Given a relational system* $\mathfrak{A} = \langle \mathscr{J}, \{o_i\}_{i\in I}, \{r_k\}_{k\in I'}\rangle$ *and an enumeration* α *of its elements. If all operations* o_i $(i \in I)$ *and all relations* r_k $(k \in I')$ *are total recursive in the enumeration* α *then all programmable operations and relations are partial-recursive in the enumeration* α.

COROLLARY. *With the assumptions of Theorem* 3.1 *every strongly programmable operation (relation) is total-recursive.*

The theorem states that the notion of programmability is a generalization of the notion of recursiveness. For every relational system it is possible to consider programmable relations and functions. If the assumption of 3.1 is satisfied, then all programmable functions and relations are recursive. Similar statements were proved by Shepherdson and Sturgis [54], and others. All of them assumed the system of natural numbers or an extension of it as the universe.

It is easy to observe that every constructive relational system satisfies the assumption of 3.1. There are other systems which satisfy the assumption and are not constructive.

EXAMPLE. The algebra of terms of a given countable signature satisfies the assumption of 3.1 if the set of variables is countable. The simple case of the example consists of a free semigroup A^* over a countable set A with a concatenation of words and the empty word λ.

§ 4. Main theorem on constructive relational systems

The fundamental property of constructive systems is expressed by the following theorem.

4.1. *A relational system $\mathfrak{A}$ is constructive if and only if there exists a one-one enumeration α of its elements such that*

(a) *every programmable (strongly programmable) function or relation is partial-recursive (total-recursive) in enumeration α,*

(b) *every recursive function of natural numbers represents a certain relation or operation which is programmable in $\mathfrak{A}$. If the function in question is total-recursive, then the corresponding relation or operation is strongly programmable* [31].

§ 5. Products of constructive relational systems

It seems quite natural to ask what kinds of algebraic operations performed on constructive relational systems lead again to constructive systems. We shall first observe

5.1. *The Cartesian product of constructive relational systems is not itself a constructive system* [31].

The theorem indicates the need of a product operation which is performable on any two relational systems even if their signatures differ and which preserves the property of being constructive.

Let $\mathfrak{A}$ and $\mathfrak{B}$ be relational systems

$$\mathfrak{A} = \langle A, \{f_i\}_{i\in I}, \{r_j\}_{j\in J}\rangle,$$
$$\mathfrak{B} = \langle B, \{g_k\}_{k\in K}, \{p_l\}_{l\in L}\rangle.$$

We introduce relations and partial operations in the set $A\times B$, the relational system obtained in this way will be called the *full product* $\mathfrak{A}\underset{f}{\times}\mathfrak{B}$ of $\mathfrak{A}$ and $\mathfrak{B}$.

For every $j\in J$, an m_j-ary relation $\hat{r}_j$ in $(A\times B)^{m_j}$ is defined as follows

$$\langle (a_1,b_1),\ldots,(a_{m_j},b_{m_j})\rangle \in \hat{p}_l \Leftrightarrow (a_1,\ldots,a_{m_j})\in r_j;$$

similarly, for every $l\in L$,

$$\langle (a_1,b_1),\ldots,(a_{m_l},b_{m_l})\rangle \in \hat{r}_j \Leftrightarrow (b_1,\ldots,b_{m_l})\in p_l.$$

For every $i\in I$ we define a partial n_i-ary operation $\hat{f}_i$ in $A\times B$ as follows:

the value of $\hat{f}_i((a_1,n_1),\ldots,(a_{n_i},b_{n_i}))$ is defined iff $b_1=\ldots=b_{n_i}$ and if it is defined then is equal to $(f_i(a_1,\ldots,a_{n_i}),b_1)$.

For every $k\in K$ we define a partial n_k-ary operation g_k in $A\times B$ as follows:

the value of $\hat{g}_k((a_1,b_1),\ldots,(a_{n_k},b_{n_k}))$ is defined iff $a_1=\ldots=a_{n_k}$ and if it is defined then is equal to $(a_1, g_k(b_1,\ldots,b_{n_k}))$.

Constants, i.e. zero-argument operations, are treated in a slightly different way: namely, any pair of constants from the set $A\times B$ is a constant in the system $\mathfrak{A}\underset{f}{\times}\mathfrak{B}$.

Remark. Let $\mathfrak{A}$ and $\mathfrak{B}$ be two systems. If there exists a Cartesian product $\mathfrak{A}\times\mathfrak{B}$, i.e. if the two systems are similar, then the system $\mathfrak{A}\times\mathfrak{B}$ is definable within $\mathfrak{A}\underset{f}{\times}\mathfrak{B}$.

5.2. *A full product of constructive relational systems is a constructive system* [31].

Let $\mathfrak{A}$ and $\mathfrak{B}$ be two relational systems. By the *disjoint sum* $\mathfrak{A}\sqcup\mathfrak{B}$ of the systems $\mathfrak{A}$ and $\mathfrak{B}$ we shall understand the following relational system: the universe of the system $\mathfrak{A}\sqcup\mathfrak{B}$ is the $A\sqcup B$—disjoint sum of the universes A and B, operations in $\mathfrak{A}\sqcup\mathfrak{B}$ are partial operations defined on the elements of the corresponding subset of the sum $A\sqcup B$, relations are defined in an obvious way.

From this as an easy consequence follows

5.3. *A disjoint sum of constructive systems is a constructive system* [31].

§ 6. Semiconstructive relational systems

In this section we shall consider a certain class of relational systems containing all constructive relational systems. As we shall see, the systems of this class will be characterized by possessing the stack mechanism.

A relational system $\mathfrak{A}=\langle A, \{o_i\}_{i\in I}, \{r_j\}_{j\in J}\rangle$ is called *semiconstructive* provided the following conditions are fulfilled:

(1) the unary relation defining a certain subset $S\subset A$ is strongly programmable in $\mathfrak{A}$;

(2) the operations binary c, unary h, t and constant e are strongly programmable in $\mathfrak{A}$;

(3) for every natural number $n\in\mathcal{N}$, the n-ary operation $(x_1,\ldots,x_n)$ is strongly programmable in $\mathfrak{A}$ and the following conditions are satisfied:

(30) every element $s\in S$ can be uniquely represented as $(x_1,\ldots,x_n)$ for some $x_1,\ldots,x_n$ in A,

(31) $(\,)=e$,

(32) $h(e)=e$ and $h((x_1,\ldots,x_n))=x_1$ for $n\geqslant 1$,

(33) $t(e)=e$ and $t((x_1,\ldots,x_n))=(x_2,\ldots,x_n)$ for $n\geqslant 1$,

(34) $c(x_1,(x_2,\ldots,x_n))=(x_1,x_2,\ldots,x_n)$.

The standard model of arithmetic $\langle \mathcal{N}, S, 0, = \rangle$ is an example of a semiconstructive system. In the next section we shall consider systems which are semiconstructive but not constructive.

6.1. *Every constructive system is semiconstructive.*

As a consequence of possessing the stack mechanism we obtain

6.2. *In every semiconstructive relational system, every system of non-functional procedures with parameters called by value can be replaced by a system of non-functional, non-recursive procedures.*

At this moment the question arises whether we can accept Church's thesis in the following form:

In every semiconstructive relational system, every effective algorithm can be represented as an FS-program.

The results of Dańko [8] show that this formulation of Church's thesis does not hold for every relational system.

The next lemma shows that condition (3) can be replaced by a condition not referring to n-ary operations $(x_1, \ldots, x_n)$.

6.3. *A relational system $\mathfrak{A}$ is semiconstructive iff it satisfies* (1), (2) *and the following condition*:

(4) *for every $x \in A$ and $y \in S$,*

(41) $\neg c(x, y) = e$,

(42) $y \neq e \Rightarrow \big(c(h(y), t(y)) = y\big)$,

(43) $h\big(c(x, y)\big) = x \cap h(e) = e$,

(44) $t\big(c(x, y)\big) = y \cap t(e) = e$,

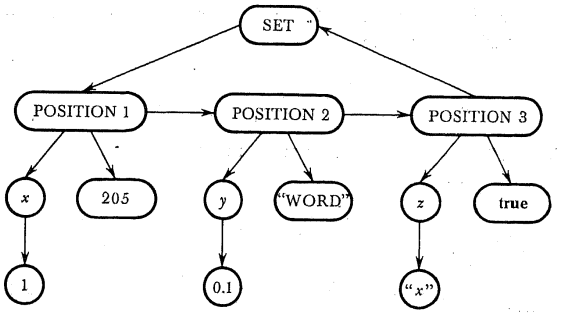
(45) $\bigcup [y/t(y)](y = e)$.

The operations mentioned in (3) can be defined as follows: $( ) = e$ and $(x_1, x_2, \ldots, x_n) = c\big(x_1, c\big(x_2, \ldots, c(x_n, e), \ldots\big)\big)$.

By these definitions if $s \in S - \{e\}$ then $s = \big(h(s), h\big(t(s)\big), \ldots, h\big(t^{n-1}(s)\big)\big)$ where n is the least natural number such that $t^n(s) = e$. The existence of such an n results from condition (45).

§ 7. Relational systems of data structures

We can conceive a data structure as a finite directed graph. The terminal vertices can be identified with atoms of information. The non-terminal vertices can be identified with the names of fields of machine memory. For example,



We shall choose another equivalent way of defining the notion of a data structure (see [4]). The definition will have a linguistic character oriented for input-output processes.

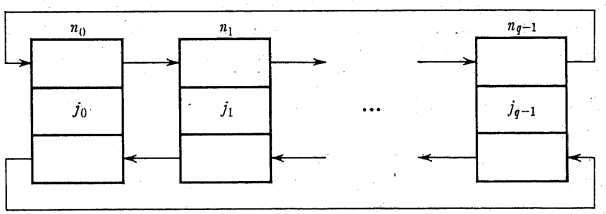
We shall form data structures over a finite nonempty set J of objects (entities of information) and over an enumerable set N of names (of memory locations) disjoint with J. By a *form* we shall understand either a name or an object or the empty form $[\ ]$ or a form $[\varphi_1 \varphi_2 \ldots \varphi_k]$ where in turn $\varphi_1, \ldots, \varphi_k$ are forms. By a *memory state* we shall mean a finite sequence of pairs $n_1 \to \varphi_1, \ldots, n_k \to \varphi_k$ where

(1) $n_1, \ldots, n_k$ are distinct names,

(2) $\varphi_1, \ldots, \varphi_k$ are forms different from names and such that every name occurring in any form φ_i appears also among the $n_1, \ldots, n_k$.

By a *data structure* we shall mean the pair $l = \langle \varphi; m \rangle$ where φ is a form, $m = n_1 \to \varphi_1, \ldots, n_k \to \varphi_k$ is a memory state and, moreover, every name occurring in φ appears also among the $n_1, \ldots, n_k$.

EXAMPLE. The ring



can be represented as the following data structure

$$l = \langle n_0;\ \{n_i \to [n_{(i-1) \bmod q}\, j_i\, n_{(i+1) \bmod q}]\}_{0 \leqslant i < q} \rangle$$

where $n_0, n_1, \ldots, n_{q-1}$ are names of fields, $j_0, j_1, \ldots, j_{q-1}$ are objects.

In the set L^m we shall distinguish atoms $a_j = \langle j; m \rangle$ for $j \in J$. Now we shall define LISP-like operations in the set L^m. Let $m = n_1 \to \varphi'_1, \ldots, n_{k'} \to \varphi'_{k'}$ be a mem-

ory state and let $l = \langle \varphi; m\rangle$, $l' = \langle \psi; m\rangle$ be data structures over m. We need the following auxiliary function:

$$\vartheta(l) = \begin{cases} \varphi & \text{if} \quad \varphi \notin N, \\ \varphi_i' & \text{if} \quad \varphi = n_i \text{ for some } i = 1, \ldots, k'. \end{cases}$$

Let e, h, t, c be the following operations in L^m:

$$e = \langle [\ ]; m\rangle$$

$$h(l) = \begin{cases} \langle \varphi_1; m\rangle & \text{if} \quad \vartheta(l) = [\varphi_1 \ldots \varphi_k] \text{ and } k \geqslant 1, \\ e & \text{otherwise;} \end{cases}$$

$$t(l) = \begin{cases} \langle [\varphi_2 \ldots \varphi_k]; m\rangle & \text{if} \quad \vartheta(l) = [\varphi_1 \ldots \varphi_k] \text{ and } k \geqslant 1, \\ e & \text{otherwise;} \end{cases}$$

$$c(l', l) = \begin{cases} \langle [\psi\varphi_1 \ldots \varphi_k]; m\rangle & \text{if} \quad \vartheta(l) = [\varphi_1 \ldots \varphi_k] \text{ and } k \geqslant 0, \\ \langle [\psi]; m\rangle & \text{otherwise.} \end{cases}$$

By a *relational system of data structures over the state m* we shall understand the system $DS^m = \langle L^m, \{a_j\}_{j\in J} \cup \{e, h, t, c\}\rangle$. When $m = \emptyset$ is the empty state, the system $DS^{\emptyset}$ is called the *relational system of trees*.

7.1. *For every state m, DS^m is a semiconstructive relational system* (*with e, h, t, c as the required operations, S being* $DS^m - \{a_j\}_{j\in J}$).

The system DS^m is constructive only in the case of $m = \emptyset$ (i.e., *only in the case of a system of trees*).

Chapter VII

MULTIPLE-VALUED EXTENSIONS OF ALGORITHMIC LOGIC

The idea of extending systems of algorithmic logic in such a way that programs with recursive procedures would be expressions of their formalized languages may be realized by applying ω^+-valued logic and by an approach to the programs by means of Mazurkiewicz's algorithms, as proposed in [33]. The first attempt at such a solution has been formulated in [25] and [24]. In the first part of this chapter a modified version (see [26]) will be presented, to be called extended algorithmic logic (*EAL*).

It is worth mentioning that ω^+-valued logic is only used in ***EAL*** to describe the control functions of push-down algorithms, which are realizations of procedures, whereas their actions are described by means of two-valued logic. This means that neither m-valued predicates nor m-valued propositional variables for $m > 2$ occur in formalized languages of ***EAL***. On the other hand, they contain label variables and label constants interpreted in ω^+-valued logic. The iteration sign $*$ is replaced by the sign o* of a procedure operator and the iteration quantifiers are replaced by the infinite disjunction sign and by the infinite conjunction sign. Generalized formulas may have infinite length.

The generalized Post algebra $\mathfrak{P}_\omega$ of order ω^+, which for the logic under consideration plays an analogous part to that of the two-element Boolean algebra for classical logic, is introduced in § 1. The formalized languages of ***EAL*** and their realizations are presented in § 2. The ***FS***-expressions are equivalent to certain procedures (corresponding to finite control algorithms) in the formalized languages of ***EAL***. This question is discussed in § 3. The fundamental theorems concerning programs and generalized terms of ***EAL***, including a theorem on the normal form of a program, are formulated in § 4. § 5 is devoted to a formalization of the systems of ***EAL*** with a completeness theorem.

In § 6 m-valued relations, for $m \geqslant 2$, are discussed. It is shown that they may be treated as certain $(m-1)$-element sequences of characteristic functions of relations in the usual sense.

§§ 7 and 8 are devoted to a presentation of ω^+-valued algorithmic logic. It is an extension of algorithmic logic which includes expressions interpreted as **case... of begin...end** statements. In the formalized languages of this logic there may occur n-valued predicates and n-valued propositional variables for all $n \geqslant 2$. If $2 \leqslant n \leqslant m$ for an established integer m, then we obtain a mixed-valued algorithmic logic with logical values restricted to m. With regard to the connection between n-valued relations and relations in the usual sense, as indicated in § 6, the multiple-valued extensions of algorithmic logic mentioned above may be considered as formalisms which permit a complex treatment of two-valued predicates, and accordingly enable us to write complicated programs and generalized formulas in a simpler way.

A formalization of systems of the ω^+-valued algorithmic logic and those of mixed-valued algorithmic logic is given in § 8. The completeness theorems and a theorem establishing relationships between systems of ω^+-valued algorithmic logic and systems of mixed-valued algorithmic logics with logical values restricted by an $m \geqslant 2$ are also formulated. The last theorem may be considered as a weak separation theorem for the ω^+-valued algorithmic logic.

The investigations in Chapters I–VI may be extended to systems discussed in §§ 7 and 8.

It is worth mentioning that the construction of systems of ***EAL*** may be easily extended to systems of ω^+-valued algorithmic logic and those of mixed-valued algorithmic logics.

§ 1. The generalized Post algebra $\mathfrak{P}_\omega$ of order ω^+

Let $\mathfrak{B}_0 = \langle B_0, \cup, \cap, \rightarrow, -\rangle$, where $B_0 = \{\wedge, \vee\}$, be the two-element Boolean algebra. Consider an infinite chain

$$\wedge = e_0 \leqslant e_1 \leqslant \ldots \leqslant e_\omega = \vee$$

of order ω^+, i.e., isomorphic to the chain of ordinals less than or equal to ω.

Extend the operations $\cup$, $\cap$, $\to$, $-$ on this chain as follows:

$$e_i \cup e_j = e_{\max(i,j)}, \quad e_i \cap e_j = e_{\min(i,j)}, \tag{1}$$

$$e_i \to e_j = \begin{cases} \top & \text{if } i \leqslant j, \\ e_j & \text{if } i > j; \end{cases} \qquad -e_i = e_i \to e_0 = \begin{cases} \top & \text{if } i = 0, \\ \bot & \text{if } i \neq 0. \end{cases} \tag{2}$$

The algebra $\langle \mathscr{P}_\omega, \top, \cup, \cap, \to, - \rangle$, where $\mathscr{P}_\omega = \{e_i\}_{0 \leqslant i \leqslant \omega}$, is a linear pseudo-Boolean algebra (see e.g. [49]).

In the sequel let $\mathscr{N}$ stand for the set of non-negative integers and $\mathscr{N}_0$ for the set of positive integers.

In the algebra defined above introduce one-argument operations d_i, $i \in \mathscr{N}_0$, by adopting the following definition

$$d_i(e_j) = \begin{cases} \top & \text{for } i \leqslant j, \\ \bot & \text{for } i > j; \end{cases} \qquad 0 \leqslant j \leqslant \omega. \tag{3}$$

The algebra $\mathfrak{P}_\omega$ is then defined as follows:

$$\mathfrak{P}_\omega = \langle \mathscr{P}_\omega, \top, \cup, \cap, \to, -, (d_i)_{i \in \mathscr{N}_0}, (e_i)_{0 \leqslant i \leqslant \omega} \rangle.$$

It is easy to see that in $\mathfrak{P}_\omega$ the following equations hold:

$$\begin{gathered} d_i(e_j \cup e_k) = d_i(e_j) \cup d_i(e_k), \quad d_i(e_j \cap e_k) = d_i(e_j) \cap d_i(e_k), \\ d_i(e_j \to e_k) = \big(d_1(e_j) \to d_1(e_k)\big) \cap \ldots \cap \big(d_i(e_j) \to d_i(e_k)\big), \\ d_i(-e_j) = -d_1(e_j). \end{gathered} \tag{4}$$

Moreover,

$$d_1(e_j) \geqslant d_2(e_j) \geqslant \ldots \tag{5}$$

Notice that by (3)

$$d_i(\bot) = d_i(e_0) = \bot \quad \text{and} \quad d_i(\top) = d_i(e_\omega) = \top \quad \text{for each } i \in \mathscr{N}_0,$$

thus the operations d_i, $i \in \mathscr{N}_0$, are the identity mappings on $\{\top, \bot\}$.

Let us set for each $m \geqslant 2$, $\mathscr{P}_m = \{e_0, \ldots, e_{m-2}, e_\omega\}$. The algebras $\mathfrak{P}_m = \langle \mathscr{P}_m, \top, \cup, \cap, \to, -, d_1, \ldots, d_{m-1}, e_0, \ldots, e_{m-2}, e_\omega \rangle$, being subalgebras of the reducts

$$\langle \mathscr{P}_\omega, \top, \cup, \cap, \to, -, d_1, \ldots, d_{m-1}, e_0, \ldots, e_{m-2}, e_\omega \rangle$$

of $\mathfrak{P}_\omega$, are m-element Post algebras of order $m \geqslant 2$. In particular,

$$\mathfrak{P}_2 = \langle \{e_0, e_\omega\}, \top, \cup, \cap, \to, -, d_1, e_0, e_\omega \rangle,$$

where d_1 is the identity mapping and $\langle \{e_0, e_\omega\}, \cup, \cap, \to, - \rangle$ is the two-element Boolean algebra.

The following operations j_i, $i \in \mathscr{N}$, are definable in $\mathfrak{P}_\omega$

$$j_i(e_k) = \begin{cases} \top & \text{if } i = k, \\ \bot & \text{if } i \neq k; \end{cases} \qquad 0 \leqslant k \leqslant \omega, \tag{6}$$

the definitions being thus

$$j_0(e_k) = -d_1(e_k), \quad j_i(e_k) = -d_{i+1}(e_k) \cap d_i(e_k), \quad \text{for} \quad i \in \mathscr{N}_0. \tag{7}$$

Observe that $\mathfrak{P}_\omega$ is isomorphic to the algebra of all decreasing sequences $(b_1, b_2, \ldots)$ of elements in $B_0 = \{\bot, \top\}$, the operations $\cup, \cap, \to, -$, d_i for $i \in \mathscr{N}_0$ being defined as follows: for any $b = (b_1, b_2, \ldots)$ and $c = (c_1, c_2, \ldots)$

$$\begin{gathered} b \cup c = (b_1 \cup c_1, b_2 \cup c_2, \ldots), \quad b \cap c = (b_1 \cap c_1, b_2 \cap c_2, \ldots), \\ b \to c = \big(b_1 \to c_1, (b_1 \to c_1) \cap (b_2 \to c_2), \ldots\big), \\ -b = (-b_1, -b_1, \ldots), \quad d_i(b) = (b_i, b_i, \ldots). \end{gathered}$$

Clearly,

$$\bot = e_0 = (\bot, \bot, \ldots), \ \ldots, \ e_i = (\underbrace{\top, \ldots, \top}_{i\text{-times}}, \bot, \ldots), \ \ldots, \ \top = e_\omega = (\top, \top, \ldots).$$

For the theory of generalized Post algebras of order ω^+ the reader is referred to [50].

§ 2. Formalized languages of the extended algorithmic logic (*EAL*) and their realizations

Roughly speaking, the alphabets of the formalized languages of ***EAL*** differ from those of algorithmic logic by the elimination of the iteration sign $*$ and of the iteration quantifiers, and by adopting new sets of signs: a set of label variables, a set of label constants, and moreover by introducing a procedure operation sign $\circ^*$, the infinite disjunction sign $\bigvee$ and the infinite conjunction sign $\bigwedge$.

Instead of ***FS***-expressions there occur in the formalized languages of ***EAL*** $F_L S$-expressions interpreted as programs with recursive procedures. Generalized formulas in these languages describe properties of programs, as in the case of algorithmic logic.

More exactly, any alphabet A of a formalized language of ***EAL*** is the union of disjoint, at most enumerable sets

$$A = V_i \cup V_o \cup V_L \cup \Phi \cup P \cup L_0' \cup L_1' \cup L_2 \cup \mathrm{Ic} \cup \Pi' \cup U_0,$$

where V_i, V_o, Φ, P, L_2 are defined as in I, § 2; $U_0 = \{[,], /\}$; the elements of V_o, i.e., the propositional variables will be denoted in this chapter by p, q with indices if necessary; $V_L = \{a_i\}_{i \in \mathscr{N}_0}$ and a_i for $i \in \mathscr{N}_0$ are called *label variables*; $L_0' = \{E_i\}_{0 \leqslant i \leqslant \omega}$, where E_0 and E_ω occur, respectively, instead of **0** and **1** in algorithmic languages, and E_i for $0 < i < \omega$ are called *label constants*;

$$L_1' = \{\neg\} \cup \{D_i\}_{i \in \mathscr{N}_0}; \quad \mathrm{Ic} = \{\bigvee; \bigwedge\}; \quad \Pi' = \{\circ, \leftarrow, \circ^*\}.$$

By a *formalized language of **EAL** under A* we shall understand the system

$$\mathscr{L} = \langle A, T \cup F^o \cup S \cup F_L S \cup F_L ST \cup F_L SF \rangle,$$

where the sets T, F^o and S are defined as in algorithmic languages (see I, § 2) and $T \cup F^o \cup S \cup F_L S \cup F_L ST \cup F_L SF$ is the set of all well-formed expressions in $\mathscr{L}$.

The definitions of the sets $F_L S$, $F_L ST$ and $F_L SF$ will be given simultaneously with the definitions of realizations of expressions belonging to these sets.

By a *realization of $\mathscr{L}$ in a set $U \neq \emptyset$* we shall mean, as in the case of any algorithmic language, a mapping R assigning to each $\varphi \in \Phi_k$, $k \in \mathscr{N}$, a function φ_R: $U^k \to U$, and to each $\varrho \in P_k$, $k \in \mathscr{N}_0$, a function ϱ_R: $U^k \to \{\swarrow, \nearrow\}$. Let W_U be the set of all valuations of free individual variables and propositional variables. By a *label valuation* we shall mean any v_L: $V_L \to \{e_i\}_{i \in \mathscr{N}}$ such that

(1) there is $n \in \mathscr{N}_0$, such that $v_L(a_n) = e_0$,
(2) for each $n \in \mathscr{N}_0$, if $v_L(a_n) = e_0$, then $v_L(a_{n+1}) = e_0$.

The set of all label valuations will be denoted by W_L. We shall often identify a certain $v_L \in W_L$ with the sequence $(v_L(a_1), v_L(a_2), \ldots)$. Observe that any valuation is either $(e_0, e_0, \ldots)$ or $(e_{k_1}, \ldots, e_{k_n}, e_0, e_0, \ldots)$, where $k_i \neq 0$ for $i = 1, \ldots, n$.

Any pair $(v_L, v) \in W_L \times W_U$ will be said to be a *state*.

Let us set

(1) $\tau_R(v_L, v) = \tau_R(v)$ for each term $\tau \in \boldsymbol{T}$, and
$\alpha_R(v_L, v) = \alpha_R(v)$ for $\alpha \in \boldsymbol{F}^o$.

Similarly, let us set

(2) $s_R(v_L, v) = (v_L, s_R(v))$ for each substitution $s \in \boldsymbol{S}$.

Atomic $\boldsymbol{F}_L\boldsymbol{S}$-expressions are substitutions in $\boldsymbol{S}$, label substitutions and label supervisors.

The set $\boldsymbol{S}_L^0$ of label substitutions consists of the following expressions:

(s_L1) $[a_1/E_{k_1}\, a_2/a_1]$, $k_1 \neq 0$,
(s_L2) $[a_1/E_{k_1}, \ldots, a_n/E_{k_n}\, a_{n+1}/a_2]$, $n \geqslant 1$, $k_i \neq 0$ for $i = 1, \ldots, n$,
(s_L3) $[a_1/a_2]$,
(s_L4) $[a_1/E_0]$,
(s_L5) $[\,]$.

$\boldsymbol{S}_L$ will denote the set of all label substitutions except (s_L4).

Realization R is now extended on $\boldsymbol{S}_L^0$ as follows. For any $s^* \in \boldsymbol{S}_L^0$, s_R^* is a mapping s_R^*: $W_L \times W_U \to W_L \times W_U$ defined thus:

$$s_R^*(v_L, v) = (v_L', v),$$

where

($s_L r1$) $v_L' = (e_{k_1}, v_L(a_1), v_L(a_2), \ldots)$, if s^* is in form (s_L1),
($s_L r2$) $v_L' = (e_{k_1}, \ldots, e_{k_n}, v_L(a_2), v_L(a_3), \ldots)$, if s^* is in form (s_L2),
($s_L r3$) $v_L' = (v_L(a_2), v_L(a_3), \ldots)$, if s^* is in form (s_L3),
($s_L r4$) $v_L' = (e_0, e_0, \ldots)$, if s^* is in form (s_L4),
($s_L r5$) $v_L' = v_L$, if s^* is in form (s_L5).

Since, for any s^* in $\boldsymbol{S}_L^0$, s_R^* does not depend on $v \in W_U$ and does not change $v \in W_U$, it may be treated as a mapping from W_L into W_L. Therefore we shall also write $s_R^*(v_L) = v_L'$ and $s_R^*(v_L, v) = (s_R^*(v_L), v)$.

Assume that

(3) $J_0 a_n$ will be written instead of $\neg D_1 a_n$, for $n \in \mathscr{N}_0$,

and

(4) $J_k a_n$ will be written instead of $(\neg D_{k+1} a_n \wedge D_k a_n)$, for $n \in \mathscr{N}_0$, $k \in \mathscr{N}_0$.

The set $\boldsymbol{Sp}$ of label supervisors consists of the following expressions:

(sp) $[J_k a_n]$, $k \in \mathscr{N}$, $n \in \mathscr{N}_0$.

We extend R on $\boldsymbol{Sp}$ by adopting the following definition:

(spr) $[J_k a_n]_R(v_L, v) = \begin{cases} (v_L, v) & \text{if } j_k(v_L(a_n)) = \swarrow, \text{ i.e., if } v_L(a_n) = e_k, \\ \text{undefined} & \text{otherwise.} \end{cases}$

Thus $[J_k a_n]_R$ is the identity mapping on the set of states (v_L, v) for which $v_L(a_n) = e_k$ and is undefined on any other state.

On applying $\circ$, $\veebar$, $\circ^*$ we form of atomic $\boldsymbol{F}_L\boldsymbol{S}$-expressions certain new $\boldsymbol{F}_L\boldsymbol{S}$-expressions, to be called *instructions* and *procedures*.

The set $\boldsymbol{I}_1$ of instructions of order 1 is the set of the following expressions:

(i11) $\circ[\circ[[J_k a_1] s^*] s]$, where $k \in \mathscr{N}_0$, $s^* \in \boldsymbol{S}_L$, $s \in \boldsymbol{S}$,

(i12) $\circ[[J_k a_1] \veebar [\alpha \circ [s_1^* s_1] \circ [s_2^* s_2]]]$, where $k \in \mathscr{N}_0$, $\alpha \in \boldsymbol{F}^o$, $s_1^*, s_2^* \in \boldsymbol{S}_L$, $s_1, s_2 \in \boldsymbol{S}$.

If $H \in \boldsymbol{I}_1$, is in form (i11), then we define

(ir11) $H_R(v_L, v) = \begin{cases} s_R(s_R^*([J_k a_1]_R(v_L, v))) & \text{if this is defined,} \\ \text{undefined} & \text{in the opposite case.} \end{cases}$

It follows from (spr), the definition of s_R^* for s^* in $\boldsymbol{S}_L$ and (2) that

$$H_R(v_L, v) = \begin{cases} (s_R^*(v_L), s_R(v)) & \text{if } v_L(a_1) = e_k, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

If $H \in I_1$ is in form (i12), then we define

(ir12) $H_R(v_L, v) = \begin{cases} s_{1R}(s_{1R}^*([J_k a_1]_R(v_L, v))) & \text{if this is defined and } \alpha_R(v) = \swarrow, \\ s_{2R}(s_{2R}^*([J_k a_1]_R(v_L, v))) & \text{if this is defined and } \alpha_R(v) = \nearrow, \\ \text{undefined} & \text{otherwise.} \end{cases}$

It follows from (spr), the definitions of s_R^* for $s^* \in \boldsymbol{S}_L$ and (2) that

$$H_R(v_L, v) = \begin{cases} (s_{1R}^*(v_L), s_{1R}(v)) & \text{if } v_L(a_1) = e_k \text{ and } \alpha_R(v) = \swarrow, \\ (s_{2R}^*(v_L), s_{2R}(v)) & \text{if } v_L(a_1) = e_k \text{ and } \alpha_R(v) = \nearrow, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In the case of instructions (i11) and (i12), e_k will be said to be their *label*.

The set $\boldsymbol{P}_1$ of procedures of order 1 is the set of expressions

(p1) $\circ^*[H_{k_1} H_{k_1} \ldots H_{k_n} H_i]$, $n \geqslant 1$,

where

(i) $H_{k_1}, \ldots, H_{k_n} \in I_1$ and have different labels $e_{k_1}, \ldots, e_{k_n}$,

(ii) E_t does not occur in $H_{k_1}, \ldots, H_{k_n}$, and e_t is called a *terminal label* of this procedure,

(iii) $H_{k_1 t} = \circ\big[\circ[[J_{k_1} a_1][a_1/E_{k_1}\ a_2/E_t\ a_3/a_2]][\ ]\big]$ and is called a *preparatory instruction*,

(iv) $H_t = \circ\big[\circ[[J_t a_1][a_1/a_2]][\ ]\big]$ and is called a *terminal instruction*.

If H in $\boldsymbol{P}_1$ is in form (p1), then e_{k_1} is said to be its *label*.

It follows from (ir11) that

$$H_{k_1 tR}(v_L, v) = \begin{cases} \big((e_{k_1}, e_t, v_L(a_2), v_L(a_3) \ldots), v\big) & \text{if} \quad v_L(a_1) = e_{k_1}, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$H_{tR}(v_L, v) = \begin{cases} \big((v_L(a_2), v_L(a_3), \ldots), v\big) & \text{if} \quad v_L(a_1) = e_t, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In order to extend the realization R on $\boldsymbol{P}_1$ we introduce the notion of a computation of any H in $\boldsymbol{P}_1$.

If H in $\boldsymbol{P}_1$ is in form (p1), then a computation of H by realization R for a state (v_L, v) is understood as any finite sequence of states

(c) $\quad H_R^0(v_L, v) = (v_L^0, v^0), \ldots, H_R^{\overline{m}+1}(v_L, v) = (v_L^{\overline{m}+1}, v^{\overline{m}+1})$

such that

(c1) $\quad (v_L^0, v^0) = H_{k_1 tR}(v_L, v)$,

(c2) $\quad$ for each $0 \leqslant i \leqslant \overline{m}-1$, $(v_L^{i+1}, v^{i+1}) = H_{k_j R}(v_L^i, v^i)$ for some $j = 1, \ldots, n$,

(c3) $\quad (v_L^{\overline{m}+1}, v^{\overline{m}+1}) = H_{tR}(v_L^{\overline{m}}, v^{\overline{m}})$,

(c4) $\quad$ all states in (c) are defined.

The number $\overline{m}$ is said to be the *length of a computation* (c).

The following statement holds (see [26]).

2.1. *For every $H \in \boldsymbol{P}_1$, realization R in any set $U \neq \emptyset$ and state $(v_L, v) \in W_L \times W_U$, there is at most one computation of H by R for (v_L, v) and it is effectively defined. Moreover, if* (c) *is that computation, then* $v_L^{\overline{m}+1} = (v_L(a_2), v_L(a_3), \ldots)$.

Now we define for $H \in \boldsymbol{P}_1$

$$\text{(pr1)} \quad H_R(v_L, v) = \begin{cases} H_R^{\overline{m}+1}(v_L, v) & \text{if (c) is a computation of } H \text{ by } R \text{ for } (v_L, v), \\ \text{undefined} & \text{if a computation of } H \text{ by } R \text{ for } (v_L, v) \text{ does not exist.} \end{cases}$$

EXAMPLE. The following procedure $H \in \boldsymbol{P}_1$ is an implementation of a recursive program $F(x) \Leftarrow$ **if** $x = 0$ **then** 1 **else** $x.\ F(x-1)$, over $\mathcal{N}$.

$$H = \circ^*[H_{13} H_1 H_2 H_3],$$

where

$$H_{13} = \circ\big[\circ\big[[J_1 a_1][a_1/E_1\ a_2/E_3\ a_3/a_2]\big][\ ]\big],$$

$$H_1 = \circ\Big[[J_1 a_1] \preceq \big[x = 0 \circ [[a_1/a_2][y/1]] \circ \big[[a_1/E_1\ a_2/E_2\ a_3/a_2][x/x-1]\big]\big]\Big],$$

$$H_2 = \circ\big[\circ[[J_2 a_1][a_1/a_2]][x/x+1\ \ y/(x+1)\cdot y]\big],$$

$$H_3 = \circ\big[\circ[[J_3 a_1][a_1/a_2]][\ ]\big].$$

H_{13} is a preparatory instruction in H and H_3 is a terminal instruction. Let R be the standard realization in $\mathcal{N}$. The following sequence of states is an example of a computation of H by R for $(v_L, v) \in W_L \times W_N$, where $v_L = (e_1, e_2, e_3, e_0, \ldots)$ and $v(x) = 3$, $v(y) \in \mathcal{N}$, $v(z) \in \mathcal{N}$ for $z \in V_i$. In this computation we shall only write the values of variables in $V_i \cup V_0$ which occur in H.

$v_L^0 = (e_1, e_3, e_2, e_3, e_0, \ldots)$,	$v^0(x) = 3,\ \ v^0(y) = v(y)$,
$v_L^1 = (e_1, e_2, e_3, e_2, e_3, e_0, \ldots)$,	$v^1(x) = 2,\ \ v^1(y) = v(y)$,
$v_L^2 = (e_1, e_2, e_2, e_3, e_2, e_3, e_0, \ldots)$,	$v^2(x) = 1,\ \ v^2(y) = v(y)$,
$v_L^3 = (e_1, e_2, e_2, e_2, e_3, e_2, e_3, e_0, \ldots)$,	$v^3(x) = 0,\ \ v^3(y) = v(y)$,
$v_L^4 = (e_2, e_2, e_2, e_3, e_2, e_3, e_0, \ldots)$,	$v^4(x) = 0,\ \ v^4(y) = 1$,
$v_L^5 = (e_2, e_2, e_3, e_2, e_3, e_0, \ldots)$,	$v^5(x) = 1,\ \ v^5(y) = 1$,
$v_L^6 = (e_2, e_3, e_2, e_3, e_0, \ldots)$,	$v^6(x) = 2,\ \ v^6(y) = 2$,
$v_L^7 = (e_3, e_2, e_3, e_0, \ldots)$,	$v^7(x) = 3,\ \ v^7(y) = 6$,
$v_L^8 = (e_2, e_3, e_0, \ldots)$,	$v^8(x) = 3,\ \ v^8(y) = 6$.

It can be proved that $H_R(v_L, v)$ is defined for each state (v_L, v) in $W_L \times W_{\mathcal{N}}$ such that $v_L(a_1) = e_1$. Moreover, if $H_R(v_L, v) = (\overline{v}_L, \overline{v})$, then $\overline{v}(y) = (v(x))!$.

Suppose that for each n, $1 \leqslant n \leqslant m$, the sets I_n of instructions of order n and $\boldsymbol{P}_n$ of procedures of order n have been defined and that realization R has been extended to I_n and $\boldsymbol{P}_n$. Then we define I_{m+1} and $\boldsymbol{P}_{m+1}$ as follows:

The set I_{m+1} consists of all expressions

(i1) $\circ\big[\circ[[J_k a_1] s^*] H_q\big]$, where $k \in \mathcal{N}_0$, $H_q \in \boldsymbol{P}_m$, e_q is the label of H_q, $q \in \mathcal{N}_0$, $s^* \in \boldsymbol{S}_L$ and s^* is in one of the following forms:

$(s_L q1)$ $\quad [a_1/E_q]$,

$(s_L q2)$ $\quad [a_1/E_q\ a_2/a_1]$,

$(s_L q3)$ $\quad [a_1/E_q\ a_2/E_{k_2} \ \ldots\ a_n/E_{k_n}\ a_{n+1}/a_2]$, $n > 1$, $k_2, \ldots, k_n \in \mathcal{N}_0$,

(i2) $\circ\big[[J_k a_1] \preceq [\alpha \circ [s_1^* H_{q_1}] \circ [s_2^* H_{q_2}]]\big]$, where $k \in \mathcal{N}_0$, $\alpha \in \boldsymbol{F}^o$, H_{q_1}, H_{q_2} are procedures of orders $\leqslant m$ and at least one of them is of order m, e_{q_1} and e_{q_2} are the labels of H_{q_1} and H_{q_2}, respectively, s_1^* and $s_2^* \in \boldsymbol{S}_L$ and each of them

is in a form corresponding to $(s_L q1)-(s_L q3)$,

(i3) $\circ\big[[J_k a_1]\swarrow[\alpha\circ[s^* H_q]\circ[s_1^* s_1]]\big]$, where $k\in\mathcal{N}_0$, $\alpha\in F^o$, s^* and H_q satisfy the conditions in (i1), $s_1^*\in S_L$, $s_1\in S$,

(i4) $\circ\big[[J_k a_1]\swarrow[\alpha\circ[s_1^* s_1]\circ[s^* H_q]]\big]$, where all conditions in (i3) are satisfied.

The set P_{m+1} of procedures of order $m+1$ consists of the expressions

(p) $\circ^*[H_{k_1} H_{k_1} \dots H_{k_n} H_t]$, $n\in\mathcal{N}_0$,

where

(pi) $H_{k_1}, \dots, H_{k_n}$ are instructions of orders $\leqslant m+1$ and at least one is of order $m+1$, the labels $e_{k_1}, \dots, e_{k_n}$ of $H_{k_1}, \dots, H_{k_n}$, respectively, are different, and e_{k_1} is said to be the *label* of (p),

(pii) conditions (ii), (iii), (iv) of the definition of procedures in P_1 are satisfied.

In order to extend the realization R to I_{m+1} we adopt for any instruction H of order $m+1$, which is in the form (i1), (i2), (i3), (i4) the following definitions, respectively:

(ir1) $$H_R(v_L, v) = \begin{cases} H_{qR}\big(s_R^*([J_k a_1]_R(v_L, v))\big) & \text{if this is defined,} \\ \text{undefined} & \text{otherwise;} \end{cases}$$

(ir2) $$H_R(v_L, v) = \begin{cases} H_{q_1 R}\big(s_{1R}^*([J_k a_1]_R(v_L, v))\big) & \text{if this is defined and } \alpha_R(v) = \swarrow, \\ H_{q_2 R}\big(s_{2R}^*([J_k a_1]_R(v_L, v))\big) & \text{if this is defined and } \alpha_R(v) = \nearrow, \\ \text{undefined} & \text{otherwise;} \end{cases}$$

(ir3) $$H_R(v_L, v) = \begin{cases} H_{qR}\big(s_R^*([J_k a_1]_R(v_L, v))\big) & \text{if this is defined and } \alpha_R(v) = \swarrow, \\ s_{1R}^*\big(s_{1R}([J_k a_1]_R(v_L, v))\big) & \text{if this is defined and } \alpha_R(v) = \nearrow, \\ \text{undefined} & \text{otherwise;} \end{cases}$$

(ir4) $$H_R(v_L, v) = \begin{cases} s_{1R}^*\big(s_{1R}([J_k a_1]_R(v_L, v))\big) & \text{if this is defined and } \alpha_R(v) = \swarrow, \\ H_{qR}\big(s_R^*([J_k a_1]_R(v_L, v))\big) & \text{if this is defined and } \alpha_R(v) = \nearrow, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In order to extend the realization R to P_{m+1} we adopt for $H\in P_{m+1}$ the definition of a computation of H by R for a state (v_L, v) the same as for procedures of order 1. The following statement holds (see [26]).

2.2. *If theorem* 2.1 *holds for all procedures of orders* i, $1\leqslant i\leqslant m$, *then it also holds for procedures of order* $m+1$.

Theorems 2.1 and 2.2 enable us to extend R to P_{m+1}.

For any $H\in P_{m+1}$ we set

(pr) $H_R(v_L, v) = H_R^{\overline{m}+1}(v_L, v)$, if (c) is a computation of H by R for (v_L, v), and $H_R(v_L, v)$ is undefined if a computation of H by R for (v_L, v) does not exist.

Let us set $P = \bigcup_{m\in\mathcal{N}_0} P_m$.

The set $F_L S$ of programs ($F_L S$-expressions) is the least set of expressions under A satisfying the following conditions:

(f_Ls1) $S\cup S_L^0\cup Sp\cup P\subset F_L S$,

(f_Ls2) if $H_1, H_2\in F_L S$, then $\circ[H_1 H_2]\in F_L S$,

(f_Ls3) if $H_1, H_2\in F_L S$, then for each $\alpha\in F^o$, $\swarrow[\alpha H_1 H_2]\in F_L S$.

In order to extend realization R to $F_L S$ we adopt the following additional definitions:

($f_L$sr2) $$\circ[H_1 H_2]_R(v_L, v) = \begin{cases} H_{2R}\big(H_{1R}(v_L, v)\big) & \text{if this is defined,} \\ \text{undefined} & \text{otherwise;} \end{cases}$$

($f_L$sr3) $$\swarrow[\alpha H_1 H_2]_R(v_L, v) = \begin{cases} H_{1R}(v_L, v) & \text{if this is defined and } \alpha_R(v) = \swarrow, \\ H_{2R}(v_L, v) & \text{if this is defined and } \alpha_R(v) = \nearrow, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Notice that (ir11), (ir12), (ir1), (ir2), (ir3), (ir4) are conformable to these definitions.

The set $F_L ST$ of generalized terms is the least set of finite sequences of elements in A satisfying the following conditions:

(gt1) $T\subset F_L ST$,

(gt2) if $H\in F_L S$ and $\tau\in F_L ST$, then $H\tau\in F_L ST$,

(gt3) if $\varphi\in\Phi_k$, $k>0$, and $\tau_1, \dots, \tau_k\in F_L ST$, then $\varphi(\tau_1 \dots \tau_k)\in F_L ST$.

Realization R is now extended to $F_L ST$ thus: For each $\tau\in F_L ST$, τ_R is a partial mapping from $W_L\times W_U$ into U which is defined as follows:

(gtr1) $\tau_R(v_L, v) = \tau_R(v)$ for $\tau\in T$,

(gtr2) $H\tau_R(v_L, v) = \tau_R\big(H_R(v_L, v)\big)$ if $(\overline{v}_L, \overline{v}) = H_R(v_L, v)$ and $\tau_R(\overline{v}_L, \overline{v})$ are defined, and is undefined in the opposite case,

(gtr3) $\varphi(\tau_1 \dots \tau_k)_R(v_L, v) = \varphi_R\big(\tau_{1R}(v_L, v), \dots, \tau_{kR}(v_L, v)\big)$ if all $\tau_{iR}(v_L, v)$ for $i = 1, \dots, k$ are defined, and is undefined in the opposite case.

The set $\boldsymbol{F_LSF}$ of generalized formulas under A is the least set satisfying the following conditions:

(f1) if $\varrho \in P_k$ and $\tau_1, \ldots, \tau_k \in \boldsymbol{F_LST}$, then $\varrho(\tau_1 \ldots \tau_k) \in \boldsymbol{F_LSF}$,

(f2) if $p \in V_0$, then $p \in \boldsymbol{F_LSF}$,

(f3) $E_i \in \boldsymbol{F_LSF}$ for $0 \leqslant i \leqslant \omega$,

(f4) $a_i \in \boldsymbol{F_LSF}$ for $i \in \mathcal{N}_0$,

(f5) if $\alpha, \beta \in \boldsymbol{F_LSF}$, then $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \Rightarrow \beta)$, $\neg\alpha$, $D_i\alpha$ for $i \in \mathcal{N}_0$ are in $\boldsymbol{F_LSF}$,

(f6) if $\alpha \in \boldsymbol{F_LSF}$, $H \in \boldsymbol{F_LS}$, then $H\alpha \in \boldsymbol{F_LSF}$,

(f7) if $\alpha_1, \alpha_2, \ldots$ is a sequence of generalized formulas in $\boldsymbol{F_LSF}$ and the set of all individual variables and propositional variables (i.e. variables in V_i and in V_0) which occur in these generalized formulas is finite, then $\bigvee(\alpha_1 \alpha_2 \ldots)$ and $\bigwedge(\alpha_1 \alpha_2 \ldots)$ are in $\boldsymbol{F_LSF}$.

Generalized formulas satisfying one of the conditions (f1) where $\tau_1, \ldots, \tau_k \in \boldsymbol{T}$, (f2), (f3), (f4) are said to be *atomic*. The set of all atomic generalized formulas will be denoted by $\boldsymbol{F}_{at}$. The set of generalized formulas satisfying one of the conditions (f1), (f2), (f3), (f4) will be denoted by $\overline{\boldsymbol{F}}_{at}$.

Given realization R in $U \neq 0$, generalized formulas will be realized as mappings from $W_L \times W_U$ in $\mathfrak{P}_\omega$. Notice that $\mathfrak{P}_\omega$ is a complete lattice, i.e., that for each infinite set of its elements there exist a least upper bound (l.u.b.) and a greatest lower bound (g.l.b.). The inductive definition of realizations of generalized formulas is as follows:

(fr1) $$\varrho(\tau_1 \ldots \tau_k)_R(v_L, v) = \begin{cases} \varrho_R(\tau_{1R}(v_L, v), \ldots, \tau_{kR}(v_L, v)) \\ \quad \text{if } \tau_{iR}(v_L, v) \text{ for } i = 1, \ldots, k \text{ are defined,} \\ \nearrow \quad \text{in the opposite case;} \end{cases}$$

(fr2) $p_R(v_L, v) = v(p)$, for $p \in V_0$,

(fr3) $E_{iR}(v_L, v) = e_i$, for $0 \leqslant i \leqslant \omega$,

(fr4) $a_{iR}(v_L, v) = v_L(a_i)$, for $i \in \mathcal{N}_0$,

(fr5) $(\alpha \vee \beta)_R(v_L, v) = \alpha_R(v_L, v) \cup \beta_R(v_L, v)$;
$(\alpha \wedge \beta)_R(v_L, v) = \alpha_R(v_L, v) \cap \beta_R(v_L, v)$;
$(\alpha \Rightarrow \beta)_R(v_L, v) = \alpha_R(v_L, v) \to \beta_R(v_L, v)$;
$\neg\alpha_R(v_L, v) = -\alpha_R(v_L, v)$;
$D_i\alpha_R(v_L, v) = \boldsymbol{d}_i\alpha_R(v_L, v)$, $i \in \mathcal{N}_0$,

(fr6) $$H\alpha_R(v_L, v) = \begin{cases} \alpha_R(H_R(v_L, v)) & \text{if } H_R(v_L, v) \text{ is defined,} \\ \nearrow & \text{in the opposite case;} \end{cases}$$

(fr7) $\bigvee(\alpha_1 \alpha_2 \ldots)_R(v_L, v) = \text{l.u.b. } \alpha_{iR}(v_L, v)$, $i \in \mathcal{N}_0$;
$\bigwedge(\alpha_1 \alpha_2 \ldots)_R(v_L, v) = \text{g.l.b. } \alpha_{iR}(v_L, v)$, $i \in \mathcal{N}_0$.

All operations on the right-hand sides of these equations are in $\mathfrak{P}_\omega$.

Let $\boldsymbol{BF_LSF}$ be the least subset of $\boldsymbol{F_LSF}$ satisfying the conditions (f1), (f2), (f5), (f6), (f7), in which $\boldsymbol{F_LSF}$ is replaced by $\boldsymbol{BF_LSF}$, and moreover, $E_0, E_\omega \in \boldsymbol{BF_LSF}$. Then the following holds.

2.3. *For each* $\alpha \in \boldsymbol{BF_LSF}$, $\alpha_R(v_L, v) \in \{\swarrow, \nearrow\}$, $D_i\alpha_R(v_L, v) = \alpha_R(v_L, v)$, $i \in \mathcal{N}_0$, *and* $\cup, \cap, \to, -$, l.u.b., g.l.b., *which realize* $\vee, \wedge, \Rightarrow, \neg, \bigvee, \bigwedge$, *respectively, occurring in* α, *are Boolean operations in the two-element Boolean algebra* $\mathfrak{B}_0$.

§ 3. *FS*-expressions

A procedure $H \in \boldsymbol{P}$, $H = \circ^*[H_{k_i}, H_{k_1} \ldots H_{k_n} H_t]$, is said to be *equivalent* to $K \in \boldsymbol{FS}$ if, for every realization R in any set $U \neq \emptyset$ and every state (v_L, v) such that $v_L(a_1) = e_k$,

(1) $H_R(v_L, v)$ is defined if and only if $K_R(v)$ is defined,

(2) if $H_R(v_L, v) = (\overline{v}_L, \overline{v})$ is defined, then $K_R(v) = \overline{v}$.

A procedure $H \in \boldsymbol{P}_1$ is said to be a *finite control procedure* if all label substitutions occurring in H_{k_i}, $i = 1, \ldots, n$, are in forms $[a_1/E_k]$, $k \in \mathcal{N}_0$, $[a_1/a_2]$, $[\,]$. The set of all finite control procedures in $\boldsymbol{P}_1$ will be denoted by $\boldsymbol{FCP}_1$.

3.1. *For each FS-expression* K *there is an* $H \in \boldsymbol{FCP}_1$ *equivalent to* K.

The easy proof is left to the reader.

It follows from 3.1 that $\boldsymbol{F_LS}$ may be treated as an extension of $\boldsymbol{FS}$.

§ 4. Fundamental theorems on programs in F_LS and generalized terms

4.1. *Let* $H = \circ^*[H_{k_i}, H_{k_1} \ldots H_{k_n} H_t]$ *be a procedure in* $\boldsymbol{P}$. *For each realization* R *in any set* $U \neq \emptyset$ *and any states* (v_L, v), (w_L, w) *in* $W_L \times W_U$, *such that* $v_L(a_1) = w_L(a_1)$ *and* $v = w$,

(i) $H_R(v_L, v)$ *is defined if and only if* $H_R(w_L, w)$ *is defined,*

(ii) *if* $H_R(v_L, v) = (\overline{v}_L, \overline{v})$ *and* $H_R(w_L, w) = (\overline{w}_L, \overline{w})$, *then* $\overline{v} = \overline{w}$.

For a proof of 4.1 see [26].

4.2. (*Theorem on the normal form of programs in* $\boldsymbol{F_LS}$). *For every* $H \in \boldsymbol{F_LS}$ *there is a program* $\text{nor}(H) \in \boldsymbol{F_LS}$, *effectively defined, such that for every realization* R *in any set* $U \neq \emptyset$ *and any* $(v_L, v) \in W_L \times W_U$,

(i) $H_R(v_L, v)$ *is defined if and only if* $\text{nor}(H)_R(v_L, v)$ *is defined,*

(ii) *if* $H_R(v_L, v) = (\overline{v}_L, \overline{v})$, *then* $\text{nor}(H)_R(v_L, v) = (v_L, \overline{v})$.

For the proof of 4.2 see [26].

4.3. *If* $H \in \boldsymbol{P}$ *and* e_k *is the label of* H, *then for each term* $\tau \in \boldsymbol{T}$, *any realization* R *in* $U \neq \emptyset$ *and any states* (v_L, v), (w_L, w) *such that* $v_L(a_1) = w_L(a_1) = e_k$,

$$H\tau_R(v_L, v) = H\tau_R(w_L, v).$$

This equation means that either both sides are defined or both are undefined, and if both are defined, then they are equal. Theorem 4.3 follows from 4.1 and the definition of $H\tau_R(v_L, v)$.

On applying generalized terms we may define the notion of the programmability of partial functions by means of procedures. Let R be a realization of $\mathscr{L}$ in a set $U \neq \emptyset$ and let f be an n-argument partial function from U into U. We say that f is *programmable in R by means of a procedure* if there is an $H \in \boldsymbol{P}$ with a label e_k such that $x_1, \ldots, x_n$ are all free individual variables in H (i.e., H_R depends on $v(x_1), \ldots, v(x_n)$) and x_{n+1} occurring in H is not free, moreover, the following conditions are satisfied: for each $(v_L, v) \in W_L \times W_U$ such that $v_L(a_1) = e_k$,

(1) $f(v(x_1), \ldots, v(x_n))$ is defined if and only if $Hx_{n+1R}(v_L, v)$ is defined;

(2) if $Hx_{n+1R}(v_L, v)$ is defined, then $f(v(x_1), \ldots, v(x_n)) = Hx_{n+1R}(v_L, v)$.

It follows from 3.1 that programmability by means of procedures is a generalization of programmability.

On applying 4.2 and a method similar to that used in the proof of theorem 1.1 in [22], we can prove that following theorem on generalized terms.

4.4. *For every generalized term $\tau \in \boldsymbol{F}_L\boldsymbol{ST}$ there exist a program H in $\boldsymbol{F}_L\boldsymbol{S}$ and a term $\tau' \in \boldsymbol{T}$, both effectively defined, such that for every realization R in any $U \neq \emptyset$ and for each state (v_L, v) in $W_L \times W$*

$$\tau_R(v_L, v) = H\tau'_R(v_L, v),$$

i.e., either both sides of this equations are defined or both are undefined and if both are defined, then they are equal.

Theorem 4.4 is applied to prove the following theorem.

4.5. *There is a mapping χ effectively defined which assigns to each $\varrho(\tau_1 \ldots \tau_k) \in \overline{\boldsymbol{F}}_{at}$ a formula $H\varrho(\tau'_1, \ldots, \tau'_k)$, where $H \in \boldsymbol{F}_L\boldsymbol{S}$ and $\tau'_1, \ldots, \tau'_k \in \boldsymbol{T}$, such that for every realization R in any $U \neq \emptyset$ and for every state $(v_L, v) \in W_L \times W_U$,*

$$\varrho(\tau_1 \ldots \tau_k)_R(v_L, v) = \chi(\varrho(\tau_1 \ldots \tau_k))_R(v_L, v).$$

§ 5. Formalization of *EAL*

Let $\mathscr{L}$ be a formalized language of ***EAL***. A realization R of $\mathscr{L}$ in U is said to be a *model* of a generalized formula $\alpha \in \boldsymbol{F}_L\boldsymbol{SF}$ if for every *state* $(v_L, v) \in W_L \times W_U$, $\alpha_R(v_L, v) = |\!/$. A realization R is said to be a model of $\mathscr{A} \subset \boldsymbol{F}_L\boldsymbol{SF}$ if it is a model of each $\alpha \in \mathscr{A}$. A generalized formula α is said to be a *tautology* of ***EAL*** if every realization of $\mathscr{L}$ is a model of α. A generalized formula is said to be a *semantic consequence* of a set $\mathscr{A} \subset \boldsymbol{F}_L\boldsymbol{SF}$ if each model of $\mathscr{A}$ is a model of α. This will be written as $\mathscr{A} \models \alpha$. The set of all semantic consequences of $\mathscr{A}$ will be denoted by $\mathrm{Cn}(\mathscr{A})$. In particular, instead of $\emptyset \models \alpha$ we shall write $\models \alpha$, and $\mathrm{Cn}(\emptyset) = \{\alpha \in \boldsymbol{F}_L\boldsymbol{SF}: \models \alpha\}$ is the set of all tautologies of ***EAL*** in $\mathscr{L}$.

The semantic consequence operation will be replaced by a formal consequence operation C. For this purpose we shall introduce a set $\mathscr{A}_l$ of logical axioms and a set of rules of inference.

Let us adopt the following definitions and notation.

Every label substitution s^* in S^0_L determines a mapping $\bar{s}^*$ which is defined as follows: If $s^* = [a_1/E_k\, a_2/a_1]$, then $\bar{s}^*(a_1) = E_k$ and $\bar{s}^*(a_{n+1}) = a_n$ for $n \in \mathscr{N}_0$; if $s^* = [a_1/E_{k_1} \ldots a_n/E_{k_n}\, a_{n+1}/a_2]$, then $\bar{s}^*(a_i) = E_{k_i}$ for $i = 1, \ldots, n$ and $\bar{s}^*(a_{n+i}) = a_{i+1}$ for $i \in \mathscr{N}_0$; if $s^* = [a_1/a_2]$, then $\bar{s}^*(a_i) = a_{i+1}$, for $i \in \mathscr{N}_0$; if $s^* = [a_1/E_0]$, then $\bar{s}^*(a_n) = E_0$ for each $n \in \mathscr{N}_0$; if $s^* = [\,]$, then $\bar{s}^*(a_n) = a_n$ for $n \in \mathscr{N}_0$.

Observe that the following equation holds:

$$s^*a_{nR}(v_L, v) = \bar{s}^*a_{nR}(v_L, v), \quad n \in \mathscr{N}_0.$$

For any well formed expression θ in $\mathscr{L}$, and $s^* \in S^0_L$, let $\bar{s}^*\theta$ denote the expression obtained from θ by the simultaneous replacement of each a_n, $n \in \mathscr{N}_0$, occurring in θ by $\bar{s}^*\theta$. Similarly, for any substitution $s \in \boldsymbol{S}$, $s = [x_1/\tau_1 \ldots x_n/\tau_n\, p_1/\alpha_1 \ldots p_m/\alpha_m]$, and any θ, let $\bar{s}\theta$ denote the expression obtained from θ by the simultaneous replacement of x_i by τ_i, $i = 1, \ldots, n$, and of p_i by α_i, for $i = 1, \ldots, m$.

Instead of $((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$ we shall write, for brevity, $(\alpha \Leftrightarrow \beta)$ for any generalized formulas α, β.

As the logical axiom schemes for ***EAL*** we adopt the following ones.

Group A (*Axiom schemes for intuitionistic logic*).

(A_1) $(\alpha \Rightarrow (\beta \Rightarrow \alpha))$,

(A_2) $((\alpha \Rightarrow (\beta \Rightarrow \gamma)) \Rightarrow ((\alpha \Rightarrow \beta) \Rightarrow (\alpha \Rightarrow \gamma)))$,

(A_3) $(\alpha \Rightarrow (\alpha \vee \beta))$,

(A_4) $(\beta \Rightarrow (\alpha \vee \beta))$,

(A_5) $((\alpha \Rightarrow \gamma) \Rightarrow ((\beta \Rightarrow \gamma) \Rightarrow ((\alpha \vee \beta) \Rightarrow \gamma)))$,

(A_6) $((\alpha \wedge \beta) \Rightarrow \alpha)$,

(A_7) $((\alpha \wedge \beta) \Rightarrow \beta)$,

(A_8) $((\alpha \Rightarrow \beta) \Rightarrow ((\alpha \Rightarrow \gamma) \Rightarrow (\alpha \Rightarrow (\beta \wedge \gamma))))$,

(A_9) $((\alpha \Rightarrow \neg\beta) \Rightarrow (\beta \Rightarrow \neg\alpha))$,

(A_{10}) $(\neg(\alpha \Rightarrow \alpha) \Rightarrow \beta)$.

Group B (*Additional axiom schemes for the ω^+-valued logic*).

(B_1) $(D_i(\alpha \vee \beta) \Leftrightarrow (D_i\alpha \vee D_i\beta))$, $\quad i \in \mathscr{N}_0$,

(B_2) $(D_i(\alpha \wedge \beta) \Leftrightarrow (D_i\alpha \wedge D_i\beta))$, $\quad i \in \mathscr{N}_0$,

(B_3) $(D_i(\alpha \Rightarrow \beta) \Leftrightarrow ((D_1\alpha \Rightarrow D_1\beta) \wedge (\ldots \wedge (D_i\alpha \Rightarrow D_i\beta) \ldots)))$, $\quad i \in \mathscr{N}_0$,

(B_4) $(D_i\neg\alpha \Leftrightarrow \neg D_1\alpha)$, $\quad i \in \mathscr{N}_0$,

(B_5) $(D_iD_j\alpha \Leftrightarrow D_j\alpha)$, $\quad i, j \in \mathscr{N}_0$,

(B_6) D_iE_k for $i \leqslant k$, $i \in \mathscr{N}_0$, $0 \leqslant k \leqslant \omega$; $\neg D_iE_k$ for $i > k$, $i \in \mathscr{N}_0$, $k \in \mathscr{N}$,

(B_7) $(D_{i+1}\alpha \Rightarrow D_i\alpha)$, $i \in \mathscr{N}_0$,

(B_8) E_ω,

(B_9) $(D_1\alpha \vee \neg D_1\alpha)$,

(B_{10}) $((D_i\alpha \wedge E_i) \Rightarrow \alpha)$, $i \in \mathcal{N}_0$.

*Group C (Additional axiom schemes for **EAL**).*

(C_1) $(D_1 p \Leftrightarrow p)$, $p \in V_0$,

(C_2) $(D_1\varrho(\tau_1, \ldots, \tau_k) \Leftrightarrow \varrho(\tau_1, \ldots, \tau_k))$, $\varrho \in P_k$, $k \in \mathcal{N}_0$, $\tau_1, \ldots, \tau_k \in \boldsymbol{F}_L\boldsymbol{ST}$,

(C_3) $(\alpha_i \Rightarrow \mathbf{\vee}(\alpha_1 \ldots \alpha_i \ldots))$, $i \in \mathcal{N}_0$,

(C_4) $(\mathbf{\wedge}(\alpha_1 \ldots \alpha_i \ldots) \Rightarrow \alpha_i)$, $i \in \mathcal{N}_0$,

(C_5) $(\mathbf{\wedge}((\alpha_1 \Rightarrow \beta)(\alpha_2 \Rightarrow \beta) \ldots) \Rightarrow (\mathbf{\vee}(\alpha_1\alpha_2 \ldots) \Rightarrow \beta))$,

(C_6) $(\mathbf{\wedge}((\alpha \Rightarrow \beta_1)(\alpha \Rightarrow \beta_2) \ldots) \Rightarrow (\alpha \Rightarrow \mathbf{\wedge}(\beta_1\beta_2 \ldots)))$,

(C_7) $\neg\mathbf{\wedge}(a_1a_2 \ldots)$,

(C_8) $\neg\mathbf{\wedge}(D_1a_n\ D_2a_n \ldots)$, $n \in \mathcal{N}_0$,

(C_9) $(\neg a_n \Rightarrow \neg a_{n+1})$, $n \in \mathcal{N}_0$,

(C_{10}) $(s\alpha \Leftrightarrow \bar{s}\alpha)$, $\alpha \in \boldsymbol{F}_{at}$, $s \in \boldsymbol{S}$,

(C_{11}) $(s^*\alpha \Leftrightarrow \bar{s}^*\alpha)$, $\alpha \in \boldsymbol{F}_{at}$, $s^* \in \boldsymbol{S}_L^0$,

(C_{12}) $(s^*s\alpha \Leftrightarrow ss^*\alpha)$, $s \in \boldsymbol{S}$, $s^* \in \boldsymbol{S}_L^0$,

(C_{13}) $(H\varrho(\tau_1, \ldots, \tau_k) \Leftrightarrow \varrho(H\tau_1, \ldots, H\tau_k))$,

(C_{14}) $(\chi(\varrho(\tau_1, \ldots, \tau_k)) \Leftrightarrow \varrho(\tau_1, \ldots, \tau_k))$, $\varrho \in P_k$, $k \in \mathcal{N}_0$, $\tau_1, \ldots, \tau_k \in \boldsymbol{F}_L\boldsymbol{ST}$ (see 4.5).

(C_{15}) $(H(\alpha \vee \beta) \Leftrightarrow (H\alpha \vee H\beta))$,

(C_{16}) $(H(\alpha \wedge \beta) \Leftrightarrow (H\alpha \wedge H\beta))$,

(C_{17}) $(H\neg\alpha \Rightarrow \neg H\alpha)$,

(C_{18}) $(HE_\omega \Rightarrow (\neg H\alpha \Rightarrow H\neg\alpha))$,

(C_{19}) $(H(\alpha \Rightarrow \beta) \Rightarrow (H\alpha \Rightarrow H\beta))$,

(C_{20}) $(HE_\omega \Rightarrow ((H\alpha \Rightarrow H\beta) \Rightarrow H(\alpha \Rightarrow \beta)))$,

(C_{21}) $(HD_i\alpha \Leftrightarrow D_iH\alpha)$, $i \in \mathcal{N}_0$,

(C_{22}) $(H\mathbf{\vee}(\alpha_1\alpha_2 \ldots) \Leftrightarrow \mathbf{\vee}(H\alpha_1\ H\alpha_2 \ldots))$,

(C_{23}) $(H\mathbf{\wedge}(\alpha_1\alpha_2 \ldots) \Leftrightarrow \mathbf{\wedge}(H\alpha_1\ H\alpha_2 \ldots))$,

(C_{24}) $([J_ka_n]\beta \Leftrightarrow (J_ka_n \wedge \beta))$,

(C_{25}) $(\circ[H_1H_2]\beta \Leftrightarrow H_1H_2\beta)$,

(C_{26}) $(\measuredangle[\alpha H_1H_2]\beta \Leftrightarrow ((\alpha \wedge H_1\beta) \vee (\neg\alpha \wedge H_2\beta)))$,

(C_{27}) $(\circ^*[H_{k_1r}H_{k_1} \ldots H_{k_n}H_t]\beta \Leftrightarrow \mathbf{\vee}(\circ[H_{k_1r}H_{k_1}H_t]\beta \ldots \circ[H_{k_1r}H_{k_1}H_{i_1} \ldots H_{i_m}H_t]\beta \ldots))$, where $i_1, \ldots, i_m$ are in $\{k_1, \ldots, k_n\}$, $m \in \mathcal{N}_0$ and $\circ[H_{j_1} \ldots H_{j_i}]$ is written instead of $\circ[H_{j_1} \circ [\ldots \circ [H_{j_{i-1}}H_{j_i}] \ldots]]$ for any $H_{j_1}, \ldots, H_{j_i}$ in $\boldsymbol{F}_L\boldsymbol{S}$.

The following rules of inference are adopted

(r1) $\dfrac{\alpha, (\alpha \Rightarrow \beta)}{\beta}$,

(r2) $\dfrac{\alpha, HE_\omega}{H\alpha}$,

(r3) $\dfrac{\alpha_i, i \in \mathcal{N}_0}{\mathbf{\wedge}(\alpha_1\alpha_2 \ldots)}$,

(r4) $\dfrac{\alpha}{D_i\alpha, i \in \mathcal{N}_0}$,

(r5) $\dfrac{D_i\alpha, i \in \mathcal{N}_0}{\alpha}$.

For any set $\mathscr{A} \subset \boldsymbol{F}_L\boldsymbol{SF}$ we define $C(\mathscr{A})$ as the least set of generalized formulas containing the union of $\mathscr{A} \cup \mathscr{A}_l$ and closed with respect to the inference rules (r1)–(r5). In particular, $C(0)$ is the least set of generalized formulas containing logical axioms and closed with respect to the inference rules (r1)–(r5).

5.1. (*Completeness theorem for **EAL***).

$$Cn(0) = C(0).$$

*Thus the set of all tautologies of **EAL** coincides with the set $C(0)$.*

As in algorithmic logic, various properties of programs with recursive procedures may be expressed by means of generalized formulas in $\boldsymbol{F}_L\boldsymbol{SF}$. For instance, for any $H \in \boldsymbol{F}_L\boldsymbol{S}$, HE_ω describes the stop property of H, i.e., for any realization R and state (v_L, v)

$$HE_{\omega R}(v_L, v) = E_{\omega R}(H_R(v_L, v)) = \bigvee \text{ if and only if } H_R(v_L, v) \text{ is defined.}$$

In particular, if $H \in \boldsymbol{P}$, then $HE_{\omega R}(v_L, v) = \bigvee$ if and only if there exists a computation of H by R for (v_L, v).

Let $H \in \boldsymbol{P}$ and let e_k be the label of H. Consider the formula $(J_ka_1 \wedge (\alpha \Rightarrow H\beta))$ for some α, β in $\boldsymbol{F}^0$. This generalized formula describes the correctness of H (with respect to an initial condition α for data and a terminal condition β for the results of computations of H).

Let H and G be any procedures in $\boldsymbol{P}$ having a common label e_k. Then the generalized formula

$$\Big(J_ka_1 \wedge \Big((HE_\omega \Rightarrow (GE_\omega \wedge (H\tau_1 = G\tau_2))) \wedge \Big((GE_\omega \Rightarrow (HE_\omega \wedge (H\tau_1 = G\tau_2)))\Big)\Big)\Big),$$

where $\tau_1, \tau_2 \in \boldsymbol{T}$, describes an equivalence of H and G.

On applying generalized formulas in $\boldsymbol{F}_L\boldsymbol{SF}$ we can define the notion of *relations programmable by means of procedures*. Let R be a realization of $\mathscr{L}$ in $U \neq \emptyset$ and let $r \subset U^n$. The relation r is *programmable in R by means of $H \in \boldsymbol{P}$* if there is

a formula $\alpha \in F^o$ such that all its free individual variables are $x_1, \ldots, x_n$ and such that the following holds:

$$(v(x_1), \ldots, v(x_n)) \in r \text{ if and only if } H\alpha_R(v_L, v) = \bigvee$$

for each state (v_L, v) in $W_L \times W_U$ such that $v_L(a_1) = e_k$, where e_k is the label of H.

We can also introduce the notion of *strong programmability of relations by means of procedures*, as in Chapter I, § 6.

§ 6. Multiple-valued relations and mixed-valued relational systems

By a *k-argument m-valued relation* $(2 \leqslant m < \omega)$ *on a set* $U \neq \emptyset$ we shall understand any mapping $r\colon U^k \to \mathscr{P}_m$, where $\mathscr{P}_m = \{e_0, \ldots, e_{m-2}, e_\omega\}$ is the set of elements of the m-element Post algebra $\mathscr{P}_m$ of order m, $e_0 = \bigwedge$, $e_\omega = \bigvee$ (see § 1). In particular, any k-argument 2-valued relation on U is meant to be a mapping $r\colon U^k \to \{\bigvee, \bigwedge\}$, i.e., the characteristic function of a k-argument relation on U in the usual sense.

EXAMPLE. Setting $r(0) = e_0$, $r(2n-1) = e_1$ and $r(2n) = e_\omega$ for $n \in \mathscr{N}_0$, we define a one-argument three-valued relation on $\mathscr{N}$.

Every $r\colon U^k \to \mathscr{P}_m$ determines uniquely $m-1$ two-valued relations $d_1 r, \ldots, d_{m-1} r$ defined as follows: for each $(u_1, \ldots, u_k) \in U^k$ and $1 \leqslant i < m-1$,

(1)
$$d_i r(u_1, \ldots, u_k) = \begin{cases} \bigvee & \text{if } r(u_1, \ldots, u_k) \geqslant e_i, \\ \bigwedge & \text{in the opposite case;} \end{cases}$$
$$d_{m-1} r(u_1, \ldots, u_k) = \begin{cases} \bigvee & \text{if } r(u_1, \ldots, u_k) = e_\omega, \\ \bigwedge & \text{in the opposite case.} \end{cases}$$

Moreover, the following holds:

(2) $$d_{m-1} r(u_1, \ldots, u_k) \leqslant d_{m-2} r(u_1, \ldots, u_k) \leqslant \ldots \leqslant d_1(u_1, \ldots, u_k).$$

In fact, if $d_i r(u_1, \ldots, u_k) = \bigvee$, $i > 1$ then $r(u_1, \ldots, u_k) \geqslant e_i \geqslant e_{i-1}$. Thus

$$d_{i-1} r(u_1, \ldots, u_k) = \bigvee.$$

It is also easy to verify that

(3) $$r(u_1, \ldots, u_k) = d_1 r(u_1, \ldots, u_k) \cap e_1 \cup \ldots \cup d_{m-2} r(u_1, \ldots, u_k) \cap e_{m-2} \cup d_{m-1} r(u_1, \ldots, u_k)$$

where $\cap$ and $\cup$ are operations in $\mathfrak{P}_m$ (see § 1).

Conversely, given $m-1$ two-valued k-argument relations on U

$$r_1\colon U^k \to \{\bigvee, \bigwedge\}, \quad \ldots, \quad r_{m-1}\colon U^k \to \{\bigvee, \bigwedge\}$$

such that $r_{m-1}(u_1, \ldots, u_k) \leqslant \ldots \leqslant r_1(u_1, \ldots, u_k)$ for each $(u_1, \ldots, u_k) \in U^k$, the equation

$$r(u_1, \ldots, u_k) = r_1(u_1, \ldots, u_k) \cap e_1 \cup \ldots \cup r_{m-2}(u_1, \ldots, u_k) \cap e_{m-2} \cup r_{m-1}(u_1, \ldots, u_k)$$

defines a k-argument m-valued relation r on U such that $d_i r = r_i$, $i = 1, \ldots, m-1$.

On the other hand, each $r\colon U^k \to \mathscr{P}_m$ determines also $m-1$ k-argument two-valued relations $j_0 r, \ldots, j_{m-2} r$ defined thus:

$$j_i r(u_1, \ldots, u_k) = \begin{cases} \bigvee & \text{if } r(u_1, \ldots, u_k) = e_i, \\ \bigwedge & \text{in the opposite case,} \end{cases} \qquad i = 0, \ldots, m-2.$$

The following equations hold (see Sec. 1):

$$j_0 r(u_1, \ldots, u_k) = -d_1 r(u_1, \ldots, u_k),$$
$$j_i r(u_1, \ldots, u_k) = -d_{i+1} r(u_1, \ldots, u_k) \cap d_i r(u_1, \ldots, u_k), \qquad 1 \leqslant i \leqslant m-2.$$

EXAMPLES. Let $\mathscr{Z}$ be the set of integers and let $r\colon \mathscr{Z} \to \mathscr{P}_3$ be defined as follows:

$$r(u) = \begin{cases} e_0 = \bigwedge & \text{for } u < 0, \\ e_1 & \text{for } u = 0, \\ e_\omega = \bigvee & \text{for } u > 0. \end{cases}$$

Then

$$d_1 r(u) = \begin{cases} \bigvee & \text{for } u \geqslant 0, \\ \bigwedge & \text{otherwise;} \end{cases} \qquad d_2 r(u) = \begin{cases} \bigvee & \text{for } u > 0, \\ \bigwedge & \text{otherwise;} \end{cases}$$

$$r(u) = d_1 r(u) \cap e_1 \cup d_2 r(u),$$

$$j_0 r(u) = \begin{cases} \bigvee & \text{for } u < 0, \\ \bigwedge & \text{otherwise;} \end{cases} \qquad j_1 r(u) = \begin{cases} \bigvee & \text{for } u = 0, \\ \bigwedge & \text{otherwise.} \end{cases}$$

Let $\mathscr{R}$ be the set of real numbers and let $r\colon \mathscr{R}^2 \to \mathscr{P}_4$ be defined thus:

$$r(u_1, u_2) = \begin{cases} e_0 = \bigwedge & \text{if } u_1 < u_2, \\ e_1 & \text{if } u_1 = u_2, \\ e_2 & \text{if } u_2 < u_1 < u_2+1, \\ e_\omega = \bigvee & \text{if } u_2+1 \leqslant u_1. \end{cases}$$

Then

$$d_1 r(u_1, u_2) = \begin{cases} \bigvee & \text{if } u_2 \leqslant u_1, \\ \bigwedge & \text{otherwise;} \end{cases} \qquad d_2 r(u_1, u_2) = \begin{cases} \bigvee & \text{if } u_2 < u_1, \\ \bigwedge & \text{otherwise;} \end{cases}$$

$$d_3 r(u_1, u_2) = \begin{cases} \bigvee & \text{if } u_2+1 \leqslant u_1, \\ \bigwedge & \text{otherwise;} \end{cases}$$

$$j_0 r(u_1, u_2) = \begin{cases} \bigvee & \text{if } u_1 < u_2, \\ \bigwedge & \text{otherwise,} \end{cases} \qquad j_1 r(u_1, u_2) = \begin{cases} \bigvee & \text{if } u_1 = u_2, \\ \bigwedge & \text{otherwise;} \end{cases}$$

$$j_2 r(u_1, u_2) = \begin{cases} \bigvee & \text{if } u_2 < u_1 < u_2+1, \\ \bigwedge & \text{otherwise.} \end{cases}$$

Moreover,

$$r(u_1, u_2) = d_1 r(u_1, u_2) \cap e_1 \cup d_2 r(u_1, u_2) \cap e_2 \cup d_3 r(u_1, u_2).$$

A system $\mathfrak{U} = (U, (o_i)_{i \in I}, (r_k)_{k \in I'})$, where $U \neq \emptyset$ and

(1′) for each $i \in I$, o_i is a n_i-argument operation on U, $n_i \in \mathscr{N}$,

(2′) for each $k \in I'$, r_k is a m_k-valued ν_k-argument relation on U,

$$m_k \geqslant 2, \quad m_k \in \mathcal{N}_0, \quad \nu_k \in \mathcal{N}_0,$$

is said to be a *mixed-valued relational system under* U.

It is worth mentioning that each mixed-valued relational system determines a relational system if each ν_k-argument m_k-valued relation r_k is replaced by $m_k - 1$ ν_k-argument two-valued relations $\boldsymbol{d}_1 r_k, \ldots, \boldsymbol{d}_{m_k-1} r_k$, as defined by (1), (2).

§ 7. Formalized ω^+-valued algorithmic languages

Formalized ω^+-valued algorithmic languages are extensions of algorithmic ones.

Let $A = V_i \cup V_0^{(2)} \cup \Phi \cup P^{(2)} \cup L_0 \cup L_1 \cup L_2 \cup Q \cup \Pi \cup U_0$ be the alphabet of an algorithmic language

$$\mathcal{L} = \langle A, \boldsymbol{T} \cup \boldsymbol{F}^o \cup \boldsymbol{S} \cup \boldsymbol{FS} \cup \boldsymbol{FST} \cup \boldsymbol{FSF} \rangle.$$

We assume that $V_0^{(2)}$ is an enumerable set of two-valued propositional variables, to be denoted by p^2, q^2 with indices if necessary, $P^{(2)}$ is a set of two-valued predicates, to be denoted by ϱ^2 with indices if necessary, L_0 is composed of two elements denoted by E_0 and E_ω instead of **0** and **1**, respectively, and corresponding to any false statement and to any true statement, and V_i, Φ, L_1, L_2, Q, Π, U_0 are defined as in I, § 2. Let us set $A_2 = A \cup \{D_1\}$ and

$$A_{n+1} = A_n \cup V_0^{(n+1)} \cup P^{(n+1)} \cup \{E_{n-1}\} \cup \{D_n\}, \quad \text{for} \quad n \geqslant 2,$$

where $V_0^{(n+1)}$ is an enumerable set of $(n+1)$-valued propositional variables, to be denoted by p^{n+1}, q^{n+1} with indices if necessary, $P^{(n+1)}$ is a set of $(n+1)$-valued predicates, to be denoted by ϱ^{n+1} with indices if necessary, E_{n-1} is a propositiona constant and D_n is a unary connective. We define $A_\omega = \bigcup_{n=2}^{\infty} A_n$.

The *algorithmic* ω^+*-valued language under* A_ω is a system

$$\mathcal{L}_\omega = \langle A_\omega, \boldsymbol{T} \cup \boldsymbol{F}_\omega^o \cup \boldsymbol{S}_\omega \cup \boldsymbol{F}_\omega \boldsymbol{S} \cup \boldsymbol{F}_\omega \boldsymbol{ST} \cup \boldsymbol{F}_\omega \boldsymbol{SF} \rangle$$

to be defined as follows. The set $\boldsymbol{T}$ of terms coincides with the set $\boldsymbol{T}$ in $\mathcal{L}$ (see I, § 2).

The set $\boldsymbol{F}_\omega^o$ of open formulas in $\mathcal{L}_\omega$ is the least set such that

(f1) $p^m \in \boldsymbol{F}_\omega^o$ for each $p^m \in V_0^{(m)}$, $m \geqslant 2$, $m \in \mathcal{N}_0$,

(f2) $E_i \in \boldsymbol{F}_\omega^o$ for $0 \leqslant i \leqslant \omega$,

(f3) $\varrho^m(\tau_1, \ldots, \tau_k) \in \boldsymbol{F}_\omega^o$ for each k-argument $(k \in \mathcal{N}_0)$ $\varrho^m \in P^{(m)}$, $m \geqslant 2$, $m \in \mathcal{N}_0$, and any $\tau_1, \ldots, \tau_k \in \boldsymbol{T}$,

(f4) if $\alpha, \beta \in \boldsymbol{F}_\omega^o$, then $\neg\alpha$, $D_i\alpha$ for $i \in \mathcal{N}_0$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \Rightarrow \beta)$ are in $\boldsymbol{F}_\omega^o$.

Open formulas satisfying one of conditions (f1), (f2), (f3) are called *atomic*. The set of atomic open formulas will be denoted by $\boldsymbol{F}_\omega^{at}$. Among the formulas in $\boldsymbol{F}_\omega^o$ we distinguish the Boolean ones. They assume under any realization and any valuation only Boolean values $\swarrow$, $\nearrow$. The set $\boldsymbol{BF}_\omega^o$ of Boolean open formulas is the least set such that: (1) $V_0^{(2)} \subset \boldsymbol{BF}_\omega^o$; (2) $E_0, E_\omega \in \boldsymbol{BF}_\omega^o$; (3) $\varrho^2(\tau_1, \ldots, \tau_k) \in \boldsymbol{BF}_\omega^o$ for each k-argument $(k \in \mathcal{N}_0)$ predicate $\varrho^2 \in P^{(2)}$ and any $\tau_1, \ldots, \tau_k \in \boldsymbol{T}$; (4) $\neg\alpha$, $D_i\alpha$ for $i \in \mathcal{N}_0$, $(\alpha \Rightarrow \beta)$ are in $\boldsymbol{BF}_\omega^o$ for any $\alpha \in \boldsymbol{F}_\omega^o$ and $\beta \in \boldsymbol{BF}_\omega^o$; (5) $(\alpha \vee \beta)$, $(\alpha \wedge \beta) \in \boldsymbol{BF}_\omega^o$ for $\alpha, \beta \in \boldsymbol{BF}_\omega^o$.

By $\mathrm{ord}(\alpha)$ for $\alpha \in \boldsymbol{F}_\omega^o$ we shall understand the least $m \geqslant 2$, $m \in \mathcal{N}_0$, such that α is composed of elements in A_m. For instance, $\mathrm{ord}(E_\omega) = 2$, $\mathrm{ord}\big(D_3(E_1 \Rightarrow p^4)\big) = 4$, $\mathrm{ord}\big(D_2 \varrho^2(x)\big) = 3$.

We shall write, for any $\alpha \in \boldsymbol{F}_\omega^o$, $J_0\alpha$ instead of $\neg D_1\alpha$, and $J_k\alpha$ for $k \in \mathcal{N}_0$ instead of $(\neg D_{k+1}\alpha \wedge D_k\alpha)$. It follows that if $\mathrm{ord}(\alpha) = m$, then $\mathrm{ord}(J_k\alpha) = m$ for $k = 0, \ldots, m-2$.

The set $\boldsymbol{S}_\omega$ of (mixed-valued) substitutions consists of the expressions

(1) $$[x_1/\tau_1 \ldots x_n/\tau_n \; p_1^{m_1}/\alpha_1 \ldots p_k^{m_k}/\alpha_k], \quad n, m \in \mathcal{N},$$

where $x_1, \ldots, x_n$ are different individual variables in V_i, $\tau_1, \ldots, \tau_n \in \boldsymbol{T}$, $p_i^{m_i}$, for $i = 1, \ldots, k$, are different m_i-valued propositional variables in $V_0^{(m_i)}$, $\alpha_i \in \boldsymbol{F}_\omega^o$ and satisfy the condition $\mathrm{ord}(\alpha_i) \leqslant m_i$.

The set $\boldsymbol{F}_\omega \boldsymbol{S}$ of programs is the least set such that

(fs1) $\boldsymbol{S}_\omega \subset \boldsymbol{F}_\omega \boldsymbol{S}$,

(fs2) if $K, M \in \boldsymbol{F}_\omega \boldsymbol{S}$, then $\circ[KM] \in \boldsymbol{F}_\omega \boldsymbol{S}$,

(fs3) if $\alpha \in \boldsymbol{BF}_\omega^o$ and $K, M \in \boldsymbol{F}_\omega \boldsymbol{S}$, then $\curlywedge[\alpha KM] \in \boldsymbol{F}_\omega \boldsymbol{S}$,

(fs4) if $\alpha \in \boldsymbol{F}_\omega^o$, $\mathrm{ord}(\alpha) = m > 2$, $\alpha \notin \boldsymbol{BF}_\omega^o$, and $K_0, \ldots, K_{m-2}, K_\omega \in \boldsymbol{F}_\omega \boldsymbol{S}$, then $\curlywedge[\alpha K_\omega K_{m-2} \ldots K_0] \in \boldsymbol{F}_\omega \boldsymbol{S}$,

(fs5) if $\alpha \in \boldsymbol{BF}_\omega^o$ and $K \in \boldsymbol{F}_\omega \boldsymbol{S}$, then $*[\alpha K] \in \boldsymbol{F}_\omega \boldsymbol{S}$.

The set $\boldsymbol{F}_\omega \boldsymbol{ST}$ of generalized terms is the least set such that

(fst1) $V_i \subset \boldsymbol{F}_\omega \boldsymbol{ST}$,

(fst2) if $\varphi \in \Phi$ is a k-argument functor $(k \in \mathcal{N})$, and $\tau_1, \ldots, \tau_k \in \boldsymbol{F}_\omega \boldsymbol{ST}$, then $\varphi(\tau_1, \ldots, \tau_k) \in \boldsymbol{F}_\omega \boldsymbol{ST}$,

(fst3) if $K \in \boldsymbol{F}_\omega \boldsymbol{S}$ and $\tau \in \boldsymbol{F}_\omega \boldsymbol{ST}$, then $K\tau \in \boldsymbol{F}_\omega \boldsymbol{ST}$.

The set $\boldsymbol{F}_\omega \boldsymbol{SF}$ of generalized formulas is the least set satisfying the following conditions:

(fsf1) $p^m \in \boldsymbol{F}_\omega \boldsymbol{SF}$ for each $p^m \in V_0^{(m)}$, $m \geqslant 2$, $m \in \mathcal{N}_0$,

(fsf2) $E_i \in \boldsymbol{F}_\omega \boldsymbol{SF}$ for $0 \leqslant i \leqslant \omega$,

(fsf3) $\varrho^m(\tau_1, \ldots, \tau_k) \in \boldsymbol{F}_\omega \boldsymbol{SF}$ for each k-argument $(k \in \mathcal{N}_0)$ m-valued predicate $\varrho^m \in P^{(m)}$, $m \geqslant 2$, $m \in \mathcal{N}_0$, and any $\tau_1, \ldots, \tau_k \in \boldsymbol{F}_\omega \boldsymbol{ST}$,

(fsf4) if $\alpha, \beta \in \boldsymbol{F}_\omega \boldsymbol{SF}$, then $\neg\alpha$, $D_i\alpha$ for $i \in \mathcal{N}_0$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \Rightarrow \beta)$ are in $\boldsymbol{F}_\omega \boldsymbol{SF}$,

(fsf5) if $K \in \boldsymbol{F}_\omega \boldsymbol{S}$ and $\alpha \in \boldsymbol{F}_\omega \boldsymbol{SF}$, then $K\alpha$, $\bigcup K\alpha$, $\bigcap K\alpha$ are in $\boldsymbol{F}_\omega \boldsymbol{SF}$.

Realizations of $\mathscr{L}_\omega$ in $U \neq \emptyset$ are mappings R which assign to every k-argument functor $\varphi \in \Phi$ ($k \in \mathscr{N}$) a function $\varphi_R\colon U^k \to U$ and to every k-argument ($k \in \mathscr{N}_0$) m-valued predicate $\varrho^m \in P^{(m)}$, $m \geqslant 2$, a k-argument m-valued relation $\varrho^m_R\colon U^k \to \mathscr{P}_m$ on U (see § 6). Notice that every realization of $\mathscr{L}_\omega$ in U determines a mixed-valued relational system

$$\mathfrak{U} = \langle U, (\varphi_R)_{\varphi\in\Phi}, (\varrho_R)_{\varrho\in P}\rangle,$$

where $P = \bigcup_{m=2}^{\infty} P^{(m)}$ (cf. § 6).

Valuations in U are mappings v assigning to each $x \in V_i$ an element $v(x) \in U$ and to each $p^m \in V_0^{(m)}$ ($m \geqslant 2$, $m \in \mathscr{N}_0$) an element $v(p^m) \in \mathscr{P}_m = \{e_0, \ldots, e_{m-2}, e_\omega\}$. The set of all valuations in U will be denoted by W_U.

For each term $\tau \in T$, $\tau_R(v)$ is defined by (t1R), (t2R) in I, § 3. The following equations extend R to $\boldsymbol{F}^o_\omega$.

(fr1) $p^m_R(v) = v(p^m)$, $p^m \in P^{(m)}$, $m \geqslant 2$, $m \in \mathscr{N}_0$,

(fr2) $E_{iR}(v) = e_i$, $0 \leqslant i \leqslant \omega$,

(fr3) $\varrho^m(\tau_1, \ldots, \tau_k)_R(v) = \varrho^m_R\big(\tau_{1R}(v), \ldots, \tau_{kR}(v)\big)$,

(fr4) $\neg\alpha_R(v) = -\alpha_R(v)$, $D_i\alpha_R(v) = \boldsymbol{d}_i\alpha_R(v)$ for $i \in \mathscr{N}_0$, $(\alpha\vee\beta)_R(v) = \alpha_R(v)\cup\beta_R(v)$, $(\alpha\wedge\beta)_R(v) = \alpha_R(v)\cap\beta_R(v)$, $(\alpha \Rightarrow \beta)_R(v) = \alpha_R(v) \to \beta_R(v)$, where the operations $-$, $\boldsymbol{d}_i$ for $i \in \mathscr{N}_0$, $\cup$, $\cap$, $\to$ are those in $\mathfrak{P}_\omega$ (see § 1).

The following statements are easy to prove by applying (fr1)–(fr4) and (1), (2), (3) in § 1.

7.1. *For each* $\alpha \in \boldsymbol{BF}^o_\omega$, $\alpha_R(v) \in \{|\!\diagup, \diagup\!|\}$.

7.2. *If* $\alpha \in \boldsymbol{F}^o_\omega$ *and* $\mathrm{ord}(\alpha) = m$, *then* $\alpha_R(v) \in \mathscr{P}_m$.

Programs in $\boldsymbol{F}_\omega\boldsymbol{S}$ are realized as partial mappings from W_U into W_U. If $V_i \cup \bigcup_{m=2}^{\infty} V_0^{(m)}$ is considered as a set of addresses and $U \cup \mathscr{P}_\omega$ as a set of data, then W_U may be considered as a set of state vectors. Thus programs in $\boldsymbol{F}_\omega\boldsymbol{S}$ are realized as partial mappings from a set of state vectors into itself. The exact definition is as follows:

(fsr1) if $s \in \boldsymbol{S}_\omega$ and s has form (1), then $s_R(v) = v'$, where $v'(x) = v(x)$ for $x \neq x_i$, $i = 1, \ldots, n$, $x \in V_i$, $v'(x_i) = \tau_{iR}(v)$, $i = 1, \ldots, n$, $v'(p) = v(p)$ for $p \neq p^{m_i}_i$, $i = 1, \ldots, k$, $p \in \bigcup_{m=2}^{\infty} V_0^{(m)}$ and $v(p^{m_i}_i) = \alpha_{iR}(v)$ for $i = 1, \ldots, k$.

(fsr2) $$\circ[KM]_R(v) = \begin{cases} M_R\big(K_R(v)\big) & \text{if this is defined,} \\ \text{undefined} & \text{otherwise;} \end{cases}$$

(fsr3) $$\leftthreetimes[\alpha KM]_R(v) = \begin{cases} K_R(v) & \text{if this is defined and } \alpha_R(v) = |\!\diagup, \\ M_R(v) & \text{if this is defined and } \alpha_R(v) = \diagup\!|, \\ \text{undefined} & \text{otherwise;} \end{cases}$$

(since $\alpha \in \boldsymbol{BF}^o_\omega$, by 7.1, $\alpha_R(v) \in \{|\!\diagup, \diagup\!|\}$);

(fsr4) $$\leftthreetimes[\alpha K_\omega K_{m-2} \ldots K_0]_R(v) = \begin{cases} K_{iR}(v) & \text{if this is defined and } \alpha_R(v) = e_i, \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where $i = 0, \ldots, m-2, \omega$ (since $\mathrm{ord}(\alpha) = m$, hence by 7.2, $\alpha_R(v) \in \mathscr{P}_m$);

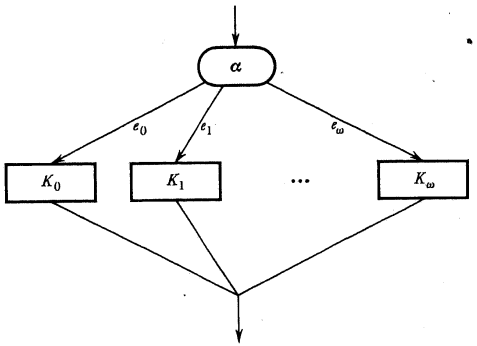
(fsr5) $$*[\alpha K]_R(v) = \begin{cases} K^i_R v, & \text{if } i \text{ is the least } j \in \mathscr{N} \text{ such that } \alpha_R\big(K^j_R(v)\big) \\ & = \diagup\!|, \text{ and all } K^j_R(v)\ (j \leqslant i) \text{ are defined,} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where $K^0_R(v) = v$, and $K^{j+1}_R(v) = K_R\big(K^j_R(v)\big)$ for $j \in \mathscr{N}_0$. Since $\alpha \in \boldsymbol{BF}^o_\omega$, $\alpha_R(\overline{v}) \in \{|\!\diagup, \diagup\!|\}$ for each $\overline{v} \in W_U$.

It follows from (fsr1), (fsr2), (fsr3) and (fsr5) that the translation of the $\boldsymbol{F}_\omega\boldsymbol{S}$-expressions: $s \in \boldsymbol{S}_\omega$, $\circ[KM]$, $\leftthreetimes[\alpha KM]$, $*[\alpha K]$ into an ALGOL-like language is analogous to that of $\boldsymbol{FS}$-expressions (see I, § 2).

Suppose that K_ω, $K_{m-2}, \ldots, K_0$ have been translated into programs Π_ω, $\Pi_{m-2}, \ldots, \Pi_0$, then $\leftthreetimes[\alpha K_\omega K_{m-2} \ldots K_0]$ may be translated into

case $D_{m-1}\alpha$; $J_{m-2}\alpha$; ...; $J_0\alpha$ **of begin** Π_ω; Π_{m-2}; ...; Π_0 **end**



For example, let us consider $\leftthreetimes[\varrho^4(xy)[x/0][x/y][y/1][x/y+1]]$ by the standard realization R in the set $\mathscr{R}$ of real numbers, where (cf. § 6)

$$\varrho^4_R(u_1, u_2) = \begin{cases} e_\omega & \text{if} \quad u_2+1 \leqslant u_1, \\ e_2 & \text{if} \quad u_2 < u_1 < u_2+1, \\ e_1 & \text{if} \quad u_1 = u_2, \\ e_0 & \text{if} \quad u_1 < u_2. \end{cases}$$

Then the program written above by this realization should be translated into

case $y+1 \leqslant x$; $y < x < y+1$ $x = y$; $x < y$ **of begin** $x := 0$; $x := y$; $y := 1$; $x := y+1$ **end**.

The extension of any realization R on $\boldsymbol{F}_\omega\boldsymbol{ST}$ is defined by means of (fst1R), (fst2R), (fst3R) in I, § 3.

In order to extend R to $\boldsymbol{F}_\omega\boldsymbol{SF}$ we adopt (fr1), (fr2), (fr4) and, moreover,

(fsfr3) $\varrho^m(\tau_1, \ldots, \tau_k)_R(v)$

$$= \begin{cases} \varrho^m_R(\tau_{1R}(v), \ldots, \tau_{k_R}(v)) & \text{if } \tau_{iR}(v) \text{ are defined for } i = 1, \ldots, k, \\ \nearrow & \text{otherwise;} \end{cases}$$

(fsfr5) $K\alpha_R(v) = \begin{cases} \alpha_R(K_R(v)) & \text{if } K_R(v) \text{ is defined,} \\ \nearrow & \text{otherwise,} \end{cases}$

$$\bigcup K\alpha_R(v) = \underset{i\in\mathcal{N}}{\text{l.u.b.}}\, K^i\alpha_R(v); \quad \bigcap K\alpha_R(v) = \underset{i\in N}{\text{g.l.b.}}\, K^i_R(v),$$

where $K^0\alpha = \alpha$, $K^{i+1}\alpha = KK^i\alpha$ for $i \in \mathcal{N}$, and l.u.b., g.l.b. denote the least upper bound and the greatest lower bound in $\mathfrak{P}_\omega$, respectively.

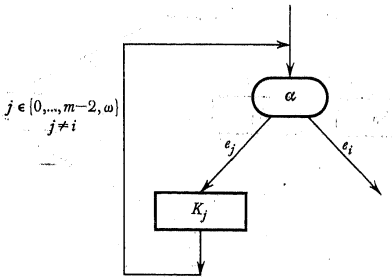
The following examples of definable program operations explain the sufficiency of including $*$ in the sense adopted by (fs5) and (fsr5) among the primitive ones.

EXAMPLE 1. For each m, $2 < m < \omega$, $i = 0, \ldots, m-2, \omega$, let us define $*^m_i$ as follows: if $\alpha \in F^o_\omega$, $\mathrm{ord}(\alpha) = m$, $\alpha \notin \boldsymbol{BF}^o_\omega$ and $K_0, \ldots, K_{m-2}$, $K_\omega \in \boldsymbol{F}_\omega\boldsymbol{S}$, then

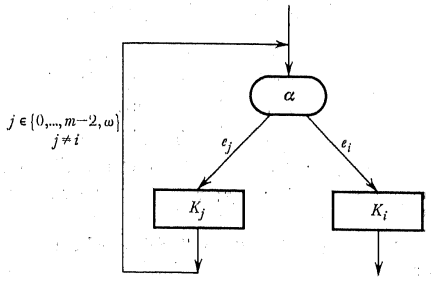
$$*^m_i[\alpha K_\omega K_{m-2} \ldots K_0] \overset{\mathrm{df}}{=} *[\neg J_i\alpha \veebar [\alpha K_\omega K_{m-2} \ldots K_0]] \quad \text{for} \quad i = 0, \ldots, m-2,$$

$$*^m_\omega[\alpha K_\omega K_{m-2} \ldots K_0] \overset{\mathrm{df}}{=} *[\neg D_{m-1}\alpha \veebar [\alpha K_\omega K_{m-2} \ldots K_0]].$$

These programs in $\boldsymbol{F}_\omega\boldsymbol{S}$ may be illustrated thus:



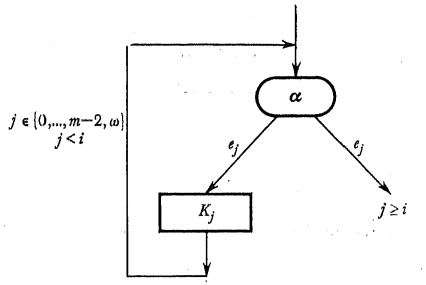
Observe that $\circ\,[*^m_i[\alpha K_\omega K_{m-2} \ldots K_0]K_i]$ may be illustrated thus:



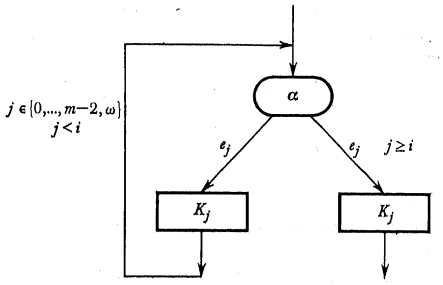
EXAMPLE 2. For each m, $2 < m < \omega$, let us define $*^m_{\geqslant i}$, $i = 1, \ldots, m-2$, as follows: if $\alpha \in F^o_\omega$, $\mathrm{ord}(\alpha) = m$, $\alpha \notin \boldsymbol{BF}^o_m$, and $K_0, \ldots, K_{m-2}$, $K_\omega \in \boldsymbol{F}^1_\omega\boldsymbol{S}$, then

$$*^m_{\geqslant i}[\alpha K_\omega K_{m-2} \ldots K_0] \overset{\mathrm{df}}{=} *[\neg D_i\alpha \veebar [\alpha K_\omega K_{m-2} \ldots K_0]].$$

These programs in $\boldsymbol{F}_\omega\boldsymbol{S}$ may be illustrated thus:



Observe that $\circ[*^m_{\geqslant i}[\alpha K_\omega K_{m-2} \ldots K_0] \veebar [\alpha K_\omega K_{m-2} \ldots K_0]]$ may be illustrated as follows:



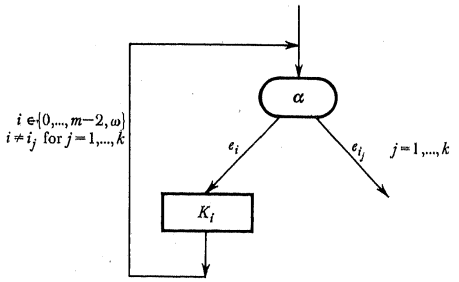
EXAMPLE 3. Let $0 \leqslant i_1 < \ldots < i_k \leqslant m-2$ or $i_k = \omega$ and $i_{k-1} \leqslant m-2$, where $2 < m < \omega$. Define $*^m_{i_1,\ldots,i_k}$ as follows: If $\alpha \in F^o_\omega$, $\mathrm{ord}(\alpha) = m$, $\alpha \notin \boldsymbol{BF}^o_\omega$ and $K_0, \ldots, K_{m-2}$, $K_\omega \in \boldsymbol{F}_\omega\boldsymbol{S}$, then

$$*^m_{i_1,\ldots,i_k}[\alpha K_\omega K_{m-2} \ldots K_0] \overset{\mathrm{df}}{=}$$

$$*\big[\neg\big(J_{i_1}\alpha \vee (\ldots \vee (\neg J_{i_{k-1}}\alpha \vee J_{i_k}\alpha)\ldots)\big) \veebar [\alpha K_\omega K_{m-2} \ldots K_0]\big] \quad \text{for} \quad i_k \neq \omega,$$

and if $i_k = \omega$ then instead of $J_{i_k}\alpha$ we set $D_{m-1}\alpha$ in the definition written above.

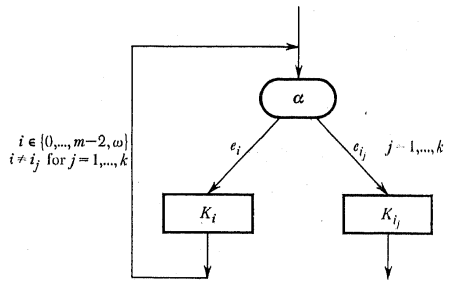
These programs in $F_\omega S$ may be illustrated as follows:



Observe that

$$\circ\,[*^m_{i_1,\dots,i_k}[\alpha K_\omega K_{m-2} \dots K_0] \curlywedge [\alpha K_\omega K_{m-2} \dots K_0]]$$

may be illustrated thus:



Given a formalized ω^+-valued algorithmic language $\mathscr{L}_\omega$, we can introduce a semantic consequence operation Cn in $\mathscr{L}_\omega$ as follows: A realization R of $\mathscr{L}_\omega$ in $U \neq \emptyset$ is said to be a *model of* $\alpha \in F_\omega SF$ if $\alpha_R(v) = \bigvee$ for each $v \in W_U$. A realization R of $\mathscr{L}_\omega$ is said to be a *model of* $\mathscr{A} \subset F_\omega SF$ if it is a model of each $\alpha \in \mathscr{A}$. A generalized formula $\alpha \in F_\omega SF$ is said to be a *tautology in* $\mathscr{L}_\omega$ if every realization of $\mathscr{L}_\omega$ is a model of α.

A generalized formula α is said to be a *semantic consequence of* $\mathscr{A} \subset F_\omega SF$ if every model of $\mathscr{A}$ is a model of α. This will be written as $\mathscr{A} \models_\omega \alpha$. We set, for each $\mathscr{A} \subset F_\omega SF$,

$$Cn(\mathscr{A}) = \{\alpha \in F_\omega SF\colon \mathscr{A} \models_\omega \alpha\}.$$

In particular, $Cn(\emptyset)$ is the set of all tautologies in $\mathscr{L}_\omega$.

The systems $\mathscr{S}_\omega = \langle \mathscr{L}_\omega, Cn\rangle$, where $\mathscr{L}_\omega$ is a ω^+-valued algorithmic language and Cn the semantic consequence operation in $\mathscr{L}_\omega$ as defined above, are systems of ω^+-valued algorithmic logic.

Each $\mathscr{L}_\omega$ determines uniquely, for any $m \geqslant 2$, $m \in \mathscr{N}_0$, a mixed-valued algorithmic language $\mathscr{L}_m = \langle A_m, T \cup F^o_m \cup S_m \cup F_m S \cup F_m ST \cup F_m SF\rangle$, where F^o_m, S_m, $F_m S$, $F_m ST$, $F_m SF$ are obtained by restricting F^o_ω, S_ω, $F_\omega S$, $F_\omega ST$, $F_\omega SF$, respectively, to the expressions which are constructed of elements in A_m.

Realizations of $\mathscr{L}_m$ are defined analogously to those of $\mathscr{L}_\omega$ and the restrictions of the realizations of $\mathscr{L}_\omega$ to $\mathscr{L}_m$ are realizations of $\mathscr{L}_m$. The same concerns the valuations. On applying the realizations of $\mathscr{L}_m$ we define the models of generalized formula $\alpha \in F_m SF$, of any $\mathscr{A} \subset F_m SF$, the notion of a tautology in $\mathscr{L}_m$ and a semantic consequence operation Cn in $\mathscr{L}_m$ in the same way as for $\mathscr{L}_\omega$. We write $\mathscr{A} \models_m \alpha$ for any $\mathscr{A} \subset F_m SF$ and $\alpha \in F_m SF$ if α is a semantic consequence of $\mathscr{A}$ with respect to Cn in $\mathscr{L}_m$.

The following statement holds.

7.3. *If* $\alpha \in F_m SF$ *and* $\mathscr{A} \subset F_m SF$, *then*

$$\mathscr{A} \models_m \alpha \quad \textit{if and only if} \quad \mathscr{A} \models_\omega \alpha.$$

The systems $\mathscr{S}_m = \langle \mathscr{L}_m, Cn\rangle$, where $\mathscr{L}_m$ is a mixed-valued algorithmic language and Cn the semantic consequence operation in $\mathscr{L}_m$ are said to be *systems of mixed-valued (with values restricted to m) algorithmic logic.*

It is worth mentioning that investigations and results concerning algorithmic logic and its applications to the theory of programs as developed in Chapters I–VI may be generalized to ω^+-valued algorithmic logic and to mixed-valued algorithmic logics with values restricted to $m \geqslant 2$.

Moreover, it is possible to extend ***EAL*** to the extended ω^+-valued algorithmic logic, the first version of which has been formulated in [24] and [25], by introducing to its formalized languages m-valued propositional variables and m-valued predicates for $2 \leqslant m < \omega$, and by adopting S_ω instead of S as well as on generalizing branching $\curlywedge$. In instructions and procedures $s \in S_\omega$ and generalized branchings may occur. A formalization of the extended ω^+-valued algorithmic logic may be given and a completeness theorem analogous to 5.1 holds.

§ 8. Formalization of the ω^+-valued algorithmic logic

Adopt the following notation. For any substitution $s \in S_\omega$ in form (1) § 7, and any well-formed expression Θ in $\mathscr{L}_\omega$, let $\bar{s}\Theta$ denote the expression obtained from Θ by the simultaneous replacement of x_i, $i = 1, \dots, n$, by τ_i and of $p^{m_i}_i$, $i = 1, \dots, k$, by α_i. The same concerns $s \in S_m$ and Θ in $\mathscr{L}_m$.

For every $\alpha \in F_\omega SF$, let $\mathrm{ord}(\alpha)$ be the least n, $n \geqslant 2$, such that $\alpha \in F_n SF$.

8.1. *There is a mapping* χ *effectively defined which assigns to each* $\varrho^m(\tau_1 \dots \tau_k)$, *where* $\tau_1, \dots, \tau_k \in F_\omega ST$, *a generalized formula* $K\varrho^m(\tau'_1 \dots \tau'_k)$, *where* $K \in F_\omega S$ *and*

$\tau'_1, \ldots, \tau'_k \in \boldsymbol{T}$, *such that for every realization* R *of* $\mathscr{L}_\omega$ *in any* $U \neq \emptyset$ *and every* $v \in W_U$,

$$\varrho^m(\tau_1 \ldots \tau_k)_R(v) = \chi\big(\varrho^m(\tau_1 \ldots \tau_k)\big)_R(v).$$

Moreover, if $\tau_1, \ldots, \tau_k \in \boldsymbol{F}_n\boldsymbol{ST}$, *then* $K \in \boldsymbol{F}_n\boldsymbol{S}$.

For a proof see [22].

The logical axiom schemes in $\mathscr{L}_\omega$ (in $\mathscr{L}_m$) are:

Group A—axiom schemes (A_1)–(A_{10}) in § 5.

Group B—axiom schemes (B_1)–(B_6), (B_8), (B_9) in § 5 (where for $\mathscr{L}_m$, $i, j \in \{0, \ldots, m-1\}$ and $k \in \{0, \ldots, m-2, \omega\}$), and

(B_n) $\Big(\alpha \Leftrightarrow \big((D_1\alpha \wedge E_1) \vee \big(\ldots \vee \big((D_{n-2}\alpha \wedge E_{n-2}) \vee D_{n-1}\alpha\big)\ldots\big)\big)\Big)$ for every $\alpha \in \boldsymbol{F}_\omega\boldsymbol{SF}$ $(\alpha \in \boldsymbol{F}_m\boldsymbol{SF})$ such that $\operatorname{ord}(\alpha) = n$ $(n \leqslant m$ for $\mathscr{L}_m)$.

Group C—additional axiom schemes for ω^+-valued algorithmic logic (for the mixed-valued algorithmic logic with values restricted to m).

(C_1) $(s\alpha \Leftrightarrow \bar{s}\alpha)$ for $s \in \boldsymbol{S}_\omega$, $\alpha \in \boldsymbol{F}^{at}_\omega$ $(s \in \boldsymbol{S}_m, \alpha \in \boldsymbol{F}^{at}_\omega \cap \boldsymbol{F}^o_m)$,

(C_2) $\big(K\varrho^n(\tau_1, \ldots, \tau_k) \Leftrightarrow \varrho^n(K\tau_1, \ldots, K\tau_k)\big)$ $(n \leqslant m, K \in \boldsymbol{F}_m\boldsymbol{S}, \tau_1, \ldots, \tau_k \in \boldsymbol{F}_m\boldsymbol{ST})$,

(C_3) $\big(\chi\varrho^n(\tau_1, \ldots, \tau_k) \Leftrightarrow \varrho^n(\tau_1, \ldots, \tau_k)\big)$ $(n \leqslant m, \tau_1, \ldots, \tau_k \in \boldsymbol{F}_m\boldsymbol{ST})$,

(C_4) $\big(K(\alpha \vee \beta) \Leftrightarrow (K\alpha \vee K\beta)\big)$,

(C_5) $\big(K(\alpha \wedge \beta) \Leftrightarrow (K\alpha \wedge K\beta)\big)$,

(C_6) $\big(K\neg\alpha \Rightarrow \neg K\alpha\big)$,

(C_7) $\big(KE_\omega \Rightarrow (\neg K\alpha \Rightarrow K\neg\alpha)\big)$,

(C_8) $\big(K(\alpha \Rightarrow \beta) \Rightarrow (K\alpha \Rightarrow K\beta)\big)$,

(C_9) $\big(KE_\omega \Rightarrow \big((K\alpha \Rightarrow K\beta) \Rightarrow K(\alpha \Rightarrow \beta)\big)\big)$,

(C_{10}) $(KD_i\alpha \Leftrightarrow D_iK\alpha)$, $i \in \mathscr{N}_0$ $(i \in \{1, \ldots, m-1\})$,

(C_{11}) $\big(M \bigcup K\alpha \Leftrightarrow (M\alpha \vee M \bigcup KK\alpha)\big)$,

(C_{12}) $\big(M \bigcap K\alpha \Leftrightarrow (M\alpha \wedge M \bigcap KK\alpha)\big)$,

(C_{13}) $(\circ[KM]\alpha \Leftrightarrow KM\alpha)$,

(C_{14}) $\big(\veebar[\alpha KM]\beta \Leftrightarrow \big((\alpha \wedge K\beta) \vee (\neg\alpha \wedge M\beta)\big)\big)$, $\alpha \in \boldsymbol{BF}^o_\omega$ $(\alpha \in \boldsymbol{BF}^o_\omega \cap \boldsymbol{F}^o_m)$,

(C_{15}) $\Big(\veebar[K_\omega K_{n-2} \ldots K_0]\beta \Leftrightarrow \big((D_{n-1}\alpha \wedge K_\omega\beta) \vee \big((J_{n-2}\alpha \wedge K_{n-2}\beta) \vee \ldots \vee (J_0\alpha \wedge K_0\beta)\ldots\big)\big)\Big)$, $\alpha \in \boldsymbol{F}^o_\omega$, $\operatorname{ord}(\alpha) = n$, $\alpha \notin \boldsymbol{BF}^o_\omega$ $(\alpha \in \boldsymbol{F}^o_m$, $\operatorname{ord}(\alpha) = n \leqslant m$, $\alpha \notin \boldsymbol{BF}^o_\omega)$,

(C_{16}) $\big(*[\alpha K]\beta \Leftrightarrow \bigcup\veebar[\alpha K[\ ]](\neg\alpha \wedge \beta)\big)$, $\alpha \in \boldsymbol{BF}^o_\omega$ $(\alpha \in \boldsymbol{BF}^o_\omega \cap \boldsymbol{F}^o_m)$.

In schemes (C_4)–(C_{16}) $K, M \in \boldsymbol{F}_\omega\boldsymbol{S}$ and $\alpha, \beta \in \boldsymbol{F}_\omega\boldsymbol{SF}$ $(K, M \in \boldsymbol{F}_m\boldsymbol{S}, \alpha, \beta \in \boldsymbol{F}_m\boldsymbol{SF})$. We adopt the following rules of inference in $\mathscr{L}_\omega$ (in $\mathscr{L}_m$): (r1), (r3), (r4) in I, § 11, the rule (r2) $\dfrac{\alpha,\ KE_\omega}{K\alpha}$ being analogous to (r2) in I, § 11, and moreover the following one

(r5) $\dfrac{\alpha}{D_i\alpha,\ i \in \mathscr{N}_0}$ $(1 \leqslant i \leqslant m-1$, for $\mathscr{L}_m)$.

For any set $\mathscr{A} \subset \boldsymbol{F}_\omega\boldsymbol{SF}$ $(\mathscr{A} \subset \boldsymbol{F}_m\boldsymbol{SF})$ and $\alpha \in \boldsymbol{F}_\omega\boldsymbol{SF}$ $(\alpha \in \boldsymbol{F}_m\boldsymbol{SF})$ we shall write $\mathscr{A} \underset{\omega}{\vdash} \alpha$ $(\mathscr{A} \underset{m}{\vdash} \alpha)$, if α belongs to the least set of generalized formulas in $\boldsymbol{F}_\omega\boldsymbol{SF}$ (in $\boldsymbol{F}_m\boldsymbol{SF}$) containing the union of $\mathscr{A}$ and of the set of logical axioms in $\mathscr{L}_\omega$ (in $\mathscr{L}_m$) and closed with respect to the rules of inference in $\mathscr{L}_\omega$ (in $\mathscr{L}_m$). Roughly speaking, we shall write $\mathscr{A} \underset{\omega}{\vdash} \alpha$ $(\mathscr{A} \underset{m}{\vdash} \alpha)$ if α can be deduced from $\mathscr{A}$ and the logical axioms in $\mathscr{L}_\omega$ (in $\mathscr{L}_m$) by means of rules of inference adopted in $\mathscr{L}_\omega$ (in $\mathscr{L}_m$).

The following completeness theorems hold.

8.2. *For every set* $\mathscr{A} \subset \boldsymbol{F}_\omega\boldsymbol{SF}$ *and* $\alpha \in \boldsymbol{F}_\omega\boldsymbol{SF}$,

$$\mathscr{A} \underset{\omega}{\models} \alpha \quad \textit{if and only if} \quad \mathscr{A} \underset{\omega}{\vdash} \alpha.$$

In particular, α *is a tautology in* $\mathscr{L}_\omega$ *if and only if* α *can be deduced from the logical axioms in* $\mathscr{L}_\omega$ *by means of rules of inference adopted in* $\mathscr{L}_\omega$.

For a proof of 8.2 the reader is referred to [23].

8.3. *For every set* $\mathscr{A} \subset \boldsymbol{F}_m\boldsymbol{SF}$ *and* $\alpha \in \boldsymbol{F}_m\boldsymbol{SF}$,

$$\mathscr{A} \underset{m}{\models} \alpha \quad \textit{if and only if} \quad \mathscr{A} \underset{m}{\vdash} \alpha.$$

In particular, α *is a tautology in* $\mathscr{L}_m$ *if and only if* α *can be deduced from the logical axioms in* $\mathscr{L}_m$ *by means of rules of inference adopted in* $\mathscr{L}_m$.

Theorem 8.3 can be proved in the same way as 8.2.

It follows from 7.3, 8.2 and 8.3 that

8.4. *For every set* $\mathscr{A} \subset \boldsymbol{F}_m\boldsymbol{SF}$ *and* $\alpha \in \boldsymbol{F}_m\boldsymbol{SF}$,

$$\mathscr{A} \underset{\omega}{\vdash} \alpha \quad \textit{if and only if} \quad \mathscr{A} \underset{m}{\vdash} \alpha.$$

Theorem 8.4 may be treated as a weak separation theorem for formalized systems of ω^+-valued algorithmic logic.

References

References are divided into two parts. The first contains almost complete bibliography of works on algorithmic logic. The second part contains the other references.

Part I

[1] Banachowski, L., D. Szczepańska, W. Wassersztrum, J. Żurek, *Automatic verification of program correctness*, Problemy węzłowe, Warsaw University 1974 (in Polish).

[2] Banachowski, L., *Formalization of the notion of data structures and programs on data structures*, Reports of the Warsaw University Computation Center 46 (1974).

[3] —, *Modular approach to the logical theory of programs*, Proc. Inter. Symp. Math. Found. Comp. Sci., Warsaw 1974.

[4] —, *An axiomatic approach to the theory of data structures*, Bull. Acad. Polon. Sci., Sér. Math. to appear.

[5] —, *Extended algorithmic logic and properties of programs*, ibid. to appear.

[6] —, *Modular properties of programs*, ibid. to appear.

[7] Banachowski, L., *Investigations of properties of programs by means of the extended algorithmic logic*, doctoral dissertation, Warsaw University 1975 (non-published).
[8] Dańko, W., *Not programmable function defined by a procedure*, Bull. Acad. Polon. Sci., Sér. Math. 22 (1974), 587–594.
[9] Grabowski, M., *The set of all tautologies of the zero-order algorithmic logic is decidable*, ibid. 20 (1972), 575–582.
[10] Kreczmar, A., *The set of all tautologies of algorithmic logic is hyperarithmetical*, ibid. 21 (1971), 781–783.
[11] —, *Degree of recursive unsolvability of algorithmic logic*, ibid. 20 (1972), 615–617.
[12] —, *Effectivity problems of algorithmic logic*, Automata, Languages and Programming, Lecture Notes in Computer Science 14, Springer-Verlag 1974, 584–600.
[13] —, *Effectivity problems of algorithmic logic*, doctoral dissertation, Warsaw University 1973.
[14] Góraj, A., G. Mirkowska, A. Paluszkiewicz, *On the notion of description of program*, Bull. Acad. Polon. Sci. Sér. Math. 18 (1970), 499–505.
[15] Mirkowska, G., *On formalized systems of algorithmic logic*, ibid. 22 (1974), 421–428.
[16] —, *Herbrand theorem in algorithmic logic*, ibid. 22 (1974), 539–543.
[17] —, *Algorithmic logic and its applications in programs theory*, doctoral dissertation, Warsaw University 1972 (non-published).
[18] Perkowska, E., *On algorithmic m-valued logics*, Bull. Acad. Polon. Sci. Sér. Math. 20 (1972), 717–719.
[19] —, *Theorem on normal form of a program*, ibid. 22 (1974), 439–441.
[20] Rasiowa, H., *On logical structure of programs*, Proc. Int. Symp. Math. Found. Comp. Sci. Warsaw–Jabłonna 1972; also in Bull. Acad. Polon. Sci., Sér. Math. 20 (1972), 319–324.
[21] —, *On logical structure of mix-valued programs and m-valued algorithmic logic*, ibid. 21 (1973), 451–458.
[22] —, *Formalized ω^+-valued algorithmic systems*, ibid. 559–565.
[23] —, *A simplified formalization of ω^+-valued algorithmic logic*, ibid. 22 (1974), 595–603.
[24] —, *Extended ω^+-valued algorithmic logic*, ibid. 605–610.
[25] —, *On ω^+-valued algorithmic logic and related problems*, CC PAS Reports (1974).
[26] —, *ω^+-valued algorithmic logic as a tool to investigate procedures*, Intern. Symp. and Summer School on MFCS Jadwisin 1974, Lecture Notes in Computer Science, Springer-Verlag.
[27] Salwicki, A., *A formalization of the notion of a program*, doctoral dissertation, Warsaw University 1969.
[28] —, *Formalized algorithmic languages*, Bull. Acad. Polon. Sci., Sér. Math. 18 (1970), 227–232.
[29] —, *On the equivalence of FS-expressions and programs*, ibid. 275–278.
[30] —, *On interpolation theorem in algorithmic logic*, internal report 1973 (in Polish).
[31] —, *Programmability and recursiveness* (*an application of algorithmic logic to procedures*), Diss. Math. to appear.

Part II

[32] de Bakker, J.W., W.P. de Roever, *Calculus for recursive program schemes*, Stichting Mathematisch Centrum MR 131/72, Amsterdam 1972.
[33] Blikle, A., A. Mazurkiewicz, *An algebraic approach to the theory of programs algorithms, languages and recursiveness*, Proc. Intern. Symp. and Summer School on MFCS Warsaw–Jabłonna 1972, CC PAS Reports, 1972.
[34] Engeler, E., *Algorithmic properties of structures*, Math. Syst. Theory 1 (1967), 183–195.
[35] —, *Remarks on the theory of geometrical constructions*, Syntax and Semantics of Infinitary Languages, Lecture Notes 72, Springer-Verlag 1968.
[36] —, *Algorithmic definability*, Proc. Intern. Symp. and Summer School on MFCS Warsaw–Jabłonna 1972.

[37] Floyd, R.W., *Assignings meanings to programs*, Proc. Symp. App. Math. AMS 19 (1967).
[38] Hoare, C.A., *An axiomatic basis for computer programming*, CACM 12 (1969), 576–583.
[39] Igarashi, S., *An axiomatic approach to the equivalence problems of algorithms with applications*, Rep. Com. CEN University Tokyo 1 (1968).
[40] King, J.C., *A program verifier*, Proc. IFIP 1971, 234–249.
[41] —, R.W. Floyd, *An interpretation oriented theorem prover over integers*, JCSS 6 (1972), 305–323.
[42] Knuth, D.E., R.W. Floyd, *Notes on avoiding "go to" statements*, Inf. Proc. Lett. 1 (1971), 23–37.
[43] Lopez-Escobar, E.G., *The interpolation theorem for denumerably long formulas*, Fund. Math. 57 (1968), 253–272.
[44] Luckham, D.C., D.M. Park, M.S. Paterson, *On formalized computer programs*, JCSS 4 (1970), 220–249.
[45] Manna, Z., *The correctness of programs*, ibid. 3 (1969), 119–127.
[46] Mazur, S., *Computable analysis*, Diss. Math. 33 (1963).
[47] Mazurkiewicz, A., *Proving properties of processes*, Algorytmy 11 (1974), 5–21 (in Polish).
[48] Mc Carthy, J., *Towards a mathematical science of computation*, Proc. IFIP 1962, 21–34.
[49] Rasiowa, H., R. Sikorski, *Mathematics of Metamathematics*, 3rd ed., PWN, Warsaw 1970.
[50] Rasiowa, H., *On generalized Post algebras of order ω^+ and ω^+-valued predicate calculii*, Bull. Acad. Polon. Sci., Sér. Math. Astron. Phys. 21 (1973), 209–219.
[51] —, *Mixed-valued predicate calculi*, Studia Logica 34 (1975), 215–234.
[52] Rogers, Jr, H., *Theory of recursive functions and effective computability*, McGraw-Hill, New York 1967.
[53] Scott, D., *Outline of a mathematical theory of computation*, Oxford mon. PRG-2, Oxford University 1970.
[54] Sheperdson, J.C., H.E. Strugis, *Computability of recursive functions*, JACM 10 (1963), 217–255.
[55] Tarski, A., *A decision method for elementary algebra and geometry*, Berkeley 1951.