ANNALES SOCIETATIS MATHEMATICAE POLONAE
Series IV: FUNDAMENTA INFORMATICAE I. 1 (1977), 93-119

Investigations of properties of programs by means of the extended algorithmic logic I

LECH BANACHOWSKI

Institute of Informatics, University of Warsaw

Received March 10, 1976

AMS Categories: 68A85

Abstract. The present paper contains investigations concerning the semantic correctness of programs. Presented methods of analysis of programs are appropriate for every domain of computation. Algorithmic logic extended by classical quantifiers is a fundamental mathematical tool used in the paper. Interrelations between properties of programs and properties of descriptions of programs are studied (a description of a program is a mathematical model of the notion of a documentation of a program).

Introduction

The property of semantic correctness of programs is one of the basic notions considered in the theory of programming. After designing a program the question arises whether it meets the programmer's objectives. The objectives which the program is supposed to realize are usually defined by means of two formulas: the input formula and the output formula. A program K is said to be *semantically correct with respect to an input formula α and an output formula β* provided, for every input satisfying α, the program K halts and the output satisfies β.

For example, let us consider the following program:

M: **begin**
  $z := x$;
  $i := 0$;
  while $z \geqslant y$ **do**
    begin
    $z := z - y$;
    $i := i + 1$
    end
end

and its realization in the system of integers.

This program finds the quotient and the remainder of the division of a natural number x by a natural number $y \neq 0$. Speaking more precisely, the program M is correct with respect to the input formula $(x \geqslant 0 \wedge y > 0)$ and the output formula $(x = i \cdot y + z \wedge z < y \wedge i \geqslant 0)$.

Contemporary mathematical machines are able to check only the syntactic correctness of programs. Admittedly, a full mechanical verification of program correctness is not possible. Namely, Kreczmar [14], [15] has shown that the problem of correctness is recursively enumerable for the class of all models, it is recursively enumerable for the field of real numbers and it is in the class $\Pi_2 - \Sigma_2$ of the arithmetical hierarchy for the ordered field of real numbers and for the standard model of arithmetic.

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, it is considered to be correct and is passed to 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 acquire the certitude of correctness of the program, is of great importance (for an exposition of these problems see [27]).

One of the possibilities consists in demanding that the programmer supplies 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.

Another demand, which is easier to fulfil, is the one for supplying the complete net of subtasks for distinguished segments of the program, the so-called *description* or *documentation* of the program. This approach to program verification is called *Floyd's method* [8]. Attempts of mechanical verification of programs based on Floyd's method have been presented in [7], [12] and [13].

The description of programs corresponds to the modular method of their designing. 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 programs accomplishing the subtasks, by an appropriate putting them together we can obtain a program correct with respect to the overall task. This net of subtasks will be called a *description*. The programs accomplishing the subtasks will be called *modules of the main program*. The configuration of all modules will be called the *modular structure of the program*.

As in the case of the whole program, the task of each module will be defined by means of two formulas: its input and output formulas.

Coming back to the previous example, we can distinguish in the program M four modules.

M_e – the main program M
M_1 – $(z := x;$
$\quad i := 0)$
M_2 – **while** $z \geqslant y$ **do**
begin
$z := z - y;$
$i := i + 1$
end
M_{21} – $(z := z - y;$
$\quad i := i + 1)$

As a description explaining the subtasks of the modules we can take the following assignments:

to M_e	$(x \geqslant 0 \wedge y > 0)$	and	$(x = i \cdot y + z \wedge z < y \wedge i \geqslant 0)$,
to M_1	$(x \geqslant 0 \wedge y > 0)$	and	$(x \geqslant 0 \wedge y > 0 \wedge z = x \wedge i = 0)$,
to M_2	$(x \geqslant 0 \wedge y > 0 \wedge z = x \wedge i = 0)$	and	$(x = i \cdot y + z \wedge z < y \wedge i \geqslant 0)$,
to M_{21}	$(x = i \cdot y + z \wedge z < y \wedge i \geqslant 0)$	and	$(x = i \cdot y + z \wedge i \geqslant 0)$.

The main purpose of the present paper consists in an analysis of the notions of modular structure and that of description of a program on the ground of algorithmic logic.

It turns out that many different approaches to the theory of programming, as those in [8], [10] and [17], can be embedded and completed in a uniform way by means of algorithmic logic.

We shall use algorithmic logic with classical quantifiers, the so-called *extended algorithmic logic* (see [2], [4], [15]), in contrast with algorithmic logic without classical quantifiers (see [22], [24]). The presence of classical quantifiers has been caused by two reasons. First, the construction of the strongest consequent (introduced in [8] and [9]) requires an application of classical quantifiers. We shall prove that neither classical quantifiers nor the strongest consequent can be defined without classical quantifiers. Next, the defining tasks of programs often require the use of classical quantifiers. In the sequel we shall often omit the adjective "extended" in the full name "extended algorithmic logic".

The whole work consists of two parts: this paper and [6]. The latter position will be referred to as Part II.

This paper is devoted to presenting basic definitions and facts concerning algorithmic logic. After introductory definitions of a language of algorithmic logic and its semantics, the definability of the general iteration quantifier by means of remaining connectives is shown in §2. Next the notion of algorithmic logic is defined and basic facts of algor-

ithmic logic are proved: the theorem on the isomorphism of realizations, the theorem on the Lindenbaum algebra of an algorithmic theory, the theorem on extensionality, the lemma on simplified form of formulas, the completeness theorem and the deduction theorem. It turns out that every algorithmic formula can be reduced to a formula in the prenex normal form (§8).

Now we present briefly the contents of Part II. In Chapter I, §1 and §2 we study the properties of the strongest consequent and its iteration. §3 contains a precise definition of the notions of correctness and partial correctness of programs and a lemma on reducing the question of partial correctness to some property of the strongest consequent of the input formula.

Chapter II is concerned with the modular structure of programs. We start with precise definitions of the modular structure and a description of programs and with the important notion of compatibility between the modular structure and a description of programs. Like for programs, the notions of correctness and partial correctness (with respect to a description), are defined for the modular structure of programs. It turns out that every modular structure compatible with a description is partially correct with respect to that description (§2). This fact was earlier proved by Manna [16] for "go to" programs, by de Bakker [1] for programs with non-functional, parameter-free procedures and by Mazurkiewicz [19] for processes. In the set of all descriptions compatible with the modular structure of a program one can introduce the quasi-ordering of inclusion of descriptions, just as for formulas.

In algorithmic logic, for a given program one can define maximal descriptions reflecting all computations beginning with the initial state (ending with the final state) satisfying a given formula (§3). We obtain necessary and sufficient conditions for the modular structure to be partially correct (correct) with respect to a description. In §5 it is proved that every assignment of input and output formulas to a program can be extended to a description, the extension preserving correctness (or partial correctness). This fact is then employed to the construction of a complete system of proving partial correctness. A similar system, but not complete, has been given by Hoare [10]. In §6 we estimate the undecidability of properties of the modular structure with regard to the properties of programs. It turns out that the correctness of modular structures with respect to descriptions composed solely of open formulas is mutually recursively reducible to the correctness of programs with respect to open formulas. Having a program K and two formulas α and β, we can effectively find an equivalent program M and its description D such that K is correct with respect to α and β if and only if the modular structure of M is correct with respect to D. In contrast with the correctness prop-

erty, the partial correctness of modular structures with respect to descriptions composed solely of open formulas is mutually recursively reducible to the validity of open formulas. Hence, supplying a program with a description simplifies an examination only in the case of partial correctness. However, in order to fully verify the program, we additionally have to check the stop property of the program. The last section is concerned with representations of the properties of programs in the second order logic. Similar results for correctness and partial correctness of "go to" programs have been obtained by Manna [16]. As a corollary we obtain the partial Herbrand theorem for some types of algorithmic formulas.

I would like to thank Professor H. Rasiowa, Dr A. Salwicki, and Dr A. Kreczmar for their aid during the preparation of this paper.

Notation

N	the set of all natural numbers,
a^n	the composition of n copies of an expression a,
$a : b$	the expressions a and b are equal,
e	the empty expression,
l.u.b.	least upper bound,
g.l.b.	greatest lower bound,
iff	if and only if,
D_f	the domain of a partial function f.

EXTENDED ALGORITHMIC LOGIC

1. Algorithmic language and its realizations

The definitions of a formalized algorithmic language and of its realization are similar to those in [23], [24], [22] and [14].

The alphabet of a formalized algorithmic language consists of the following sets:

V_i	an enumerable sequence of individual variables. The individual variables will be denoted by the letters x, y, z, t, u, with indices if necessary.
V_0	an enumerable sequence of propositional variables. The propositional variables will be denoted by the letters p, q, r, with indices if necessary.
$\{\Phi_n\}_{n\in N}$	a family of at most enumerable sets of functors. Functors will be denoted by the letter φ, with indices if necessary. The elements of the set Φ_n will be called *n-argument functors*.
$\{\Psi_n\}_{n\in N}$	a family of at most enumerable sets of predicates. Predicates will be denoted by the letters ψ and ϱ, with indices

if necessary. The elements of the set Ψ_n will be called *n-argument predicates*. We assume that the set Ψ_2 contains a distinguished predicate denoted by "$=$" and called the *equality sign*.

$\{\mathbf{0}, \mathbf{1}, \neg, \vee, \wedge, \Rightarrow, \exists, \forall, \cup, \cap, *, \leq, \circ\}$
the set of logical and program connectives.

$\{(\ ), [\ ], /\}$
the set of auxilliary symbols.

We assume that all these sets are pairwise disjoint.

The language $\mathscr{L}$ of the extended algorithmic logic consists of an alphabet as described above and of the following sets of well-formed expressions:

T the set of classical terms. Terms will be denoted by the letters τ and μ, with indices if necessary;

F_0 the set of open formulas;

F_1 the set of formulas of the first order predicate calculus;

S the set of substitutions, i.e. expressions of the form $[x_1/w_1 \ldots x_n/w_n]$ where $x_1, \ldots, x_n$ are variables, $w_1, \ldots, w_n$ are expressions and, moreover, for each $i = 1, \ldots, n$, x_i is an individual variable iff w_i is a term and x_i is a propositional variable iff w_i is an open formula. Substitutions will be denoted by the letter s, with indices if necessary;

FS the set of programs defined as the least set of expressions satisfying the following conditions:

(FS_1) *every substitution is a program in* FS;

(FS_2) *if γ is an open formula, K and M are programs, then the expressions $\circ[K\ M]$, $\leq[\gamma K\ M]$ and $*[\gamma K]$ are programs in* FS.

Programs will be denoted by the letters K, L, M and P, with indices if necessary.

F the set of formulas defined as the least set of expressions satisfying the following conditions:

(F_1) *all the propositional variables and the signs* $\mathbf{0}$ *and* $\mathbf{1}$ *are formulas*;

(F_2) *if $n \geqslant 0$, ϱ is an n-argument predicate and $\tau_1, \ldots, \tau_n$ are terms, then $\varrho(\tau_1, \ldots, \tau_n)$ is a formula*;

(F_3) *if α and β are formulas, then $\neg\alpha$, $(\alpha \vee \beta)$, $(\alpha \wedge \beta)$, $(\alpha \Rightarrow \beta)$ are formulas*;

(F_4) *if α is a formula and x is an individual variable, then $(\exists x\alpha)$ and $(\forall x\alpha)$ are formulas*;

(F_5) *if K is a program and α is a formula, then $K\alpha$, $\bigcup K\alpha$ and $\bigcap K\alpha$ are formulas.*

Formulas will be denoted by the letters $\alpha, \beta, \gamma, \delta, a, b, c, d$ with indices if necessary.

Let J be a non-empty set and let $\mathfrak{B}_0 = \langle B_0, \neg, \vee, \wedge, \Rightarrow \rangle$ be a two-element Boolean algebra. The zero of $\mathfrak{B}_0$ will be denoted by $\bigwedge$ and the unit by $\bigvee$. Let W be the set of all valuations of the variables in the sets J and B_0, i.e. let $W = J^{V_i} \times B_0^{V_0}$.

If v is in W, x is an individual variable and j is an object of the universe J, then v_j^x denotes the following valuation:

$$v_j^x(z) = \begin{cases} j & \text{if} \quad z = x, \\ v(z) & \text{if} \quad z \neq x. \end{cases}$$

By a realization of the algorithmic language $\mathscr{L}$ in the sets J and B_0 we mean a mapping R such that:

if φ is an n-argument functor, then φ_R is an n-argument operation in J, i.e., φ_R: $J^n \to J$;

if ψ is an n-argument predicate, then ψ_R is an n-argument characteristic function of a relation in J, i.e., ψ_R: $J^n \to B_0$;

if τ is a term, then τ_R: $W \to J$ (see [23]);

if α is an open formula or a formula of the first order predicate calculus, then α_R: $W \to B_0$ (see [23]);

if s is a substitution then s_R: $W \to W$ (see [22]);

the realization of an arbitrary program K is defined by induction on the length of K. Namely, if partial functions K_R and M_R from W into W are realizations of programs K and M, respectively, then

$$\circ[K\ M]_R(v) = \begin{cases} M_R(K_R(v)) & \text{if} \quad v \in D_{K_R} \text{ and } K_R(v) \in D_{M_R}, \\ \text{undefined} & \text{otherwise;} \end{cases}$$

$$\veebar[\gamma K\ M]_R(v) = \begin{cases} K_R(v) & \text{if} \quad v \in D_{K_R} \text{ and } \gamma_R(v) = \bigvee, \\ M_R(v) & \text{if} \quad v \in D_{M_R} \text{ and } \gamma_R(v) = \bigwedge, \\ \text{undefined} & \text{otherwise;} \end{cases}$$

$$*[\gamma K]_R(v) = \begin{cases} K_R^i(v) & \text{if there exists a natural } i \text{ such that } K_R^i(v) \text{ is} \\ & \text{defined, } \gamma_R(K_R^i(v)) = \bigwedge \text{ and for each } j, \\ & 0 \leqslant j < i\text{: } \gamma_R(K_R^j(v)) = \bigvee, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

So $\circ[K\ M]$ is equivalent to the algolic compound statement **begin** K; M **end**, $\veebar[\gamma K\ M]$ is equivalent to the conditional statement **if** γ **then** K **else** M and finally $*[\gamma K]$ is equivalent to the while statement **while** γ **do** K.

The realization of a formula α in F is a mapping α_R: $W \to B_0$ and it is defined by induction on the length of α in the following way:

(F_{1R}) if p is a propositional variable, then

$$p_R(v) = v(p), \quad \mathbf{1}_R(v) = V \quad \text{and} \quad \mathbf{0}_R(v) = \Lambda;$$

(F_{2R}) if ϱ is an n-argument predicate and $\tau_1, \ldots, \tau_n$ are terms, then

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

(F_{3R}) if α and β are formulas, then

$$(\alpha \vee \beta)_R(v) = \alpha_R(v) \vee \beta_R(v),$$

$$(\alpha \wedge \beta)_R(v) = \alpha_R(v) \wedge \beta_R(v), \quad (\alpha \Rightarrow \beta)_R(v) = \alpha_R(v) \Rightarrow \beta_R(v) \quad \text{and}$$

$$(\neg \alpha)_R(v) = \neg \alpha_R(v);$$

(F_{4R}) if α is a formula and x is an individual variable, then

$$(\exists x \alpha)_R(v) = \underset{j \in J}{\text{l.u.b.}}\ \alpha_R(v_j^x) \quad \text{and} \quad (\forall x \alpha)_R(v) = \underset{j \in J}{\text{g.l.b.}}\ \alpha_R(v_j^x);$$

(F_{5R}) if α is a formula and K is a program, then

$$(K\alpha)_R(v) = \begin{cases} \alpha_R\big(K_R(v)\big) & \text{if} \quad v \in D_{K_R}, \\ \Lambda & \text{if} \quad v \notin D_{K_R}; \end{cases}$$

$$\Big(\bigcup K\alpha\Big)_R(v) = \underset{i \in N}{\text{l.u.b.}}\ (K^i\alpha)_R(v),$$

$$\Big(\bigcap K\alpha\Big)_R(v) = \underset{i \in N}{\text{g.l.b.}}\ (K^i\alpha)_R(v).$$

2. Basic denotations

In the sequel we assume that $\mathscr{L}$ denotes a fixed algorithmic language. We consider only realizations of this language.

Let w be a term or an open formula and let $s = [x_1/w_1 \ldots x_n/w_n]$ be a substitution. The expression obtained from w by simultaneously replacing all occurrences of the variables x_i by the expressions w_i, for $i = 1, \ldots, n$, will be denoted by $\overline{sw}$. In the notation used in [23], p. 152, we have

$$\overline{sw} = w(x_1/w_1, \ldots, x_n/w_n).$$

If α and β are formulas, then by $\alpha \Leftrightarrow \beta$ we shall denote the formula $\big((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)\big)$.

By $V(w)$ we shall denote the set of all variables occurring in an expression w.

If $s_1 = [x_1/w_1 \ldots x_n/w_n]$ and $s_2 = [y_1/\lambda_1 \ldots y_m/\lambda_m]$ are substitutions, then by $s_1 \circ s_2$ we shall denote the composition of these substitutions, i.e.,

$$s_1 \circ s_2 = [y_1/\overline{s_1\lambda_1} \ldots y_m/\overline{s_1\lambda_m} x_{i_1}/w_{i_1} \ldots x_{i_q}/w_{i_q}]$$

where $x_{i_1}, \ldots, x_{i_q}$ are all the variables in the sequence $x_1, \ldots, x_n$ different from any variable $y_1, \ldots, y_m$.

By $\models_{\mathscr{R}} \alpha$ we shall denote the validity of a formula α in a class of realizations $\mathscr{R}$. In particular, $\models \alpha$ will denote that α is valid in the class of all realizations, i.e. that α is a tautology.

Let $\mathscr{R}$ be a class of realizations. By $\leqslant_{\mathscr{R}}$, $\leqslant$, $=_{\mathscr{R}}$ and $=$ we shall understand the following binary relations in F:

$$\begin{aligned} \alpha \leqslant_{\mathscr{R}} \beta \quad &\text{iff} \quad \models_{\mathscr{R}} (\alpha \Rightarrow \beta), \\ \alpha \leqslant \beta \quad &\text{iff} \quad \models (\alpha \Rightarrow \beta), \\ \alpha =_{\mathscr{R}} \beta \quad &\text{iff} \quad \models_{\mathscr{R}} (\alpha \Leftrightarrow \beta), \\ \alpha = \beta \quad &\text{iff} \quad \models (\alpha \Leftrightarrow \beta). \end{aligned}$$

Observe that the relations $\leqslant_{\mathscr{R}}$, $\leqslant$ are quasi-orderings in F and $=_{\mathscr{R}}$, $=$ are equivalence relations in F.

Let $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ be sequences of different variables. We shall say that $\mathbf{x}$ and $\mathbf{y}$ are *compatible* provided, for each $q = 1, \ldots, n$, x_q belongs to V_i iff y_q does and x_q belongs to V_0 iff y_q does.

Let $x_{i_1}, \ldots, x_{i_k}$ be all different individual variables in $\mathbf{x}$ and let $x_{j_1}, \ldots, x_{j_s}$ be all different propositional variables in $\mathbf{x}$. For compatible sequences $\mathbf{x}$ and $\mathbf{y}$ we shall use the following abbreviations:

$\mathbf{x} = \mathbf{y}$ for the formula

$$(x_{i_1} = y_{i_1} \wedge \ldots \wedge x_{i_k} = y_{i_k} \wedge x_{j_1} \Leftrightarrow y_{j_1} \wedge \ldots \wedge x_{j_s} \Leftrightarrow y_{j_s});$$

$\exists \mathbf{x} \alpha$ for the formula

$$\exists x_{i_1}, \ldots, x_{i_k} \bigvee_{\varepsilon = (\varepsilon_1, \ldots, \varepsilon_s) \in \{0,1\}^s} [x_{j_1}/\varepsilon_1 \ldots x_{j_s}/\varepsilon_s]\alpha;$$

and $\forall \mathbf{x} \alpha$ for the formula

$$\forall x_{i_1}, \ldots, x_{i_k} \bigwedge_{\varepsilon = (\varepsilon_1, \ldots, \varepsilon_s) \in \{0,1\}^s} [x_{j_1}/\varepsilon_1 \ldots x_{j_s}/\varepsilon_s]\alpha.$$

We shall say that $\mathbf{y}$ is a *copy* of $\mathbf{x}$ provided $\mathbf{x}$ and $\mathbf{y}$ are compatible and disjoint.

For example, the expression $\exists(x, y, p) * [p \Rightarrow (x, y, p) = (z, t, q)[y/\varphi(y)]]\, \varrho(x, y, z, t)$ is an abbreviation of the following formula

$$\exists x \exists y \big([p/\mathbf{0}] * [p \Rightarrow (x = z \wedge y = t \wedge p \Leftrightarrow q)[y/\varphi(y)]]\, \varrho(x, y, z, t) \vee [p/\mathbf{1}] * [p \Rightarrow (x = z \wedge y = t \wedge p \Leftrightarrow q)[y/\varphi(y)]]\, \varrho(x, y, z, t).$$

The following lemma shows that the admitting of the universal iteration quantifier is not necessary and every formula in F can be transformed to an equivalent one without the quantifier $\bigcap$.

LEMMA 2.1. *Let K be a program and let α be a formula. Then*

$$\bigcap K\alpha \equiv K\mathbf{1} \wedge \forall \mathbf{y} \big(*[\mathbf{x} \neq \mathbf{y} K]\mathbf{1} \Rightarrow (\alpha(\mathbf{y}) \wedge K(\mathbf{y})\mathbf{1})\big),$$

where

(1) $\mathbf{x} = (x_1, \ldots, x_n)$ *is the sequence of all different variables occurring in* $K\alpha$,

(2) $\mathbf{y} = (y_1, \ldots, y_n)$ *is a copy of* $\mathbf{x}$.

Proof: Let R be a realization and let v be a valuation. In virtue of the definition of a realization, the following facts are consecutively equivalent to one another:

(i) $\left(\bigcap K\alpha\right)_R(v) = V$;

(ii) for all n in N, $(K^n\alpha)_R(v) = V$;

(iii) $(K\mathbf{1})_R(v) = V$ and for every n in N

$$\left(\forall \mathbf{y}\left(K^n(\mathbf{x} = \mathbf{y}) \Rightarrow \left(\alpha(\mathbf{y}) \wedge K(\mathbf{y})\mathbf{1}\right)\right)\right)_R(v) = V;$$

(iv) $(K\mathbf{1})_R(v) = V$ and for every n in N

$$\left(\forall \mathbf{y}\left(\left(K^n(\mathbf{x} = \mathbf{y}) \wedge \bigwedge_{j=0}^{n-1} K^j(\mathbf{x} \neq \mathbf{y})\right) \Rightarrow \left(\alpha(\mathbf{y}) \wedge K(\mathbf{y})\mathbf{1}\right)\right)\right)_R(v) = V;$$

(v) $\left(K\mathbf{1} \wedge \forall \mathbf{y}\left(*[\mathbf{x} \neq \mathbf{y}\ K]\mathbf{1} \Rightarrow \left(\alpha(\mathbf{y}) \wedge K(\mathbf{y})\mathbf{1}\right)\right)\right)_R(v) = V$.

So we have proved Lemma 2.1. ■

It is convenient to treat the connectives $\forall$, $\bigcap$, $\wedge$ and $\Rightarrow$ as secondary, introduced as abbreviation signs of the language. Evidently they are defined by means of the remaining, "primitive" connectives. Namely,

$$\begin{aligned} \alpha \wedge \beta \quad &\text{as} \quad \neg(\neg\alpha \vee \neg\beta), \\ \alpha \Rightarrow \beta \quad &\text{as} \quad \neg\alpha \vee \beta, \\ \forall x\alpha \quad &\text{as} \quad \neg(\exists x(\neg\alpha)), \\ \bigcap K\alpha \quad &\text{as in Lemma 2.1.} \end{aligned}$$

3. Theorem on the isomorphism of realizations

Like in the first order predicate calculus [23] and in the algorithmic logic without classical quantifiers [22], the theorem on the isomorphism also holds in the extended algorithmic logic.

THEOREM 3.1 (*on isomorphism of realizations*). *Let J and J' be non-empty sets. Let R and R' be realizations in J and J', respectively, and let f be a mapping satisfying the following conditions*:

(1) $$f\colon J \xrightarrow[\text{onto}]{} J';$$

(2) *for every m-argument functor φ and every $j_1, \ldots, j_m$ in J,*

$$f\left(\varphi_R(j_1, \ldots, j_m)\right) = \varphi_{R'}\left(f(j_1), \ldots, f(j_m)\right);$$

(3) *for every m-argument predicate ϱ and every $j_1, \ldots, j_m$ in J,*

$$\varrho_R(j_1, \ldots, j_m) = \varrho_{R'}\left(f(j_1), \ldots, f(j_m)\right).$$

Let v be a valuation in the sets J and B_0. By fv we shall denote the following valuation in the sets J' and B_0:

$$(fv)(x) = \begin{cases} f(v(x)) & \text{if } x \text{ is an individual variable,} \\ v(x) & \text{if } x \text{ is a propositional variable;} \end{cases}$$

Then for any formula α and any valuation v in the sets J and B_0 the following equality holds:

$$\alpha_R(v) = \alpha_{R'}(fv). \tag{4}$$

Proof: It is easy to show (see [23], p. 219 and p. 233, and [22], pp. 45–50) that for every term τ, every open formula γ, program K and valuation v in the sets J and B_0 the following equalities hold:

(i) $f(\tau_R(v)) = \tau_{R'}(fv)$,

(ii) $\gamma_R(v) = \gamma_{R'}(fv)$,

(iii) $f(K_R(v)) = K_{R'}(fv)$.

Equality (iii) should be read as follows: if one side of the equation is defined, then so is the other and they are equal.

The proof of equality (4) proceeds by induction on the length of formulas. We shall consider only three cases of induction steps. Let (4) hold for a formula α and let x be an individual variable. By definition,

$$(\exists x\alpha)_{R'}(fv) = \text{l.u.b.}_{j' \in J'} \alpha_{R'}((fv)^x_{j'}).$$

Since f maps the set J onto the set J', we obtain

$$\text{l.u.b.}_{j' \in J'} \alpha_{R'}((fv)^x_{j'}) = \text{l.u.b.}_{j \in J} \alpha_{R'}((fv)^x_{f(j)}) = \text{l.u.b.}_{j \in J} \alpha_{R'}(fv^x_j).$$

Hence by the induction hypothesis (4)

$$\text{l.u.b.}_{j \in J} \alpha_{R'}(fv^x_j) = \text{l.u.b.}_{j \in J} \alpha_R(v^x_j) = (\exists x\alpha)_R(v).$$

So equality (4) holds also for the formula $\exists x\alpha$.

Let (4) hold for a formula α and let K be a program. Applying the induction hypothesis and equality (iii) we get consecutively

$$(K\alpha)_R(v) = \begin{cases} \alpha_R(K_R(v)) & \text{if } v \in D_{K_R}, \\ \Lambda & \text{otherwise} \end{cases} = \begin{cases} \alpha_{R'}(fK_R(v)) & \text{if } v \in D_{K_R}, \\ \Lambda & \text{otherwise} \end{cases}$$

$$= \begin{cases} \alpha_{R'}(K_{R'}(fv)) & \text{if } fv \in D_{K_{R'}}, \\ \Lambda & \text{otherwise} \end{cases} = (K\alpha)_{R'}(fv).$$

So $(K\alpha)_R(v) = (K\alpha)_{R'}(fv)$.

By usual induction we obtain that for every natural n in N

$$(K^n\alpha)_R(v) = (K^n\alpha)_{R'}(fv).$$

Hence

$$\big(\bigcup Ka\big)_R(v) = \underset{n\in N}{\text{l.u.b.}}\ (K^n a)_R(v) = \underset{n\in N}{\text{l.u.b.}}\ (K^n a)_{R'}(fv) = \big(\bigcup Ka\big)_{R'}(fv). \ \blacksquare$$

4. Algorithmic formalized theories

In this section we define an algorithmic formalized theory based on the set F of formulas. Next, we prove a theorem on the Lindenbaum algebra for an algorithmic theory (4.2) and a theorem on extensionality (4.3), very often applicable in the sequel.

By Ax we denote the set of logical axioms consisting of all formulas in F of the form (T_1)–(T_{12}), (e_1)–(e_5) (these schemata are taken from [23]) and (A0)–(A11), where

(T_1) $\big((\alpha \Rightarrow \beta) \Rightarrow \big((\beta \Rightarrow \gamma) \Rightarrow (\alpha \Rightarrow \gamma)\big)\big)$,

(T_2) $\big(\alpha \Rightarrow (\alpha \vee \beta)\big)$,

(T_3) $\big(\beta \Rightarrow (\alpha \vee \beta)\big)$,

(T_4) $\big((\alpha \Rightarrow \gamma) \Rightarrow \big((\beta \Rightarrow \gamma) \Rightarrow \big((\alpha \vee \beta) \Rightarrow \gamma\big)\big)\big)$,

(T_5) $\big((\alpha \wedge \beta) \Rightarrow \alpha\big)$,

(T_6) $\big((\alpha \wedge \beta) \Rightarrow \beta\big)$,

(T_7) $\big((\gamma \Rightarrow \alpha) \Rightarrow \big((\gamma \Rightarrow \beta) \Rightarrow \big(\gamma \Rightarrow (\alpha \wedge \beta)\big)\big)\big)$,

(T_8) $\big(\big(\alpha \Rightarrow (\beta \Rightarrow \gamma)\big) \Rightarrow \big((\alpha \wedge \beta) \Rightarrow \gamma\big)\big)$,

(T_9) $\big(\big((\alpha \wedge \beta) \Rightarrow \gamma\big) \Rightarrow \big(\alpha \Rightarrow (\beta \Rightarrow \gamma)\big)\big)$,

(T_{10}) $\big((\alpha \wedge \neg\alpha) \Rightarrow \beta\big)$,

(T_{11}) $\big(\big(\alpha \Rightarrow (\alpha \wedge \neg\alpha)\big) \Rightarrow \neg\alpha\big)$,

(T_{12}) $(\alpha \vee \neg\alpha)$;

(e_1) $(x = x)$,

(e_2) $\big((x = y) \Rightarrow (y = x)\big)$,

(e_3) $\big((x = y) \Rightarrow \big((y = z) \Rightarrow (x = z)\big)\big)$,

(e_4) *for every natural number m and every m-argument functor φ*

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

(e_5) *for every natural number m and every predicate ϱ*

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

(A0) $(\mathbf{1} \wedge \neg\mathbf{0})$,

(A1) $(s\gamma \Leftrightarrow s\gamma)$ *where γ is an open formula,*

(A2) $\big(s(\exists x\alpha) \Leftrightarrow \big(\exists y\,\big(s([x/y]\alpha)\big)\big)\big)$ *where y is an individual variable not occurring in $s\alpha$,*

(A3) $\big((s_1 \circ s_2)\alpha \Leftrightarrow \big(s_1(s_2\alpha)\big)\big)$,

(A4) $\big(K(\alpha\vee\beta)\Leftrightarrow(K\alpha\vee K\beta)\big)$,

(A5) $\big(K(\alpha\wedge\beta)\Leftrightarrow(K\alpha\wedge K\beta)\big)$,

(A6) $(s\neg\alpha\Leftrightarrow\neg s\alpha)$,

(A7) $\big(\circ[K\ M]\alpha\Leftrightarrow\big(K(M\alpha)\big)\big)$,

(A8) $\big(\preceq[\gamma K\ M]\alpha\Leftrightarrow\big((\gamma\wedge K\alpha)\vee(\neg\gamma\wedge M\alpha)\big)\big)$ *where* γ *is an open formula,*

(A9) $\big(*[\gamma K]\alpha\Leftrightarrow\bigcup\preceq[\gamma K[\ ]](\neg\gamma\wedge\alpha)\big)$ *where* γ *is an open formula,*

(A10) $\big(P\bigcup K\alpha\Leftrightarrow\big(P\alpha\vee P\bigcup K(K\alpha)\big)\big)$,

(A11) $([x/\tau]\alpha\Rightarrow\exists x\alpha)$.

We admit the following rules of inference:

(MP) $\dfrac{\alpha,\ \alpha\Rightarrow\beta}{\beta}$,

(K) $\dfrac{\alpha\Rightarrow\beta}{K\alpha\Rightarrow K\beta}$,

(Ex) $\dfrac{[x/y]\alpha\Rightarrow\beta}{\exists x\alpha\Rightarrow\beta}$ *where* y *is an individual variable occurring neither in* α *nor in* β,

(It) $\dfrac{\{PK^i\alpha\Rightarrow\beta\}_{i\in N}}{P\bigcup K\alpha\Rightarrow\beta}$.

In the above schemata of axioms and in the rules of inference we assume that:

α,β,γ are formulas in F,

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

s,s_1,s_2 are substitutions;

K,M are programs in FS;

P is a program or the empty word and τ is a term.

Let A be a subset of F. By $C(A)$ we denote the set of all theorems deducible from the set $A\cup\mathrm{Ax}$ by means of the rules (MP), (K), (Ex) and (It). The system $\mathscr{T}=\{\mathscr{L},C,A\}$ will be called an *algorithmic theory* (compare with [23], p. 151).

In the sequel we shall use the following rule of inference reminding the classical substitution rule

(Sub) $\dfrac{\alpha}{s\alpha}$,

where α is a formula and s is a substitution. This rule is secondary with respect to the admitted axioms and rules of inference. Namely, if α is in $C(A)$, then the formula $(\mathbf{1}\Rightarrow\alpha)$ is in $C(A)$. Using the rule (K), we get that $(s\mathbf{1}\Rightarrow s\alpha)$ is a theorem in $C(A)$. Now, according to the schemata (A0) and (A1), $s\mathbf{1}$ is also in $C(A)$; and hence $s\alpha$ belongs to $C(A)$.

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

R is called *ordinary* if the sign $=$ of equality is realized as the identity relation.

If an ordinary realization R is a model for a theory $\mathscr{T}$, then we say that R is an ordinary model for $\mathscr{T}$.

LEMMA 4.1. *Let $\mathscr{T} = \{\mathscr{L}, C, A\}$ be an algorithmic theory. Theorems in $C(A)$ are valid in every ordinary model for the theory $\mathscr{T}$.*

Proof: Let R be an ordinary model for the theory $\mathscr{T}$ in the sets J and B_0. We shall prove that axioms (A2), (A3), (A11) are valid in R and the rule (Ex) is consistent in R. The remaining axioms and rules of inference have been examined in [22] and [23].

Let v be a valuation.

(A2): Let us assume that s is a substitution, α is a formula, x is an individual variable and y is an individual variable not occurring in the formula $s\alpha$. Observe that for any variable z and any j in J,

$$s_R(v)_j^x(z) = \begin{cases} s_R(v)(z) & \text{if} \quad z \neq x, \\ j & \text{if} \quad z = x, \end{cases}$$

and

$$[x/y]_R\big(s_R(v_j^y)\big)(z) = \begin{cases} s_R(v)(z) & \text{if} \quad z \neq x \text{ and } z \neq y, \\ j & \text{if} \quad z = x \text{ or } z \neq y. \end{cases}$$

Since the variable y does not occur in the formula α, then

$$\alpha_R\big(s_R(v)_j^x\big) = \alpha_R\Big([x/y]_R\big(s_R(v_j^y)\big)\Big).$$

Hence we obtain

$$\begin{aligned}(s\exists x\alpha)_R(v) &= \text{l.u.b.}_{j\in J}\, \alpha_R\big(s_R(v)_j^x\big) = \text{l.u.b.}_{j\in J}\, \alpha_R\Big([x/y]_R\big(s_R(v_j^y)\big)\Big) \\ &= \text{l.u.b.}_{j\in J}\, (s[x/y]\alpha)_R(v_j^y) = \big(\exists y(s[x/y]\alpha)\big)_R(v).\end{aligned}$$

(A3): Let $s_1 = [x_1/w_1 \dots x_n/w_n]$ and $s_2 = [y_1/\lambda_1 \dots y_m/\lambda_m]$ be arbitrary substitutions. By definition, $s_1 \circ s_2 = [y_1/\overline{s_1\lambda_1} \dots y_m/\overline{s_1\lambda_m}\, x_{i_1}/w_{i_1} \dots x_{i_q}/w_{i_q}]$ where $x_{i_1}, \dots, x_{i_q}$ are all the variables in the sequence $x_1, \dots, x_n$ different from $y_1, \dots, y_m$. It has been proved in [22], p. 25, that for any open formula or term w, $(sw)_R(v) = (\overline{sw})_R(v)$. Using this equality, we obtain consecutively that for every variable x,

$$\begin{aligned}s_{2R}\big(s_{1R}(v)\big)(x) &= \begin{cases} \lambda_{iR}\big(s_{1R}(v)\big) & \text{if} \quad x = y_i \text{ for some } i = 1, \dots, m \\ s_{1R}(v)(x) & \text{otherwise,} \end{cases} \\ &= \begin{cases} (\overline{s_1\lambda_i})_R(v) & \text{if} \quad x = y_i \text{ for some } i = 1, \dots, m, \\ w_{i_kR}(v) & \text{if} \quad x = x_{i_k} \text{ for some } k = 1, \dots, q, \\ v(x) & \text{otherwise,} \end{cases} \\ &= \big((s_1 \circ s_2)_R(v)\big)(x).\end{aligned}$$

So, for all formulas α, $(s_1(s_2\alpha))_R(v) = \alpha_R\big(s_{2R}(s_{1R}(v))\big) = \alpha_R\big((s_1 \circ s_2)_R(v)\big) = \big((s_1 \circ s_2)\alpha\big)_R(v)$.

(A11): Let α be a formula and let x be an individual variable. By the definition of a realization, we infer that for every term τ,

$$([x/\tau]\alpha)_R(v) = \alpha_R\big(v^x_{\tau_R(v)}\big) \leqslant_R \text{l.u.b.}_{j\in J}\, \alpha_R(v^x_j) = (\exists x\alpha)_R(v).$$

(Ex): Let α, β be formulas in F and let x be an individual variable. Let y be an individual variable occurring neither in α nor in β. On account of this, for every valuation v and every j in J we have

$$\alpha_R(v^x_j) = ([x/y]\alpha)_R(v^y_j) \quad \text{and} \quad \beta_R(v^y_j) = \beta_R(v).$$

Hence, if for every valuation v', $([x/y])\alpha_R(v') \leqslant \beta_R(v')$, then, writing $v' = v^y_j$, we infer that

$$(\exists x\alpha)_R(v) = \text{l.u.b.}_{j\in J}\, \alpha_R(v^x_j) = \text{l.u.b.}_{j\in J}\, ([x/y]\alpha)_R(v^y_j) \leqslant \text{l.u.b.}_{j\in J}\, \beta_R(v^y_j) = \beta_R(v).$$

So we have proved that if $([x/y]\alpha \Rightarrow \beta)$ is valid in R then also $(\exists x\alpha \Rightarrow \beta)$ is valid in R.

Applying the induction on the length of a proof we get that every theorem in $C(A)$ is valid in R and this completes the proof of Lemma 4.1. ■

Let $\mathscr{T} = \{\mathscr{L}, C, A\}$ be an algorithmic theory. By the *equivalence relation of the theory* $\mathscr{T}$ we mean the following binary relation on F: $\alpha \approx \beta$ iff $(\alpha \Leftrightarrow \beta)$ is in $C(A)$. Let $\|\alpha\|$ denote the equivalence class containing the formula α. Let $\dashv$ denote the following binary relation on $F/\!\approx$: $\|\alpha\| \dashv \|\beta\|$ iff $(\alpha \Rightarrow \beta)$ is in $C(A)$. The following lemma is closely related to Lemma 1.1 in [23], p. 280.

LEMMA 4.2. *The Lindenbaum algebra* $\langle F/\!\approx, \dashv\rangle$ *of the theory* $\mathscr{T}$ *is a Boolean algebra. For every formulas* α, β *and every program* K *the following facts hold*:

(1) $\|\alpha\| = \mathsf{V}$ *iff* α *is in* $C(A)$,

(2) $\|\neg\alpha\| \neq \Lambda$ *iff* $\neg\alpha$ *is not in* $C(A)$,

(3) *if* $\|\alpha\| = \|\beta\|$, *then* $\|K\alpha\| = \|K\beta\|$,

(4) $\|\exists x\alpha\| = \text{l.u.b.}_{\tau\in T}\, \|[x/\tau]\alpha\|$ *for every individual variable* x,

(5) $\|M \bigcup K\alpha\| = \text{l.u.b.}_{l\in N}\, \|MK^l\alpha\|$ *where* M *is a program or the empty word.*

Proof: The proof of assertions (1) and (2) is the same as in [23], §10, ch. VI, and therefore is omitted here. The assertion (3) results from the rule (K). To prove (4), let us observe that in virtue of the scheme (A11) of the axioms, the class $\|\exists x\alpha\|$ is greater than any class $\|[x/\tau]\alpha\|$ for τ in T. Now, take an upper bound $\|\beta\|$ for the set $\{\|[x/\tau]\alpha\|\}_{\tau\in T}$. In particular, for

any variable y occurring neither in α nor in β, $\|[x/y]\alpha\| \dashv \|\beta\|$, i.e., $([x/y]\alpha \Rightarrow \beta)$ belongs to $C(A)$. Applying the rule (Ex) we get that $(\exists x\alpha \Rightarrow \beta)$ is in $C(A)$. So $\|\exists x\alpha\| \dashv \|\beta\|$ and therefore the class $\|\exists x\alpha\|$ is the least upper bound for $\{\|[x/\tau]\alpha\|\}_{\tau\in T}$. The proof of the assertion (5) applies the scheme (A9) of the axioms and the rule (It) and was carried out in [22], p. 98. ■

Let $\alpha, \beta, \gamma, \delta$ be formulas. By $\mathrm{Replace}^{\gamma}_{\delta}$ we shall denote the least binary relation satisfying the following conditions:

(1) $\mathrm{Replace}^{\gamma}_{\delta}(\alpha, \alpha)$ holds.

(2) If

(21) β has the form $\beta_1\gamma\beta_2$,

(22) the occurrence of the formula γ in β indicated above, is not a part of any program in β,

(23) $\mathrm{Replace}^{\gamma}_{\delta}(\alpha, \beta)$ holds,

then $\mathrm{Replace}^{\gamma}_{\delta}(\alpha, \beta_1\delta\beta_2)$ also holds.

Lemma 4.2 implies the following fact frequently used in the sequel.

LEMMA 4.3 (*Theorem on extensionality*). *For every algorithmic theory* $\mathscr{T} = \{\mathscr{L}, C, A\}$ *and every formulas* α, β, γ, *and* δ, *if* $\gamma \approx \delta$ *and* $\mathrm{Replace}^{\gamma}_{\delta}(\alpha, \beta)$ *holds, then* $\alpha \approx \beta$.

Proof: The proof proceeds by induction on the length of β. In virtue of Lemma 4.2 we have

(i) if $\alpha \approx \beta$ and $\alpha' \approx \beta'$ then $\alpha\vee\alpha' \approx \beta\vee\beta'$ and $\neg\alpha \approx \neg\beta$;

(ii) if $\alpha \approx \beta$ then $K\alpha \approx K\beta$ for every program K;

(iii) if $\alpha \approx \beta$ and x is an individual variable, then by (ii), for every term τ, $[x/\tau]\alpha \approx [x/\tau]\beta$ and hence $\exists x\alpha \approx \exists x\beta$;

(iv) if $\alpha \approx \beta$ and K is a program, then in accordance with (ii), $K^l\alpha \approx K^l\beta$ for all l in N and hence $\bigcup K\alpha \approx \bigcup K\beta$. ■

5. The simplified form of a formula

In order to prove the completeness of the extended algorithmic logic we need the notion of the simplified form of a formula. Roughly speaking, this simplification consists in getting rid of programs as much as possible.

A formula α is said to be in the *simplified form* if it is either an open formula or it has one of the forms $(\alpha_1\vee\alpha_2)$, $\neg\alpha_1$, $(\exists x\alpha_1)$, $(s \bigcup K\alpha_1)$, where α_1 and α_2 are formulas in the simplified form, s is a substitution or the empty word, K is a program, and x is an individual variable.

LEMMA 5.1. *For every formula* α *there exists a formula* α' *in the simplified form such that* $(\alpha \Leftrightarrow \alpha')$ *is in* $C(\emptyset)$.

Proof: We shall define by induction an operation of reducing a formula α to its simplified form α'. This operation can be treated as an extension of the operation of the simultaneous replacement (see p. 5).

DEFINITION 5.2. We assume that s_1, s_2 are substitutions, s is a substitution or the empty word, α, β are formulas, K, M are programs, γ is an open formula and x is an individual variable.

If α is an open formula, then:

(c0) $(s_1\alpha)' : \overline{s_1}\,\overline{\alpha}$,

(c1) $\alpha' : \alpha$.

Moreover, we put:

(c2) $\big(s_1(s_2\alpha)\big)' : \big((s_1 \circ s_2)\alpha\big)'$,

(c3) $\big(s(\alpha \vee \beta)\big)' : \big((s\alpha)' \vee (s\beta)'\big)$ if $(\alpha \vee \beta)$ is not an open formula,

(c4) $(s\neg\alpha)' : \neg(s\alpha)'$ if $\neg\alpha$ is not an open formula,

(c5) $(\exists x\alpha)' : \exists x\alpha'$,

(c6) $(s_1\exists x\alpha)' : \exists y(s_1 \circ [x/y]\alpha)'$ where y is the first individual variable in the sequence V_i not appearing in $s_1\alpha$,

(c7) $(s \circ [KM]\alpha)' : (sKM\alpha)'$,

(c8) $(s\preceq[\gamma KM]\alpha)' : \big((s\gamma)' \wedge (sK\alpha)'\big) \vee \big((s\neg\gamma)' \wedge (sM\alpha)'\big)$,

(c9) $(s*[\gamma K]\alpha)' : s \bigcup \preceq[\gamma K[\ ]](\alpha' \wedge \neg\gamma)$,

(c10) $\big(s \bigcup K\alpha\big)' : s \bigcup K\alpha'$.

EXAMPLES. Let ϱ be a 2-argument predicate, φ a 1-argument functor and x, y individual variables. Let z be the first variable in the sequence V_i equal neither to x nor to y.

(1) $\big([x/y]\big(\exists x \bigcup [x/\varphi(x)]\varrho(x,y)\big)\big)'$

$= \exists z\big(([x/y] \circ [x/z]) \bigcup [x/\varphi(x)]\varrho(xy)\big)'$

$= \exists z\big([x/z] \bigcup [x/\varphi(x)]\varrho(x,y)\big)$,

(2) $\Big(\circ\big[[x/\varphi(y)]\preceq\big[\varrho(xy)[x/\varphi(x)][y/\varphi(y)]\big]\big]\exists x\varrho(x,y)\Big)'$

$= \big([x/\varphi(y)]\preceq\big[\varrho(x,y)[x/\varphi(x)][y/\varphi(y)]\big]\exists x\varrho(x,y)\big)'$

$= \Big(\big([x/\varphi(y)]\varrho(x,y)\big)' \wedge \big([x/\varphi(y)][x/\varphi(x)]\exists x\varrho(x,y)\big)'\Big) \vee$

$\vee \Big(\big([x/\varphi(y)]\neg\varrho(x,y)\big)' \wedge \big([x/\varphi(y)][y/\varphi(y)]\exists x\varrho(x,y)\big)'\Big)$

$= \Big(\varrho\big(\varphi(y)y\big) \wedge \big([x/\varphi\big(\varphi(y)\big)]\exists x\varrho(x,y)\big)'\Big) \vee \Big(\neg\varrho\big(\varphi(y)y\big) \wedge \big([y/\varphi(y)]\exists x\varrho(x,y)\big)'\Big)$

$= \Big(\Big(\varrho\big(\varphi(y)y\big) \wedge \exists z\big([x/\varphi\big(\varphi(y)\big)] \circ [x/z]\varrho(x,y)\big)'\Big) \vee$

$\vee \Big(\neg\varrho\big(\varphi(y)y\big) \wedge \exists z\big([y/\varphi(y)] \circ [x/z]\varrho(x,y)\big)'\Big)\Big)$

$= \Big(\big(\varrho\big(\varphi(y)y\big) \wedge \exists z\varrho(z,y)\big) \vee \big(\neg\varrho\big(\varphi(y)y\big) \wedge \exists z\varrho\big(z\varphi(y)\big)\big)\Big)$.

Let us observe that every formula in F can be univocally represented in the form $s\alpha$ where

(1) s is the empty word or a substitution,

(2) if s is the empty word then α does not begin with any substitution.

This fact guarantees that Definition 5.2 covers all possible forms of formulas. Moreover, in every particular inductive step either the total

number of substitutions and connectives decreases or the algorithm stops. So the algorithm 5.2 defines a total function.

We have to prove that for every formula α

(i) $(\alpha \Leftrightarrow \alpha')$ belongs to $C(\emptyset)$.

The proof proceeds by induction on the number n of direct steps of the algorithm 5.2. If $n = 0$, this is evident.

Suppose that (i) holds for some n steps and let α be a formula which requires $n+1$ steps of the algorithm. On account of the choice of the logical axioms (A1)–(A9), pp. 104–105 and of Theorem 4.3 on extensionality, our induction hypothesis implies that (i) holds also for α.

To illustrate this way of reasoning we shall carry out the induction step in the case (c8).

Let α be of the form $s\trianglelefteq[\gamma K M]\beta$. First assume that s is a substitution. In virtue of schema (A8) of axioms and of Theorem 4.3,

$$(1) \qquad s\trianglelefteq[\gamma K M]\beta \Leftrightarrow s\big((\gamma \wedge K\beta)\vee(\neg\gamma \wedge M\beta)\big) \in C(\emptyset).$$

According to schemata (A4) and (A5) and to Theorem 4.3, it follows that

$$(2) \qquad \big(s\trianglelefteq[\gamma K M]\beta \Leftrightarrow \big((s\gamma \wedge sK\beta)\vee(s\neg\gamma \wedge sM\beta)\big)\big) \in C(\emptyset).$$

By (c0) and schema (A1) we obtain that

$$(3) \qquad \big(s\gamma \Leftrightarrow (s\gamma)'\big) \quad \text{and} \quad \big(s\neg\gamma \Leftrightarrow (s\neg\gamma)'\big) \text{ belong to } C(\emptyset)$$

and, according to the induction hypothesis,

$$(4) \qquad \big(sK\beta \Leftrightarrow (sK\beta)'\big) \text{ and } \big(sM\beta \Leftrightarrow (sM\beta)'\big) \text{ belong to } C(\emptyset).$$

Applying the theorem on extensionality once more, we get that

$$(5) \quad \Big(\big((s\gamma \wedge sK\beta)\vee(s\neg\gamma \wedge sM\beta)\big) \Leftrightarrow \big((s\gamma)' \wedge (sK\beta)'\big)\vee\big((s\neg\gamma)' \wedge (sM\beta)'\big)\Big) \in C(\emptyset).$$

Hence, by (2) and by (c8), it follows finally that $(\alpha \Leftrightarrow \alpha')$ is in $C(\emptyset)$.

Now let s be the empty word. Observe that:
the assertion (1) follows from schema (A8) without use of the rule K,
the sentences (1) and (2) are identical,
the point (c1) of Definition 5.2 implies the fact (3),
the continuation of the proof remains without change. ■

6. Completeness theorem

THEOREM 6.1 (*Completeness theorem*). *For every algorithmic theory* $\mathscr{T} = \{\mathscr{L}, C, A\}$ *the following conditions are equivalent*:

(1) α *is a theorem in* $C(A)$,

(2) α *is valid in every ordinary model for the theory* $\mathscr{T}$.

Proof: The implication (1)→(2) is contained in Lemma 4.1. In order to show that assertion (2) implies (1), it is sufficient to prove the following lemma:

LEMMA 6.2. *For every formula α which is not a theorem in $C(A)$, there exists an ordinary model for $\mathscr{T}$, in which the formula $\neg\alpha$ is satisfiable.*

Proof: Let us assume that α does not belong to $C(A)$. In virtue of Lemma 4.2, assertion (2), $\|\neg\alpha\| \neq \Lambda$.

Let (Q) be the following set of infinite joins:

$$\text{(Q)} \qquad \begin{cases} \|s \bigcup K\beta\| = \text{l.u.b.}_{l\in N} \|sK^l\beta\|, \\ \|\exists x\beta\| = \text{l.u.b.}_{\tau\in T} \|[x/\tau]\beta\| \end{cases}$$

for all formulas β, programs K, substitutions, and the empty word s, individual variables x and terms τ.

Let $\approx$ be the equivalence relation of the theory $\mathscr{T}$. In virtue of lemma of Rasiowa and Sikorski ([23], p. 87), there exists a Q-homomorphism h from $F/\approx$ onto $B_0 = \{\Lambda, \mathsf{V}\}$ such that $h\|\neg\alpha\| = \mathsf{V}$. We recall that the canonical realization determined by the Q-homomorphism h is the realization R^0 in the set T of terms defined as follows: for every n-argument functor φ and terms $\tau_1, \ldots, \tau_n$

$$\varphi_{R^0}(\tau_1, \ldots, \tau_n) = \varphi(\tau_1, \ldots, \tau_n)$$

and for every n-argument predicate ϱ and terms $\tau_1, \ldots, \tau_n$

$$\varrho_{R^0}(\tau_1, \ldots, \tau_n) = h\|\varrho(\tau_1, \ldots, \tau_n)\|.$$

Now we need the following auxilliary lemma:

LEMMA 6.3. *For every term τ, formula β, and valuation v in T and B_0 the following equalities hold*:

$$\text{(6.3.1)} \qquad \tau_{R^0}(v) = \widehat{\hat{v}\tau},$$

$$\text{(6.3.2)} \qquad \beta_{R^0}(v) = h\|\hat{v}\beta\|,$$

where $\hat{v} = [x_1/w_1, \ldots, x_n/w_n]$ is a substitution satisfying the following conditions:

(1) *for each $i = 1, \ldots, n$*:

$$w_i = \begin{cases} v(x_i) & \text{if } x_i \text{ is an individual variable,} \\ \mathbf{1} & \text{if } x_i \text{ is a propositional variable and } v(x_i) = \mathsf{V}, \\ \mathbf{0} & \text{if } x_i \text{ is a propositional variable and } v(x_i) = \Lambda; \end{cases}$$

(2) *all the variables occurring in τ and β are included in the sequence $x_1, \ldots, x_n$.*

Proof of 6.3: Equality (6.3.1) is proved in [23], p. 235. The proof of (6.3.2) is divided into the proofs of the following three sublemmas:

(I) $\beta_{R^0}(v) = (\hat{v}\beta)_{R^0}(i)$,

(II) $\beta_{R^0}(i) = h\|\beta\|$ under the assumption that β is in the simplified form,

(III) $\beta_{R^0}(i) = h\|\beta\|$ for an arbitrary formula β, where i is the following valuation:

$$i(x) = \begin{cases} x & \text{if} \quad x \text{ is an individual variable,} \\ h\|x\| & \text{if} \quad x \text{ is a propositional variable.} \end{cases}$$

Equation (I) results from the observation that $\hat{v}_{R^0}(x_j) = v(x_j)$ for all variables x_j appearing in the formula β.

In virtue of Lemma 5.1, equation (II) implies (III). In fact, every formula β is equivalent to its simplified form β' in every algorithmic theory. Hence $\|\beta\| = \|\beta'\|$ and, according to Lemma 4.1, $\beta_{R^0}(i) = \beta'_{R^0}(i)$.

Statement (II) will be proved by transfinite induction on the ordinal number $\chi(\beta)$ defined for a formula β as follows: $\chi(\beta) = \sum_{n=0}^{\mathrm{ms}(\beta)} \omega^{n+1}\,\mathrm{np}_n(\beta) + \mathrm{nc}(\beta)$ where $\mathrm{nc}(\beta)$ is the number of occurrences of propositional connectives and the existential quantifier in β, $\mathrm{np}_n(\beta)$ is the number of occurrences of programs K such that K contains exactly n stars and $\bigcup K$ is a subexpression of β, $\mathrm{ms}(\beta)$ is either the greatest natural number such that $\mathrm{np}_n(\beta) \neq 0$ or, if such n does not exist, then $\mathrm{ms}(\beta) = -1$.

Let $\chi(\beta) = 0$.

If β: p is a propositional variable, then $p_{R^0}(i) = i(p) = h\|p\|$.

If β: $\mathbf{1}$ then $\mathbf{1}_{R^0}(i) = \vee = h\|\mathbf{1}\|$.

If β: $\mathbf{0}$ then $\mathbf{0}_{R^0}(i) = \wedge = h\|\mathbf{0}\|$.

If β: $\varrho(\tau_1, \ldots, \tau_n)$, where ϱ is an n-argument predicate and $\tau_1, \ldots, \tau_n$ are terms, then applying (6.3.1) we get that

$$\begin{aligned} \varrho(\tau_1, \ldots, \tau_n)_{R^0}(i) &= \varrho_{R^0}\big(\tau_{1R^0}(i), \ldots, \tau_{nR^0}(i)\big) = \varrho_{R^0}(\tau_1, \ldots, \tau_n) \\ &= h\|\varrho(\tau_1, \ldots, \tau_n)\|. \end{aligned}$$

Now suppose that equation (II) is valid for all formulas δ such that $\chi(\delta) < \chi(\beta)$. Using the induction hypothesis and the fact that h is a homomorphism, we obtain in the cases β: $\beta_1 \vee \beta_2$ and β: $\neg\beta_3$ that

$$(\beta_1 \vee \beta_2)_{R^0}(i) = \beta_{1R^0}(i) \vee \beta_{2R^0}(i) = h\|\beta_1\| \vee h\|\beta_2\| = h\|\beta_1 \vee \beta_2\|$$

and

$$(\neg\beta_3)_{R^0}(i) = \neg\beta_{3R^0}(i) = \neg h\|\beta_3\| = h\|\neg\beta_3\|.$$

The next case we consider is that where β is of the form $\exists x\beta_1$. Observe that for every term τ, $\chi\big(([x/\tau]\beta_1)'\big) < \chi(\exists x\beta_1)$. Now we can apply the induction hypothesis, getting for every term τ that

$$([x/\tau]\beta_1)'_{R^0}(i) = h\|([x/\tau]\beta_1)'\|.$$

On account of Lemma 5.1 this implies that for every term τ

$$([x/\tau]\beta_1)_{R^0}(i) = h\|[x/\tau]\beta_1\|$$

and hence

$$\text{l.u.b.}_{\tau\in T}\,([x/\tau]\beta_1)_{R^0}(i) = \text{l.u.b.}_{\tau\in T}\, h\|[x/\tau]\beta_1\|.$$

But $(\exists x\beta_1)_{R^0}(i) = \text{l.u.b.}_{\tau\in T'}\,([x/\tau]\beta_1)_{R^0}(i)$ and so, in virtue of Lemma 4.2, assertion (4) and of the fact that h is a Q-homomorphism, we get $h\|\exists x\beta_1\| = h\,\text{l.u.b.}_{\tau\in T}\,\|[x/\tau]\beta_1\| = \text{l.u.b.}_{\tau\in T}\, h\|[x/\tau]\beta_1\|$. Thus, $(\exists x\beta_1)_{R^0}(i) = h\|\exists x\beta_1\|$.

The induction step in the case β: $s \bigcup K\beta_1$ is similar. First, we must observe that for all natural numbers l $\chi\big((sK^l\beta_1)'\big) < \chi\big(s \bigcup K\beta_1\big)$. Next we apply the induction hypothesis and Lemma 5.1, getting $(sK^l\beta_1)_{R^0}(i) = h\|sK^l\beta_1\|$ for all natural l. Using these equalities, Lemma 4.2, assertion (5), and the fact that h is a Q-homomorphism, we obtain consecutively

$$(s\bigcup K\beta_1)_{R^0}(i) = \text{l.u.b.}_{l\in N}\,(sK^l\beta_1)_{R^0}(i) = \text{l.u.b.}_{l\in N}\,\|sK^l\beta_1\|$$
$$= h(\text{l.u.b.}_{l\in N}\,\|sK^l\beta_1\|) = h\|s\bigcup K\beta_1\|. \blacksquare$$

Now we return to the proof of Lemma 6.2. The canonical realization R^0 is a model for the theory $\mathscr{T}$. In fact, let β be a theorem in $C(A)$ and let v be a valuation in T and B_0. Let $\hat{v}$ be the substitution defined for β and v in Lemma 6.3. By the rule (Sub) on p. 105, the formula $\hat{v}\beta$ is in $C(A)$. In virtue of assertion (1) of Lemma 4.2, $\|\hat{v}\beta\| = V$. Hence by (6.3.2) we obtain that $\beta_{R^0}(\tilde{v}) = h\|\hat{v}\beta\| = V$. So, if β is in $C(A)$, then β is valid in R^0. Moreover, observe that $(\neg\alpha)_{R^0}(i) = h\|\neg\alpha\| = V$.

Until this point we have proved that R^0 is a model for the theory $\mathscr{T}$ and $(\neg\alpha)_{R^0}(i) = V$. This model need not be an ordinary one. The continuation of the proof of Lemma 6.2 is similar to the considerations on pp. 290–292 in [23]. Namely, since the formulas of the form (e_1)–(e_6), p. 104, are theorems, then the relation $=_R$ is a congruence of R^0. Dividing R^0 by this congruence we obtain an ordinary realization R'. Since the realizations R^0 and R' satisfy the assumptions of Theorem 3.1 on the isomorphism, then applying this theorem we get that R' is a model for $\mathscr{T}$ and $(\neg\alpha)_{R'}(i') = V$, where i' is the image of the valuation i under the canonical homomorphism between R^0 and R'. ■

Like in the classical logic, the completeness theorem implies the following corollaries:

(i) *A formula α is a theorem in $C(\emptyset)$ iff α is valid in all ordinary realizations.*

(ii) *An algorithmic theory* $\mathscr{T} = \{\mathscr{L}, C, A\}$ *is consistent iff* $\mathscr{T}$ *has an ordinary model iff* $\mathscr{T}$ *has either a finite or an enumerable ordinary model.*

Since we are interested only in ordinary realizations, from now on we shall use the word "realization" in the sense "ordinary realization".

7. Deduction theorem and the theorem on the existential quantifier

Like in the classical logic, the deduction theorem also holds in the extended algorithmic logic.

THEOREM 7.1. *If* A *is a set of formulas,* β *is a formula, and* α *is a closed formula, then* $(\alpha \Rightarrow \beta)$ *is a theorem in* $C(A)$ *iff* β *is a theorem in* $C(A \cup \{\alpha\})$.

The easy proof based on the completeness theorem and on the definition of a realization is omitted (see [23], p. 313).

By FSF we denote the set of all formulas in F which do not contain any classical quantifiers.

The algorithmic logic has been usually founded upon the set FSF of formulas (see [22], [24]). The following theorem shows that the extension by classical quantifiers is essential.

THEOREM 7.2. *The existential quantifier* $\exists$ *is not definable by means of formulas from the set* FSF.

Proof: It is sufficient to take the relational system $\mathfrak{A} = \langle J, P, = \rangle$, where $J = \{1, 2, 3\}$ and P is the binary relation defined as follows: $P(j, k)$ iff $j = 1$ and $k = 2$. Let $\overline{P}$ be the binary predicate whose realization $\overline{P}_{\mathfrak{A}}$ in the system $\mathfrak{A}$ is the relation P.

Let v be a valuation in the sets J and B_0. For $q = 1, 2, 3$, by v^q we shall denote the following valuation:

$$v^q(x) = \begin{cases} q & \text{if } x \text{ is an individual variable,} \\ v(x) & \text{if } x \text{ is a propositional variable.} \end{cases}$$

Let us observe that:

(1) $$(\overline{P}(xy))_{\mathfrak{A}}(v^q) = \Lambda \quad \text{for} \quad q = 1, 2, 3,$$

(2) $$(x = y)_{\mathfrak{A}}(v^q) = V \quad \text{for} \quad q = 1, 2, 3,$$

(3) $$r_{\mathfrak{A}}(v^q) = v(r) \quad \text{for} \quad q = 1, 2, 3 \text{ and a propositional variable } r.$$

Hence we obtain that for every open formula α, the value $\alpha_{\mathfrak{A}}(v^q)$ does not depend on the choice of q. Now observe that for every substitution s in S, $s_{\mathfrak{A}}(v^q)(x) = q$ for $q = 1, 2, 3$ and for x being an individual variable, and that the value $s_{\mathfrak{A}}(v^q)(r)$ does not depend on q for any propositional variable r. Reasoning by induction on the length of a program K in FS, we obtain that the halting of K in the system $\mathfrak{A}$ for the valuations of the form v^q does not depend on the value of q; and moreover, if $K_{\mathfrak{A}}(v^{q_0})$ is defined for some $q_0 = 1, 2, 3$, then $K_{\mathfrak{A}}(v^q)(x) = q$ for $q = 1, 2, 3$ and

for x being an individual variable, and $K_{\mathfrak{A}}(v^q)(r)$ does not depend on q for any propositional variable r.

Now we shall prove that for every formula α in FSF:

(4) *for every valuation v, the value $\alpha_{\mathfrak{A}}(v^q)$ is constant and does not depend on q.*

The proof of (4) proceeds by induction on the length of α. On account of equalities (1), (2), and (3), the assertion (4) holds for all elementary formulas. We shall consider only the induction steps concerning the formation rule (F_5) in the definition of the set F of formulas in §1.

Let condition (4) hold for some formula α and let K be an arbitrary program. Then for each $q_1, q_2 = 1, 2, 3$ the valuation $K_{\mathfrak{A}}(v^{q_1})$ is defined iff $K_{\mathfrak{A}}(v^{q_2})$ is defined and if they are defined then $K_{\mathfrak{A}}(v^{q_1}) = \big(K_{\mathfrak{A}}(v^{q_2})\big)^{q_1}$ and

$$\alpha_{\mathfrak{A}}\big(K_{\mathfrak{A}}(v^{q_1})\big) = \alpha_{\mathfrak{A}}\Big(\big(K_{\mathfrak{A}}(v^{q_2})\big)^{q_1}\Big) = \alpha_{\mathfrak{A}}\big(K_{\mathfrak{A}}(v^{q_2})\big).$$

So $(K\alpha)_{\mathfrak{A}}(v^{q_1}) = (K\alpha)_{\mathfrak{A}}(v^{q_2})$.

Now, applying the induction on the natural number i, we can deduce that for all i, $(K^i\alpha)_{\mathfrak{A}}(v^{q_1}) = (K^i\alpha)(v^{q_2})$. Hence

$$\big(\bigcup K\alpha\big)_{\mathfrak{A}}(v^{q_1}) = \big(\bigcup K\alpha\big)_{\mathfrak{A}}(v^{q_2}) \quad \text{and} \quad \big(\bigcap K\alpha\big)_{\mathfrak{A}}(v^{q_1}) = \big(\bigcap K\alpha\big)_{\mathfrak{A}}(v^{q_2}).$$

Now we return to the proof of 7.2. Suppose that for the formula $\exists y \bar{P}(x\ y)$ there exists a formula α in FSF equivalent in $\mathfrak{A}$. Then $\big(\exists y\ \bar{P}(x, y)\big)_{\mathfrak{A}}(v^1) = V$, $\big(\exists y\ \bar{P}(x, y)\big)_{\mathfrak{A}}(v^2) = \Lambda$, but on account of (4), we have $\alpha_{\mathfrak{A}}(v^1) = \alpha_{\mathfrak{A}}(v^2)$. Thus, Theorem 7.2 is proved. ■

8. Normal form theorems

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

THEOREM 8.1 (*on the normal form of programs*; [15], [22]). *For every* FS-*program K, there exists a program M in the normal form such that K and M are strongly equivalent on the set $\big((V_i \cup V_0) - \big(V(M) - V(K)\big)\big)$ 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 $x \notin V(M) - V(K)$ we have $K_R(v)(x) = M_R(v)(x)$.*

The variables $V(M) - V(K)$ are auxilliary ones. They can always be chosen so as not to occur in the formulas under consideration.

A formula is in the prenex normal form provided it is of the form $Q_1 Q_2 \dots Q_n \alpha$, where

(1) α is an open formula,

(2) for each $i = 1, \dots, n$, Q_i is of one of the forms: $\exists x$, $\forall x$, $s \bigcup K$ or $s \bigcap K$, where x is an individual variable, s is a substitution or the empty word and K is a loop-free program.

The main result of this section is this:

THEOREM 8.2 (*on the normal form of formulas*). *For every formula* α *in* F, *there exists a formula* β *in the prenex normal form such that* $\alpha \equiv \beta$ (which is equivalent to the fact that $(\alpha \Leftrightarrow \beta) \in C(\emptyset)$).

Proof: In the first step we transform the formula α to its simplified form α'. In the next step we remove all remaining stars. They can appear in programs K in the context $\bigcup K\delta$. We use the following fact:

LEMMA 8.3. *Let* M: $\circ[M_1 * [\gamma M_2]]$ *be a program in the normal form equivalent to a program* K *and such that* $(V(M) - V(K)) \cap V(\alpha) = \emptyset$. *Then*

$$\bigcup K\alpha \equiv \alpha \vee \left(M_1 \bigcup \preceq [\gamma M_2\, M_1](\neg\gamma \wedge \alpha)\right)'. \tag{1}$$

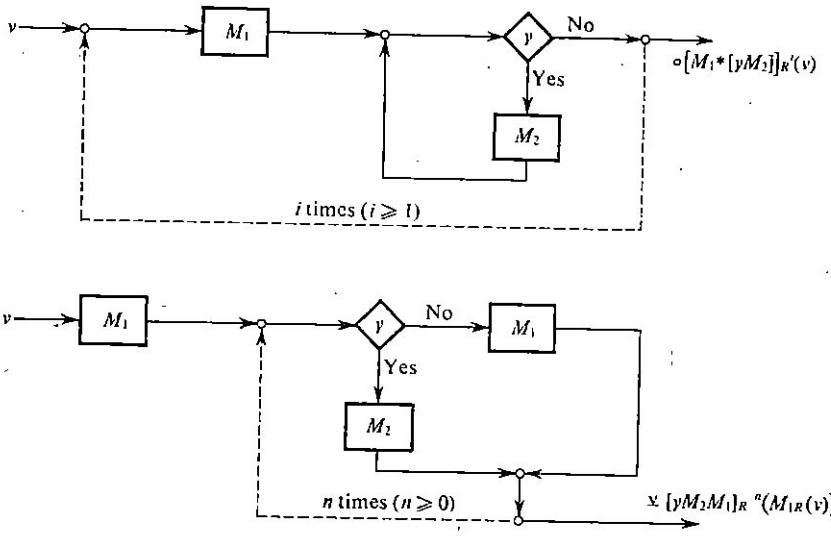
Proof: In virtue of the definition of the realization, $\bigcup K\alpha \equiv \bigcup M\alpha$. Hence, on account of Lemma 5.1, the equivalence (1) is equivalent to the following one:

$$\bigcup M\alpha \equiv \alpha \vee M_1 \bigcup \preceq [\gamma M_2\, M_1](\neg\gamma \wedge \alpha). \tag{2}$$

Now let R be a realization and v a valuation. Let us consider the following equality:

$$\circ[M_1 * [\gamma M_2]]^i_R(v) = \preceq [\gamma M_2\, M_1]^n_R\left(M_{1R}(v)\right). \tag{3}$$

The transformations defined by the left-hand side and the right-hand side of the above equality can be illustrated by means of the following flow-diagrams:



In order to prove equivalence (2), it is sufficient to observe the following facts:

(4) *for every natural* $i \geqslant 1$, *if* $\circ [M_1 * [\gamma M_2]]^i_R(v)$ *is defined, then there exists a natural number* n *such that equality* (3) *holds*;

(5) *for every natural* n, *if* $\gamma_R\big(\preceq [\gamma M_2\ M_1]^n_R(M_{1R}(v))\big) = \Lambda$, *then there exists a natural number* $i \geqslant 1$ *such that equality* (3) *holds.* ■

Now we come back to the proof of Theorem 8.2. On account of Lemma 8.3, we can transform each formula to an equivalent loop-free formula. The classical quantifiers can be pulled ahead the propositional connectives like in the first order logic. In the case of iteration quantifiers we apply the following lemma:

LEMMA 8.4. *Let* s *be a substitution,* K *a loop-free program and* α, β *formulas in* F. *Let* $x_1, \ldots, x_n$ *be all the variables in the expressions* s, K, α, *and* β. *Let* $y_1, \ldots, y_n$ *be a copy of the sequence* $x_1, \ldots, x_n$ (see §2).

Then

(1) $$\neg s \cap K\alpha \equiv s \cup K(\neg\alpha),$$

(2) $$\neg s \cup K\alpha \equiv s \cap K(\neg\alpha),$$

(3) $$s \cap K\alpha \vee \beta \equiv ([y_1/x_1 \ldots y_n/x_n] \circ s) \cap K\big(\alpha \vee ([x_1/y_1 \ldots x_n/y_n]\beta)'\big),$$

(4) $$s \cup K\alpha \vee \beta \equiv ([y_1/x_1 \ldots y_n/x_n] \circ s) \cup K\big(\alpha \vee ([x_1/y_1 \ldots x_n/y_n]\beta)'\big).$$

Proof: The proofs of the above equivalences apply only some simple facts concerning the notion of a realization. We shall prove here assertions (2) and (3). Let R be a realization and v a valuation. By virtue of the fact that K is a loop-free program, it follows that

(5) $$K^i_R(v) \text{ is defined for every natural } i.$$

Hence

$$(\neg s \cup K\alpha)_R(v) = \neg \underset{i \in N}{\text{l.u.b.}}\ \alpha_R\big(K^i_R(s_R(v))\big)$$
$$= \underset{i \in N}{\text{g.l.b.}}\ (\neg\alpha)_R\big(K^i_R(s_R(v))\big) = \big(s \cap K(\neg\alpha)\big)_R(v).$$

Thus the equivalence (2) is proved.

Observe that, on account of the choice of the variables $y_1, \ldots, y_n$, for every natural i we have

$$K^i_R\big(s_R([y_1/x_1 \ldots y_n/x_n]_R(v))\big)(z)$$
$$= \begin{cases} K^i_R(s_R(v))(z) & \text{if } z = x_q \text{ for some } q = 1, \ldots, n, \\ [y_1/x_1 \ldots y_n/x_n]_R(v)(z) & \text{if } z = y_q \text{ for some } q = 1, \ldots, n, \\ v(z) & \text{otherwise.} \end{cases}$$

Hence, according to the assumptions on the variables, it follows that

(6) $$\alpha_R\big(K^i{}_R\big(s([y_1/x_1 \ldots y_n/x_n]_R(v))\big)\big) = \alpha_R\big(K^i_R(s_R(v))\big)$$

and

$$(7)\qquad ([x_1/y_1 \dots x_n/y_n]\beta)_R\Big(K^i_R\big(s_R([y_1/x_1 \dots y_n/x_n]_R(v))\big)\Big) = \beta_R(v).$$

Now, applying the fact (5) and equalities (6) and (7), we get consecutively that

$$\begin{aligned}(s\bigcap K\alpha\vee\beta)_R(v) &= \operatorname*{g.l.b.}_{i\in N}\alpha_R\big(K^i_R(s_R(v))\big)\vee\beta_R(v)\\ &= \operatorname*{g.l.b.}_{i\in N}(\alpha\vee[x_1/y_1\dots x_n/y_n]\beta)_R\Big(K^i_R\big(s_R([y_1/x_1\dots y_n/x_n]_R(v))\big)\Big)\\ &= \big(([y_1/x_1\dots y_n/x_n]\circ s)\bigcap K(\alpha\vee[x_1/y_1\dots x_n/y_n]\beta)\big)_R(v).\end{aligned}$$

Hence, by Lemma 5.1, follows the equivalence (3). ■

EXAMPLE. Let K, M be loop-free programs, s_1, s_2 substitutions and $\alpha, \beta, \gamma, \delta$ open formulas. Let $\mathbf{x} = (x_1, \dots, x_n)$ be all the variables occurring in the above expressions. Let $\mathbf{y}, \mathbf{z}$, and $\mathbf{t}$ be mutually disjoint copies of the sequence x. Below we shall transform the formula $\big(\circ[s_1 * [\gamma K]]\alpha \Rightarrow \circ[s_2 * [\delta M]]\beta\big)$ to an equivalent formula in the prenex normal form. Namely:

$$\begin{aligned}&\big(\circ[s_1 * [\gamma K]]\alpha \Rightarrow \circ[s_2 * [\delta M]]\beta\big)\\ &\equiv \big(s_1 \bigcup \preceq[\gamma K[\ ]](\alpha\wedge\neg\gamma) \Rightarrow s_2 \bigcup \preceq[\delta M[\ ]](\beta\wedge\neg\delta)\big)\\ &\equiv \big(s_1\bigcap \preceq[\gamma K[\ ]](\neg\alpha\vee\gamma) \vee s_2\bigcup \preceq[\delta M[\ ]](\beta\wedge\neg\delta)\big)\\ &\equiv ([\mathbf{y}/\mathbf{x}]\circ s_1)\bigcap \preceq[\gamma K[\ ]]\big(\neg\alpha\vee\gamma\vee([\mathbf{x}/\mathbf{y}]\circ s_2) \bigcup \preceq[\delta M[\ ]](\beta\wedge\neg\delta)\big)\\ &\equiv ([\mathbf{y}/\mathbf{x}]\circ s_1)\bigcap \preceq[\gamma K[\ ]]\big([\mathbf{z}/\mathbf{x}\ \mathbf{t}/\mathbf{y}]\circ([\mathbf{x}/\mathbf{y}]\circ s_2)\big)\\ &\qquad\qquad \bigcup \preceq[\delta M[\ ]]\big((\beta\wedge\neg\delta)\vee([\mathbf{x}/\mathbf{z}\ \mathbf{y}/\mathbf{t}](\neg\alpha\wedge\gamma))\big)\big).\end{aligned}$$

References

[1] deBakker, J. W., de Roever, W. P., *Calculus for recursive program schemes*, Stichting Mathematisch Centrum, MR 131/72, Amsterdam 1972.

[2] Banachowski, L., *Investigations of properties of programs by means of the extended algorithmic logic*, doct. diss,. Warsaw University, 1975.

[3] –, *An axiomatic approach to the theory of data structures*, Bull. Acad. Polon. Sci., Sér. Sci. Math. Astronom. Phys. 23 (1975), 315–323.

[4] –, *Extended algorithmic logic and properties of programs*, ibid., 325–330.

[5] –, *Modular properties of programs*, ibid, 331-337.

[6] –, *Investigations of properties of programs by means of the extended algorithmic logic* (part II of this paper), Fund. Inf. to appear.

[7] Banachowski, L., Szczepańska, D., Wasersztrum, W., Żurek, J., *Automatic verification of program correctness*, Problemy węzłowe, Warsaw University, 1974 (in Polish).

[8] Floyd, R. W., *Assigning meanings to programs*, Proc. of Symposia in Applied Mathematics 19 (1967), AMS.

[9] Góraj, A., Mirkowska, M., Paluszkiewicz, A., *On the notion of description of program*, Bull. Acad. Polon. Sci., Sér. Sci. Math. Astronom. Phys. 18 (1970), 499–506.

[10] Hoare, C. A. R., *An axiomatic basis of computer programming*, Comm. ACM 12 (1969), 576–583.

[11] –, *Procedures and parameters: an axiomatic approach*, Symp. on Semantics of Algorithmic Languages, Springer Verlag, 1971.

[12] King, J. C., *A program verifier*, Proc. IFIP (1971), 234–249.

[13] King, J. C., Floyd, R. W., *An interpretation oriented theorem prover over integers*, J. Comput. System Sci. 6 (1972), 305–323.

[14] Kreczmar, A., *Efffectivity problems of algorithmic logic*, doct. diss., Warsaw University, 1973 (in Polish).

[15] –, *Effectivity problems of algorithmic logic*, 2nd Colloquium on Automata, Languages and Programming, Springer Verlag, 1974.

[16] Manna, Z., *The correctness of programs*, J. Comput. System Sci. 3 (1969).

[17] Manna, Z., Pnueli, A., *Formalization of properties of functional programs*, J. of ACM, 17 (1970), 555–569.

[18] –, –, *Axiomatic approach to total correctness of programs*, Acta Informatica, 1974.

[19] Mazurkiewicz, A., *Proving properties of processes*, Algorytmy 11 (1974), 5–21.

[20] Mirkowska, M., *On formalized systems of algorithmic logic*, Bull. Acad. Polon. Sci., Sér. Sci. Math. Astronom. Phys. 19 (1971), 421–428.

[21] –, *Herbrand theorem in algorithmic logic*, ibid. 22 (1974), 539–543.

[22] –, *Algorithmic logic and its applications in the theory of programs*, doct. diss., Warsaw University, 1972 (in Polish).

[23] Rasiowa, H., Sikorski, R., *Mathematics of metamathematics*, PWN, Warsaw 1968.

[24] Salwicki, A., *Formalized algorithmic languages*, Bull. Acad. Polon. Sci., Sér. Sci. Math. Astronom. Phys. 18 (1970), 227–232.

[25] –, *On the equivalence of FS-expressions and programs*, ibid., 275–278.

[26] –, *Programmability and recursiveness (an application of algorithmic logic to procedures)*, Diss. Math. to appear.

[27] *Efficient production of large programs*, Proc. of Intern. Workshop Jablonna 1970, CC PAS, PWN, Warsaw 1971.