ANNALES SOCIETATIS MATHEMATICAE POLONAE
Series IV: FUNDAMENTA INFORMATICAE I (1977), 195–230

Programmability in fields

ANTONI KRECZMAR

Institute of Informatics, University of Warsaw

Received May 15, 1976

AMS Categories: 68A05

Abstract. In the present paper we investigate algorithmic properties of fields. We prove that axioms of formally real fields for the field $\boldsymbol{R}$ of reals and axioms of fields of characteristic zero for the field $\boldsymbol{C}$ of complex numbers, give the complete characterization of algorithmic properties. By Kfoury's theorem programs which define total functions over $\boldsymbol{R}$ or $\boldsymbol{C}$ are effectively equivalent to loop-free programs. Examples of programmable and nonprogrammable functions and relations over $\boldsymbol{R}$ and $\boldsymbol{C}$ are given. In the case of ordered reals the axioms of Archimedean ordered fields completely characterize algorithmic properties. We show how to use the equivalent version of Archimed's axiom (the exhaustion rule) in order to prove formally the correctness of some iterative numerical algorithms.

Key words: programs and programmability, algorithmic properties, programmability in fields, axioms for algorithmic properties of reals, ordered reals and complex numbers.

Introduction

Algorithm in its primitive but already universal sense is as old as mathematics. Since the very beginning people aspired to automatize almost all mathematical tasks. From arithmetic of integers throughout geometry and algebra till the differential and integral calculi this natural tendency has had a considerable influence on the development of mathematics.

The theory of algorithms arose much later than logic and foundations of mathematics. At first it was a fragment of recursion theory. Algorithms considered there were up to isomorphism algorithms over natural numbers. Moreover, the investigations in recursion theory concentrated on the effectiveness and noneffectiveness of mathematical objects. Algorithms were more or less rather tools than objects of these investigations.

The development of computer hardware and software, in particular the revolution caused by the creation of high-level programming languages, forced the theory of algorithms to take quite a different point of view. First of all, in a programming language we can define, and hence analyse algorithms over domains which are neither enumerable nor effective as, for instance, the field of real numbers. Moreover, algorithms are now expressions of a formal language and their semantics may be defined similarly to that introduced in metamathematics.

Thus an algorithm is not connected with only one relational structure. We can treat it as a syntactical category of many relational structures. For instançe, Euclid's algorithm, which was primarily used to compute the greatest common divisor of two integers, is also valid in the case of polynomials and quite well may compute the ratio of two commeasurable segments.

In mathematics it happens very seldom that a formula or a formal proof is an object of investigations. A mathematician does not manipulate formulas of the theory he just develops. He thinks and works in semantical categories. In the case of computer practice it is just the opposite. A programmist treats his program not only semantically, as a mathematician his formulas, but also syntactically.

In theoretical computer science a formal proof of correctness of an algorithm becomes a very important task. The first inventors and advocates of it were J. McCarthy and C. Hoare [11], [17], but afterwards it was taken over by many authors. We also share this point of view. A formal proof of correctness ought to be the final but not, of course, the first step in the analysis of an algorithm. To carry out such a proof is not an easy matter. It is well known that the mathematical formal system of the theory of programming has to be essentially infinitistic. The choice of such a system is not a key to success, but at any rate it is very important. The list of these systems is not very long [2], [5], [6], [11], [18], [26], however we chose the system best known to the author.

This system is called *algorithmic logic*; it was proposed by A. Salwicki for the first time [24], then developed by a group of Warsaw logicians [3], [8], [16], [19], [21], [24], [25]. Having the logical framework, we can put concrete questions.

For instance, in algorithmic logic we consider formulas of the form $K\alpha$, where K is a program and α is an open formula.

The logical value of this formula is true if α is satisfied after the execution of K, otherwise it is false. An algorithmic property, a syntactical category introduced by E. Engeler [6], takes in algorithmic logic the form of a Boolean combination of formulas $K\alpha$. The halting property, the strong and weak equivalence of programs, the partial and total correctness of a program are well known examples of algorithmic properties.

In the present paper we try to characterize algorithmic properties over fields; the fields which are the major part of our investigations are the fields of real and complex numbers.

This analysis concentrates on the problem of axiomatization of algorithmic properties, i.e., briefly speaking, it answers the question what properties of a field must be taken under consideration in programming functions and relations over it; whether, for instance, the continuity of the reals is necessary to prove the correctness of a program.

It will turn out that solely the properties of formally real fields and those of fields of characteristic zero are completely sufficient to characterize algorithmic properties of the field of real and complex numbers, respectively. This observation implies that programs which are total over these fields can be effectively transformed to the loop-free form. This does not concern programs over the ordered field of reals. In this case the axiom of Archimedes plays the most essential role.

The first section of this paper is devoted to a brief exposition of algorithmic logic. In the second we consider the general questions concerning programmability in fields. In subsequent sections we investigate the fields of complex, real, ordered real and rational numbers, respectively.

I. Programs and programmability

By a *data structure* we mean a relational system $\mathfrak{A}$ consisting of a non-empty set A, some relations $P_i \subseteq A^{n_i}$, some functions $f_j\colon A^{n_j} \to A$, and some constants $c_k \in A$. With respect to such a data structure we imagine a set of elementary predicates, functional symbols and constants of a corresponding programming language. It is called an *algorithmic basis*.

With the aid of an algorithmic basis we construct programs over a data structure $\mathfrak{A}$. The simplest program is called an *assignment statement* and is written in the form

$$x := \tau \quad \text{or} \quad p := \alpha,$$

where x denotes an individual variable, τ a term, p a propositional variable, and, finally, α denotes an open formula.

To construct more complicated programs we use three categories of statements:

compound statement	**begin** K; M **end**
if statement	**if** α **then** K **else** M
loop statement	**while** α **do** K

where α is an open formula and K and M are programs.

Let K be a program and let $VR(K)$ denote the set of all variables occurring in a program K. By *valuation* v we understand a mapping $v\colon VR(K) \to A \cup B$ where B is the two-element Boolean algebra and $v(x) \in A$, $v(p) \in B$, provided x is an individual and p is a propositional variable.

If K is a program and v a valuation of its variables, we can define in a natural way the *valuation* $K[v]$, i.e., the state of variables after the execution of K with the initial values of variables defined by v if, of course, the computation is finite; otherwise $K[v]$ is undefined.

The variables from the set $VR(K)$ are divided into three not necessarily disjoint sets: input, output and program variables. If $x_1, \ldots, x_n$ are input and x is the output variable of a program K $(K(x_1, \ldots, x_n; x))$, then K defines a partial function $f\colon A^n \to A$ as follows:

$$f(a_1, \ldots, a_n) = K[v](x),$$

where $v(x_i) = a_i$ for $i = 1, \ldots, n$.

A partial function $f\colon A^n \to A$ is said to be *programmable* over a data structure $\mathfrak{A}$ iff there exists a program K with input variables $x_1, \ldots, x_n$ and output variable x such that f is defined by $K(x_1, \ldots, x_n; x)$. A total function f is *programmable* iff f is programmable by a program K which halts for all initial values of input variables $x_1, \ldots, x_n$.

In order to investigate programs we introduce the notion of algorithmic property (see Engeler [6]). The simplest example of an algorithmic property is just

$$K\alpha$$

where K is a program and α is an open formula. The *logical value* of a formula $K\alpha$ for a valuation v is equal to $\alpha[K[v]]$ if $K[v]$ is defined, otherwise $K\alpha$ is not satisfied by the valuation v.

A Boolean combination of formulas of the above form is called an *algorithmic property*. Examples of algorithmic properties are as follows.

Let $\mathbf{1}$ and $\mathbf{0}$ denote logical constants of truth and falsehood, respectively. Then the formula

$$K\mathbf{1}$$

defines the *halting property* of a program K,

$$K\mathbf{1} \Rightarrow (\alpha \Rightarrow K\beta) \quad \text{and} \quad \alpha \Rightarrow K\beta$$

define the *partial correctness* and the *correctness* of a program K with respect to an input condition α and an output condition β. Next, the formula

$$\textbf{begin } y_1 := x_1;\ y_2 := x_2;\ \ldots;\ y_n := x_n;\ K;\ M \textbf{ end } (x_1 = y_1) \wedge \ldots \wedge (x_n = y_n)$$

defines the *weak equivalence* of programs K and M, provided $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ are all variables occurring in K and M, respectively.

We shall denote the above formula by $K \simeq M$. Further, the formula:

$$(K1 \Leftrightarrow M1) \wedge (K \simeq M)$$

defines the *strong equivalence* of programs K and M and will be denoted by $K \equiv M$.

A relation $P \subseteq A^n$ is *programmable* over a data structure $\mathfrak{A}$ iff there exist a program K and an open formula α in the language of algorithmic basis of $\mathfrak{A}$ such that for input variables $x_1, \ldots, x_n$ of a program K

$$P(a_1, \ldots, a_n) \quad \text{iff} \quad \alpha[K[v]],$$

where $v(x_i) = a_i$, $i = 1, \ldots, n$.

A relation P is said to be *strongly programmable* if it is programmable by a program K and a formula α and if, moreover, that program halts for all initial values of input variables $x_1, \ldots, x_n$.

In algorithmic logic (see [24]) we introduce additionally two kinds of quantifiers, classical and iterational. The latter ones are written in the following way:

$$\bigcup K\alpha \quad \text{and} \quad \bigcap K\alpha,$$

where K is a program. Their realization is defined as follows:

$$\left(\bigcup K\alpha\right)[v] = \sup_{i \in \omega}(K^i \alpha[v]),$$

$$\left(\bigcap K\alpha\right)[v] = \inf_{i \in \omega}(K^i \alpha[v]).$$

In algorithmic logic we can define *non-elementary classes of models*. For example, the algorithmic property:

$$\textbf{begin } x := 0; \textbf{ while } x \neq y \textbf{ do } x := x+1 \textbf{ end } (x = y)$$

says that y is a natural number. So, the formula

$$(\forall y) \textbf{ begin } x := 0; \textbf{ while } x \neq y \textbf{ do } x := x+1 \textbf{ end } 1$$

with some finite number of additional axioms defines the standard model of arithmetic.

Similarly the axiom

$$(\forall x)(\forall y)(x > 0 \wedge y > 0 \Rightarrow z := y \bigcup (z := z+y)(x < z)$$

says that $<$ is an Archimedean order.

Both these properties are known not to be of the first order.

We introduce the notion of *semantic consequence* as usual. If α is a formula of algorithmic logic, $\mathfrak{A}$ is a data structure and v is a valuation, then $\mathfrak{A} \models \alpha[v]$ will denote that α is satisfied by v in $\mathfrak{A}$. $\mathfrak{A} \models \alpha$ will denote that α is valid in $\mathfrak{A}$, i.e. is satisfied by every valuation. If Γ denotes a set of formulas, α is a formula, then $\Gamma \models \alpha$ means that for every data structure $\mathfrak{A}$, if all formulas of Γ are valid in $\mathfrak{A}$, then α is also valid in $\mathfrak{A}$. In particular, $\models \alpha$ means that α is a tautology.

This semantic consequence possesses the complete syntactic characterization presented in several papers (see [16], [19]).

If $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are two data structures with the same algorithmic basis, they are said to be *algorithmically equivalent* if for every algorithmic property α

$$\mathfrak{A}_1 \models \alpha \quad \text{iff} \quad \mathfrak{A}_2 \models \alpha.$$

From this definition it follows that programs are interchangeable between two algorithmically equivalent data structures.

After this list of definitions we now pass to some general facts concerning programs. They will make it possible to effectively transform formulas expressing algorithmic properties into normal forms. The first lemma concerns the normal form of a program, the others the normal form of an algorithmic property.

DEFINITION 1. A program K is *in the normal form* iff it is of the form:

$$\textbf{begin } K_1; \textbf{ while } \alpha \textbf{ do } K_2 \textbf{ end}$$

where K_1, K_2 are loop-free programs.

LEMMA 1. *There is an effective way of transforming every program K into a program M in the normal form such that $\models K \equiv M$.*

Proof in [16].

LEMMA 2. *There is an effective transformation which for every two programs K, M and every two open formulas α, β gives a program N and an open formula δ such that*

$$\models (K\alpha \vee M\beta) \Leftrightarrow N\delta.$$

Proof: By Lemma 1 we can consider the case when programs K and M are in the normal form:

$$\textbf{begin } K_1; \textbf{ while } \gamma_1 \textbf{ do } K_2 \textbf{ end},$$
$$\textbf{begin } M_1; \textbf{ while } \gamma_2 \textbf{ do } M_2 \textbf{ end}.$$

Let us consider the following program (written in Algol-like notation):

```
begin boolean $q_1, q_2, r$;
      $K_1$; $M_1$; $p := q_1 := q_2 := r :=$ true;
20:
```

```
if $q_2 \Rightarrow q_1 \wedge r$ then
begin
   if $\gamma_1$ then begin $K_2$; $r := \neg r$; goto 20 end
   else if $a$ then begin $p := $ true; goto 30 end
   else if $q_2$ then begin $q_1 := $ false; goto 20 end
   else begin $p := $ false; goto 30 end
end else
begin
   if $\gamma_2$ then begin $M_2$; $r := \neg r$; goto 20 end
   else if $\beta$ then begin $p := $ true; goto 30 end
   else if $q_1$ then begin $q_2 := $ false; goto 20 end
   else begin $p := $ false; goto 30 end
end;
30:
end
```

Let us denote by N a program which is obtained from the above one after the elimination of labels 20, 30 (see [15], [25]). We shall prove that the formula Np is that mentioned in the lemma.

We consider the following table of possible values of the formula $Ka \vee M\beta$ treating it as a function of values of K, M, a, β. In this table $+$ denotes that a program is defined, $-$ that it is undefined, **1** and **0** denote, as usual, truth and falsehood.

	1	2	3	4	5	6	7	8	9
K	$-$	$-$	$-$	$+$	$+$	$+$	$+$	$+$	$+$
Ka	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**
M	$-$	$+$	$+$	$-$	$+$	$+$	$-$	$+$	$+$
$M\beta$	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
$Ka \vee M\beta$	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**

Let us consider, for example, cases 2 and 3. In both cases K is undefined, hence the condition γ_1 is always fulfilled. The two-cycle way of working of the program N is controlled by the variable r. In the considered cases the value of q_1 remains always true, because γ_1 is always true. Thus the condition $q_2 \Rightarrow q_1 \wedge r$ is equivalent to $q_2 \Rightarrow r$. But M is defined, hence after a certain number of steps the condition γ_2 will not be satisfied. Now we have two possibilities. The condition β is or is not satisfied. In the first case p becomes true and the program halts, Np is true. In the latter case q_1 is true, hence q_2 becomes false and the condition $q_2 \Rightarrow r$ will be always satisfied, N does not terminate computation and Np is false.

The remaining cases can be proved in a similar way. ■

DEFINITION 2. An algorithmic property is said to be *primitive* if it is of the following form:

$$K\alpha \Rightarrow M\beta$$

where K, M are programs and α, β are open formulas.

DEFINITION 3. An algorithmic property is said to be *in the normal form* if it is a conjunction of primitive formulas.

LEMMA 3. *There is an effective way of transforming every algorithmic property α into the normal form β such that*

$$\models \alpha \Leftrightarrow \beta .$$

Proof: First of all we transform an algorithmic property into the conjunctive-disjunctive normal form of propositional calculus. Negated elements of disjunctions can be unified with the aid of the following equivalence:

$$\models (\neg K\alpha \vee \neg M\beta) \Leftrightarrow (\textbf{begin } K;\ M \textbf{ end } (\alpha \wedge \beta))$$

after renaming variables occurring in these formulas with due care.

The unification of unnegated elements of disjunctions is given by Lemma 2.

We see that after these two steps of transformation we obtain the conjunction of formulas of the following form:

$$K\alpha \vee \neg M\beta \quad \text{equivalent to} \quad M\beta \Rightarrow K\alpha . \ \blacksquare$$

Let us again consider an algorithmic property of the form $K\alpha$. This formula may be effectively transformed into the more convenient form. This results from the following:

LEMMA 4 (Engeler [6]). *There is an effective way of transforming every formula $K\alpha$ to a formula of the form*

$$K_1 \bigcup K_2 \beta ,$$

where K_1, K_2 are loop-free programs and β is an open formula such that

$$\models K\alpha \Leftrightarrow K_1 \bigcup K_2 \beta .$$

For the proof see [24].

By the above Lemma a formula $K\alpha$ is equivalent to an open formula of the first order predicate calculus if K is a loop-free program, or to a formula of infinitistic language $L_{\omega_1\omega}$ of the following form:

$$\bigvee_{i\in\omega} \alpha_i(x_1, \ldots, x_n),$$

where all formulas α_i are open and effectively given.

Looking back at Lemmas 1–4, we observe that all we want to investigate in the theory of algorithmic properties of structures can be reduced to the case of formulas:

$$(\forall x_1, \ldots, x_n)\big(\bigvee_{i\in\omega} a_i(x_1, \ldots, x_n) \Rightarrow \bigvee_{i\in\omega} \beta_i(x_1, \ldots, x_n)\big).$$

If, for example, we want to show the algorithmic equivalence of two data structures, it suffices to show it for formulas of the above form.

One of the most important algorithmic properties is the halting property. In fact, the first what we must prove about our program is prove that it halts on all initial values of input variables from the considered domain. If a program has no loops, we need not do anything, because it always halts, otherwise the problem becomes very difficult. In the case of programs over the data structure of natural numbers the halting problem embraces all recursively enumerable sets.

Now, let us consider a class of data structure with the same algorithmic basis, i.e. assume that programs are written in the same language. We shall give one criterion which answers the following question: Is there a program K over that algorithmic basis which halts on all initial values of input variables from every data structure of that class and which is not strongly equivalent in that class to a loop-free program?

This criterion was first formulated by M. Paterson [20], next it was strengthened by D. Kfoury. We shall give it in the version very close to that formulated by Kfoury.

THEOREM 1 ([14]). *Let Γ and $\{a_i\}_{i\in\omega}$ be two sets of formulas for which Łoś's theorem about ultraproduct is valid* ([4]). *Let us denote by φ an infinite formula*

$$(\forall x_1, \ldots, x_n) \bigvee_{i\in\omega} a_i(x_1, \ldots, x_n),$$

and by φ_k its finite segment

$$(\forall x_1, \ldots, x_n) \bigvee_{i\leqslant k} a_i(x_1, \ldots, x_n).$$

If $\Gamma \models \varphi$, then there exists $k \in \omega$ such that $\Gamma \models \varphi_k$.

Proof: Let us suppose that for every $k \in \omega$ Γ non $\models \varphi_k$. Then there exist a data structure $\mathfrak{A}_k$ and a valuation v_k such that

(1) $$\mathfrak{A}_k \models \Gamma \quad \text{and} \quad \mathfrak{A}_k \models \neg\psi_k[v_k],$$

where ψ_k: $\bigvee_{i\leqslant k} a_i(x_1, \ldots, x_n)$. Now, let $\mathfrak{A}$ be an ultraproduct:

(2) $$\mathfrak{A} = \prod_{k\in\omega} \mathfrak{A}_k/\nabla$$

modulo a nonprincipal ultrafilter $\nabla \subset 2^{\omega}$. From (1) it follows that $\mathfrak{A}$ is a model of Γ and that

$$\{i\colon \mathfrak{A}_i \models \neg a_k[v_i]\} = \{i\colon i \geqslant k\}. \tag{3}$$

Hence, from (3), $\{i\colon \mathfrak{A}_i \models \neg a_k[v_i]\} \in \nabla$ and for the valuation $v \overset{\mathrm{df}}{=} \{v_i\}_{i\in\omega}/\nabla$ we have

$$\mathfrak{A} \models \neg a_k[v] \quad \text{for every } k \in \omega,$$

which implies that $\mathfrak{A}$ non $\models \varphi$. ■

Let Γ be a set of formulas of the first order predicate calculus. In view of Lemma 4 the halting formula may be expressed in the form

$$(\forall x_1, \ldots, x_n) \bigvee_{i\in\omega} K_1 K_2^i \neg a(x_1, \ldots, x_n),$$

where K_1, K_2 and a come from the normal form of a program K.

If K halts on all initial values of input variables for every data structure which is a model of Γ, then by Theorem 1 there is $k \in \omega$ such that

$$(\forall x_1, \ldots, x_n) \bigvee_{i\leqslant k} K_1 K_2^i \neg a(x_1, \ldots, x_n) \tag{4}$$

is equivalent to the halting formula of K. Hence, the programs K and $K_1 K_2^k$ have the same domains, and the program

```
begin $K_1$; if $a$ then $K_2$;
          if $a$ then $K_2$;
          . . . . . . . .
          if $a$ then $K_2$
end
```

where the if-statement is repeated k times, is strongly equivalent to K on all data structures which are models of the set of formulas Γ.

In particular, if Γ is empty, then a program K for which the formula $K\mathbf{1}$ is a tautology is strongly equivalent to a loop-free program (see Paterson [20]). Moreover, we can eliminate loops effectively, because the relation $\models a$, where a is open, is recursive.

The phenomenon of effective elimination of loops is much more general than in Paterson's version. Namely, consider an arbitrary set Γ of first order axioms. If Γ is recursively enumerable, then the elementary theory with this set of axioms is also recursively enumerable. Let K be a program over data structures which are models of Γ. If the halting property of K is a semantical consequence of Γ, then by Theorem 1 there exists a loop-free program such that

$$\Gamma \models K \equiv M.$$

But this program may be obtained effectively. In fact, it is sufficient to find an integer k such that (4) is a semantical consequence of Γ. Now we generate all proofs of the first order theory with axioms Γ. After every step we verify if it is the proof of (4). Owing to the condition that for some $k \in \omega$ (4) is the theorem of this theory, the process always terminates.

From the above it follows that programs which are total in some elementary axiomatizable class of models are effectively transformable to loop-free programs. In particular, analysis of programs over fields, rings, groups, Boolean algebras and so on is reduced to the case of loop-free programs.

II. Algorithmic properties in fields

In what follows we shall consider exclusively programmability in fields. First of all, we observe that we can disregard propositional variables. In fact, we replace every propositional variable by an individual one, the logical constants by the field constants 0, 1, and propositional connectives by the following field operations:

$$\neg p \quad \text{by} \quad (1-x),$$

and

$$p \wedge q \quad \text{by} \quad x \cdot y,$$

where x, y are individual variables which correspond to propositional variables p, q.

Finally, we replace every condition α by a condition $\tau = 1$ where τ is a term corresponding to this open formula α.

So, from now on we assume that all variables vary through some field $\mathfrak{F}$ and all non-logical constants are interpreted as field operations $+, -, \cdot, ^{-1}$ and the relation $=$ (in some cases $<$, when we consider an ordered field). We shall also investigate programmability in field $\mathfrak{F}$ with some additional operations and relations, as for example the radicals in the case of the field $\mathfrak{C}$ of complex numbers or the square root of positive numbers in the case of the field $\mathfrak{R}$ of real numbers.

Now, let us consider the simplest open formulas in the language of fields, i.e. atomic formulas and negations of atomic formulas. Without loss of generality we can consider the following cases:

$$f \cdot g^{-1} = 0 \quad \text{and} \quad f \cdot g^{-1} \neq 0,$$

where f and g are polynomials in n variables. They are equivalent to the formulas:

$$f = 0 \wedge g \neq 0 \quad \text{and} \quad f \neq 0 \wedge g \neq 0,$$

respectively.

An infinite formula (see § I)

$$(1) \qquad \bigvee_{i \in \omega} \alpha_i (x_1, \ldots, x_n)$$

may be transformed into the above form, where every a_i is a conjunction of atomic formulas or their negations of the following form:

$$f(x_1, \ldots, x_n) = 0 \quad \text{or} \quad f(x_1, \ldots, x_n) \neq 0.$$

Let $\mathfrak{F}$ be a field and let $f_1, \ldots, f_m$ be a set of polynomials of n variables with coefficients in F. The set of common zeros of the polynomials $f_1, \ldots, f_m$ is called an *algebraic variety*. It is clear that the join and the meet of algebraic varieties is again an algebraic variety. So, in terms of algebraic varieties, formula (1) can be expressed in the form:

$$\bigcup_{i \in \omega} (B_i - A_i), \tag{2}$$

where B_i are varieties defined by atomic formulas and A_i are varieties defined by negations of atomic formulas.

We shall examine infinite enumerable joins of the form (2). Let us observe that to prove the validity of formula (1) in some field $\mathfrak{F}$ is the same as to prove

$$\bigcup_{i \in \omega} (B_i - A_i) = F^n. \tag{3}$$

Condition (3) is infinitistic but in some cases it is equivalent to a finitistic one.

DEFINITION 1. The field $\mathfrak{F}$ *satisfies "finite covering condition"* iff, for every algebraic variety A and every enumerable set $\{B_i\}_{i \in \omega}$ of algebraic varieties over F, if $A \subset \bigcup_{i \in \omega} B_i$, then there exists a finite subset $I \subset \omega$ such that $A \subset \bigcup_{i \in I} B_i$.

Every finite field satisfies, of course, "finite covering condition". It will turn out that the fields of real and complex numbers also satisfy this condition.

On the other hand, every infinite enumerable field does not satisfy "finite covering condition". This results from the fact that every element of a field is a variety and the whole space F^n is also an algebraic variety.

In the following basic theorem we shall prove that in fields satisfying "finite covering condition", if (3) holds, then it holds for some finite subset of ω.

THEOREM 1. *Let $\{A_i\}$, $\{B_i\}$ be two enumerable sets of algebraic varieties over a field $\mathfrak{F}$ satisfying "finite covering condition". If*

$$\bigcup_{i \in \omega} (B_i - A_i) = F^n, \tag{4}$$

then there exists a finite subset $I \subset \omega$ such that

$$\bigcup_{i \in I} (B_i - A_i) = F^n. \tag{5}$$

Proof: By De Morgan's law we obtain from (4)

$$\bigcap_{i\in\omega}(A_i \cup -B_i) = \emptyset. \tag{6}$$

After the application of the generalized distributive law to (6) we have for every function $f\colon \omega \to 2$

$$\bigcap_{i\in\omega} D_i = \emptyset, \tag{7}$$

where

$$D_i = \begin{cases} A_i & \text{if } f(i) = 0, \\ -B_i & \text{if } f(i) = 1. \end{cases}$$

Taking as f a constant function we obtain from (7)

$$\bigcap_{i\in\omega} A_i = \emptyset \tag{8}$$

and

$$\bigcup_{i\in\omega} B_i = F^n. \tag{9}$$

If $\emptyset \neq J \neq \omega$, then taking

$$f(i) = \begin{cases} 0 & i \in J, \\ 1 & i \notin J; \end{cases}$$

by (7) we obtain

$$\bigcap_{i\in J} A_i \subset \bigcup_{i\notin J} B_i. \tag{10}$$

Let I be a finite subset of ω and let us denote by A the meet $\bigcap_{i\in I} A_i$. For $A \neq \emptyset$ consider a set $J = \{i\colon A \cap A_i = A\}$. Then $J \supset I$ and J is proper subset of ω, otherwise $\bigcap_{i\in\omega} A_i = A \neq \emptyset$ contrary to (8). By (10) and "finite covering condition" there exists I', a finite subset of $\omega - J$ such that

$$A \subset \bigcup_{i\in I'} B_i \tag{11}$$

and

$$\text{for every } i \in I',\ A \cap A_i \subsetneqq A. \tag{12}$$

Now we are ready to construct a finite set I satisfying (5). We define inductively a tree D whose elements will be finite sequences of integers. The root of D is the sequence: $\langle -1\rangle$. By (9) and "finite covering condition" there is a finite set I such that $F^n = \bigcup_{i\in I} B_i$. All sequences $\langle -1, i\rangle$ for $i \in I$ are also elements of D. Now, we proceed by induction. Let us suppose that $\langle -1, i_0, i_1, \ldots, i_k\rangle$ belongs to D. If $A = A_{i_0} \cap A_{i_1} \cap \ldots \cap A_{i_k}$ is empty, then $\langle -1, i_0, i_1, \ldots, i_k\rangle$ has no successors. If A is nonempty, then there exists a finite set $I' \subset \omega - \{i_0, \ldots, i_k\}$ such that (11) and (12) hold.

We add to D a finite number of successors of the element $\langle -1, i_0, i_1, \ldots, i_k\rangle$, i.e. $\langle -1, i_0, i_1, \ldots, i_k, i\rangle$ for $i \in I'$.

The tree D has no infinite path, because by the Hilbert basis theorem [13] there does not exist an infinite strictly decreasing sequence of algebraic varieties $A_1 \supsetneqq A_2 \supsetneqq \ldots$ and condition (12) guarantees that for $\langle -1, i_0, i_1, \ldots, i_k\rangle$ and $\langle -1, i_0, i_1, \ldots, i_k, i_{k+1}\rangle$ belonging to D $A_{i_0} \cap A_{i_1} \cap \ldots \cap A_{i_k} \supsetneqq A_{i_0} \cap A_{i_1} \cap \ldots \cap A_{i_k} \cap A_{i_{k+1}}$. Moreover, every element of D has a finite number of successors. By König's lemma the tree D must be finite.

Let I be the set of positive integers occurring in sequences which are elements of D. We shall prove that (5) holds for this I.

Take an arbitrary element x of F^n. Hence, for a certain sequence $\langle -1, i_0\rangle \in D$, $x \in B_{i_0}$. Let us assume that $x \in A_{i_0} \cap A_{i_1} \ldots \cap A_{i_k} \neq \emptyset$. Then from the construction it follows that there exists an element $\langle -1, i_0, \ldots, i_k, i_{k+1}\rangle$ of D such that $x \in B_{i_{k+1}}$. But D has no infinite path, hence there is an $i \in I$ such that $x \in B_i - A_i$. ∎

Let $\mathfrak{F}$ be a field satisfying "finite covering condition". Then by Theorem 1 we have the following corollaries.

COROLLARY 1. *Every program defined on the whole $\mathfrak{F}$ is strongly equivalent to a loop-free program.*

Proof: The halting formula $K1$ of the program K is equivalent to an open formula of finitistic logic. The same reasoning as at the end of § I allows us to find a loop-free program strongly equivalent in $\mathfrak{F}$ to the program K.

COROLLARY 2. *For $\{A_i\}$, $\{B_i\}$, $i \in \omega$, two enumerable sets of algebraic varieties in F^n:*

$$\bigcap_{i\in\omega} (A_i \cup -B_i) \neq \emptyset \textit{ iff for every finite subset } I \subset \omega,\ \bigcap_{i\in I} (A_i \cup -B_i) \neq \emptyset.$$

Proof: Only-if-part is clear. Let us suppose that

$$\bigcap_{i\in\omega} (A_i \cup -B_i) = \emptyset;$$

then by De Morgan's law

$$\bigcup_{i\in\omega} (B_i - A_i) = F^n$$

and by Theorem 1 there exists a finite subset $I \subset \omega$ such that

$$\bigcup_{i\in I} (B_i - A_i) = F^n$$

and again by De Morgan's law

$$\bigcap_{i\in I} (A_i \cup -B_i) = \emptyset.$$

COROLLARY 3. *An enumerable set of open formulas $\{a_i(x_1, \ldots, x_n)\}_{i\in\omega}$ is satisfiable in $\mathfrak{F}$ iff its every finite subset is satisfiable in $\mathfrak{F}$.*

Proof: Immediately from Corollary 2.

COROLLARY 4. *For* $\{\alpha_i(x_1, \ldots, x_n)\}$, $\{\beta_i(x_1, \ldots, x_n)\}$ *two enumerable sets of open formulas*:

$$\mathfrak{F} \models (\exists x_1, \ldots, x_n)\big(\bigvee_{i\in\omega} \alpha_i(x_1, \ldots, x_n) \wedge \neg \bigvee_{i\in\omega} \beta_i(x_1, \ldots, x_n)\big)$$

iff $(\exists k)(\forall m)$

$$\mathfrak{F} \models (\exists x_1, \ldots, x_n)\big(\alpha_k(x_1, \ldots, x_n) \wedge \neg \bigvee_{i\leqslant m} \beta_i(x_1, \ldots, x_n)\big).$$

Proof: Only-if-part is clear. Let us suppose that for some $k \in \omega$ and every $m \in \omega$ there exists a valuation v_m such that

$$(13) \qquad \mathfrak{F} \models \alpha_k[v_m] \quad \text{and} \quad \mathfrak{F} \models \neg\beta_i[v_m] \text{ for } i \leqslant m.$$

From (13) we infer that for every m the set of open formulas

$$\{\alpha_k(x_1, \ldots, x_n),\ \neg\beta_1(x_1, \ldots, x_n), \ldots,\ \neg\beta_m(x_1, \ldots, x_n)\}$$

is satisfiable in $\mathfrak{F}$. By the last Corollary the infinite set

$$\{\alpha_k(x_1, \ldots, x_n),\ \neg\beta_i(x_1, \ldots, x_n),\ i \in \omega\}$$

is satisfiable in $\mathfrak{F}$. ■

Theorem 1 gives the algebraic criterion for elimination of loops from programs which define some total function over a field $\mathfrak{F}$. We shall see that it is not true in every field, for example, in the field $\mathfrak{Q}$ of rationals most of programs have essential loops. But programs which define a total function in every field are equivalent to loop-free programs. In fact, if Φ denotes the set of axioms of fields and $\Phi \models K\mathbf{1}$, then by Theorem 1, § I, there is a loop-free program M such that $\Phi \models K \equiv M$. Now, we shall give examples of programs which are correct in every field.

EXAMPLE 1. At first sight it could seem that the algorithmic evaluation of total functions defined over an arbitrary field is a quite simple problem, because we have merely loop-free programs and four arithmetic operations. But it is not the case, as the following example shows.

Let us recall Strassen's algorithm [27] for the fast matrix multiplication. We consider two 2×2 matrices

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

with items from some noncommutative ring. First we compute 7 products:

$$p_1 := a\cdot e, \quad p_2 := b\cdot g, \quad p_3 := (c+d-a)\cdot(h-f+e),$$
$$p_4 := (c+d)\cdot(f-e), \quad p_5 := (a-c)\cdot(h-f),$$
$$p_6 := (b-c-d+a)\cdot h, \quad p_7 := d\cdot(g-h+f-e).$$

Next in order to compute the whole product:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

we must only perform the following operations:

$$A := p_1 + p_2, \quad B := p_1 + p_3 + p_4 + p_6, \quad C := p_1 + p_3 + p_5 + p_7,$$
$$D := p_1 + p_3 + p_4 + p_5.$$

Now if we have two $n \times n$ matrices over a field we treat them as 2×2 matrices, each of whose elements are $(n/2) \times (n/2)$ submatrices. Then the whole product can be expressed in terms of sums, differences and products of these submatrices. Recursive application of this method gives the algorithm for multiplication of two $n \times n$ matrices with approximate cost:

$$O(n^{\log_2 7}).$$

This algorithm uses a lot of auxiliary memory cells. In fact, every level of recursion requires a declaration of $7+4$ new matrices of an order $(n/2)$ (7 for $p_1, \ldots, p_7$ and 4 for the result). Hence the straightforward application of this method needs at least $11/3\, n^2$ memory cells.

We shall define the algorithm, which computes the same arithmetic expressions as in Strassen's method, but using only input variables: a, b, c, d, e, f, g, h and two auxiliary variables: x, y. The result of multiplication is put on the left or the right argument depending on the value of a control variable k.

Moreover all seven products computed in our algorithm are so constructed that the result is also put on the left or on the right argument of the operation. Hence our algorithm can be applied recursively.

```
begin
  $x := c+d$; $y := f-e$; $c := x-a$; $f := h-y$;
                          $x := x \cdot y$; $y := b-c$; $h := y \cdot h$;
  $y := g-f$; $b := b \cdot g$; $g := d-c$; $d := d \cdot y$;
                          $y := f-e$; $c := c \cdot f$; $g := g \cdot y$;
  $a := a \cdot e$; $c := c+a$; $x := x+c$; $y := g+c$;
  if $k = 0$ then
  begin
    $a := a+b$; $b := x+h$; $c := y+d$; $d := x+g$
  end else
  begin
    $e := a+b$; $f := x+h$; $h := x+g$; $g := y+d$
  end
end
```

The correctness of the above is displayed on the following table:

a	b	c	d	e	f	g	h	x	y
		$c+d-a$			$h-f+e$			$c+d$	$f-e$
								p_4	undef.
									$b-c-d+a$
							p_6		undef.
						undef.			$g-h+f-e$
	p_2								
			p_7			$a-c$			undef.
					undef.				$h-f$
		p_3							
p_1				undef.		p_6			undef.
		p_1+p_3						$p_1+p_3+p_4$	$p_1+p_3+p_5$
A	B	C	D	A	B	C	D		
		$k=0$				$k \neq 0$			

This new modification of Strassen's algorithm gives exactly the same number of arithmetical operations as the original method and moreover it uses only

$$2(n^2/4+n^2/16+\ldots) = 2/3n^2$$

additional memory cells. Really, every level of recursion requires exactly two auxiliary matrices of order $n/2^k$. The whole amount of auxiliary memory cells is therefore defined by the above sum, where the number of terms is bounded by the depth of recursion.

EXAMPLE 2.

```
begin  $a: = a^{-1}$; $x: = a$; $y: = c$; $c: = c \cdot x$; $x: = a$;
    $b: = x \cdot b$; $x: = b$; $y: = y \cdot x$; $d: = y-d$; $d: = d^{-1}$;
    $x: = b$; $y: = d$; $c: = y \cdot c$; $y: = c$; $y: = x \cdot y$;
    $a: = a-y$; $y: = d$; $b: = b \cdot y$; $d: = -d$
end
```

The program realizes Strassen's method of inverting 2×2 matrix [27]. The reader can easily verify that it computes the inverse of a matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

under the same assumption about computing products as in Example 1. Combining both programs we obtain the method of inverting $n\times n$ matrices in $2n^2$ auxiliary cells of memory. It must be pointed out that it applies only to matrices whose submatrices a, d at every level of recursion are nonsingular.

Let us conclude this section with one more remark. Consider a field $\mathfrak{F}$ of characteristic zero. It contains the set of positive integers and we can speak about programmability of recursive or partial recursive functions. Hence in the following theorem we assume a field $\mathfrak{F}$ to be of characteristic zero.

THEOREM 2. *For every partial recursive function $f\colon N \to N$ there exists a program K over a field $\mathfrak{F}$ such that*

$$\mathfrak{F} \models K(y = z)[v] \quad \text{iff} \quad f(n) = m,\ n, m \in N,$$

where $v(x) = n$, $v(z) = m$, x is input and y is output variable of K.

Proof: The existence of K for the following functions:

$$S(x) = x+1, \quad 0(x) = 0, \quad I^n_m(x_1, \ldots, x_n) = x_m, \quad x \cdot y$$

is immediate.

The program

```
begin $i := 0$; while $i \neq x \wedge i \neq y$ do $i := i+1$;
    if $i = x$ then $p := 0$ else $p := 1$
end
```

computes the characteristic function of the relation $x \leqslant y$.

Now, the programs:

```
if $x \geqslant y$ then $z := x-y$ else $z := 0$
```

and

```
begin $i := 0$; while $(i+1)^2 \leqslant x$ do $i := i+1$ end
```

compute $x \dot{-} y$ and $[\sqrt{x}]$, respectively.

It remains to prove that the superposition and minimum are programmable in $\mathfrak{F}$. But the superposition of programs computes the superposition of functions and the program (while $j \neq 0$ do)

```
begin $i := 0$; $K$; begin $i := i+1$; $K$ end end
```

computes $f(x) = (\mu i)\big(g(i, x) = 0\big)$ provided K computes $g(i, x)$ on the output variable j. ■

III. Programmability in the field of complex numbers

Let $\mathfrak{C} \overset{\text{df}}{=} \langle C, +, -, \cdot, ^{-1}, 0, 1, = \rangle$ be the data structure of complex numbers. We shall investigate algorithmic properties and programmable functions over this system. First of all we shall prove that C satisfies "finite covering condition" (Def. 1, § II).

THEOREM 1 (T. Mostowski). *Let A_i, $i \in \omega$, and A be algebraic varieties in C^n.*

If

(1) $$A \subset \bigcup_{i \in \omega} A_i,$$

then there exists a finite subset $I \subset \omega$ such that

(2) $$A \subset \bigcup_{i \in I} A_i.$$

Proof: By Whitney's theorem [28], A has a finite number of topological components. So, it suffices to show (2) for a topological component B of A. From (1) we infer that

$$B = \bigcup_{i \in \omega} (B \cap A_i). \tag{3}$$

Since every component of an algebraic variety in the field of complex numbers is a locally compact space, Baire's theorem is valid for B. From (3) it follows that there exists $i_0 \in \omega$ such that $B \cap A_{i_0}$ contains a nonempty open subset

$$\emptyset \neq G \underset{\text{open}}{\subset} B \cap A_{i_0}. \tag{4}$$

But by the theory of analytical functions, if a polynomial p vanishes on G, then it must vanish on the whole topological component B. So, by (4) we obtain

$$\mathrm{Id}(B \cap A_{i_0}) \subset \mathrm{Id}(B), \tag{5}$$

where $\mathrm{Id}(A)$ for arbitrary subset $A \subset C^n$ denotes the ideal of polynomials vanishing on A. On the other hand, the set

$$\mathrm{Cl}(A) = \{x \in C^n\colon \text{ for every } f \in \mathrm{Id}(A), f(x) = 0\}$$

is an algebraic variety called the *algebraic closure* of A. If $\mathrm{Id}(A_1) \subset \mathrm{Id}(A_2)$, then of course $\mathrm{Cl}(A_1) \supset \mathrm{Cl}(A_2)$. Thus, we obtained the following chain of inclusions:

$$B \subset \mathrm{Cl}(B) \subset \mathrm{Cl}(B \cap A_{i_0}) \subset \mathrm{Cl}(A_{i_0}) = A_{i_0},$$

where the second of them follows from (5) and the remaining just from the properties of algebraic closure. We have proved that $B \subset A_{i_0}$. ∎

Remark: This theorem remains valid also in the case of real numbers, since Whitney's theorem, as well as the other facts used in the proof, are true in the field of reals.

THEOREM 2. *There is an effective method of transforming every program K for which $\mathfrak{C} \models K1$ into a loop-free program M such that*

$$\mathfrak{C} \models K \equiv M.$$

Proof: We recall that the first order theory of algebraically closed fields of characteristic zero is recursive, and moreover, it is exactly the same as the first order theory of complex numbers. In § I we showed that if the semantical consequence is for open formulas recursively enumerable and loops are eliminable, then they are effectively eliminable. ∎

THEOREM 3. *The set of algorithmic properties valid in the field $\mathfrak{C}$ is a Π_2^0-complete set.*

Proof: Let φ be an algorithmic property. By Lemma 3, § I, we can consider the case of a primitive formula:

$$\bigvee_{i\in\omega} a_i(x_1, \ldots, x_n) \Rightarrow \bigvee_{i\in\omega} \beta\ (x_1, \ldots, x_n),$$

where a_i, β_i are open formulas effectively given.

We have the following equivalences:

$$\mathfrak{C} \text{ non} \models (\forall x_1, \ldots, x_n)\big(\bigvee_{i\in\omega} a_i(x_1, \ldots, x_n) \Rightarrow \bigvee_{i\in\omega} \beta_i(x_1, \ldots, x_n)\big)$$

iff

$$\mathfrak{C} \models (\exists x_1, \ldots, x_n)\big(\bigvee_{i\in\omega} a_i(x_1, \ldots, x_n) \wedge \neg \bigvee_{i\in\omega} \beta_i(x_1, \ldots, x_n)\big)$$

iff (by Corollary 4 of Theorem 1, § II)

$$(\exists k)(\forall m)\ \mathfrak{C} \models (\exists x_1, \ldots, x_n)\big(a_k(x_1, \ldots, x_n) \wedge \neg \bigvee_{i\leqslant m} \beta_i(x_1, \ldots, x_n)\big).$$

The above proves that $\{\varphi\colon \varphi \text{ is algorithmic} \wedge \mathfrak{C} \models \varphi\}$ is Π_2^0.

By Theorem 2, § II, for every partial recursive function $f\colon N \to N$ there exists a program K over $\mathfrak{C}$ such that considering only positive integers it computes f. This theorem was proved effectively, hence we can consider the program K_i which is effectively obtained from the partial recursive function of index i.

Let M be the program:

```
begin  $y := 0$;
  while  $y \neq x$ do $y := y+1$
end
```

It is clear that the domain of M is the set of positive integers. Now, the relation

$$\mathfrak{C} \models M1 \Leftrightarrow K_i 1$$

holds iff K computes a total recursive function. This last relation is known to be a Π_2^0-complete property [23]. ■

THEOREM 4. *Let χ denote the set of axioms of fields of characteristic zero Then for every algorithmic property φ*

$$\mathfrak{C} \models \varphi \quad \textit{iff} \quad \chi \models \varphi.$$

Proof: If-part follows from the fact that $\mathfrak{C}$ has characteristic zero. Now, suppose that for a field $\mathfrak{F}$ of characteristic zero $\mathfrak{F}$ non $\models \varphi$. We can consider φ to be a primitive formula. So,

$$(6) \qquad \mathfrak{F} \models (\exists x_1, \ldots, x_n)\big(\bigvee_{i\in\omega} a_i(x_1, \ldots, x_n) \wedge \bigwedge_{i\in\omega} \neg\beta_i(x_1, \ldots, x_n)\big).$$

Let $\mathfrak{F}'$ be the algebraic closure of $\mathfrak{F}$; hence (6) holds for $\mathfrak{F}'$ due to the existential form of the above formula. There exists a $k \in \omega$ such that a set $\{\alpha_k, \neg\beta_i, i \in \omega\}$ is satisfiable in $\mathfrak{F}'$. So, its every finite subset is satisfiable in $\mathfrak{F}'$, and by algebra it is also satisfiable in $\mathfrak{C}$. If every finite subset of some enumerable set of open formulas is satisfiable in $\mathfrak{C}$, then by Corollary 3 to Theorem 1, § II, $\{\alpha_k, \neg\beta_i, i \in \omega\}$ is satisfiable in $\mathfrak{C}$. This proves that $\mathfrak{C}$ non $\models \varphi$. ■

Remark: Theorem 2 was proved for the first time by Kfoury [14], but he did not know Theorem 4. He used the fact that the first order theory of $\mathfrak{C}$ is $\aleph_1$-categorical. It is clear that his method is useless in the case of $\mathfrak{R}$, because the first order theory of $\mathfrak{R}$ is not categorical in any power. Moreover, he did not give the axiomatization of algorithmic properties over $\mathfrak{C}$.

Let us make the following observations concerning algorithmic properties in $\mathfrak{C}$. From the form of an algorithmic property φ it follows that if $\mathfrak{C} \models \varphi$, then for every subfield $\mathfrak{F}$ of $\mathfrak{C}$, $\mathfrak{F} \models \varphi$, because φ is universal. But by Theorem 4 φ is also valid in any field $\mathfrak{F}$, not necessarily a subfield of $\mathfrak{C}$, of characteristic zero.

On the other hand, consider an algorithmic property which transformed into the normal form has no negative occurrences of loops, i.e., such that its every primitive formula is the following:

$$\alpha \Rightarrow K\beta,$$

where α, β are open formulas. Then for the program M:

```
begin  if $\alpha$ then
       begin $K$;
         while $\neg\beta$ do
       end
end
```

we have

$$\mathfrak{C} \models \alpha \Rightarrow K\beta \quad \text{iff} \quad \mathfrak{C} \models M\mathbf{1}.$$

According to Theorem 2 the formula $\alpha \Rightarrow K\beta$ is equivalent to an open formula. We proved that loop-positive algorithmic properties are semantically equivalent to elementary formulas. By Theorem 4 and Robinson's theorem (see [22]), if $\mathfrak{C} \models \varphi$, then there exists a prime number p such that for all $p' \geqslant p$, $\mathfrak{F}_{p'} \models \varphi$, where $\mathfrak{F}_{p'}$ denotes arbitrary field of characteristic p'.

Now, let us consider some examples of programs in $\mathfrak{C}$.

EXAMPLE 1.

$$\textbf{begin while } x^2 = -1 \textbf{ do } x := x \textbf{ end}$$

The domain of this program is $C-\{\sqrt{-1}, -\sqrt{-1}\}$, and its halting formula is equivalent to the open formula $x^2 \neq -1$. Notice that this program is total in $\mathfrak{R}$.

EXAMPLE 2. The program K

$$\textbf{begin}\ x:=1;\ \textbf{while}\ x\neq 0\ \textbf{do}\ x:=x+1\ \textbf{end}$$

has only one variable x which is neither input nor output variable. The algorithmic property

$$\neg K1$$

is equivalent to χ–the axioms of field of characteristic zero. This example shows that we cannot reduce the set of axioms mentioned in Theorem 4.

EXAMPLE 3.

```
begin  $i:=1$; $y:=x$;
       while $y \neq 1$ ~~then~~ do
       begin
              $i:=i+1$; $y:=y\cdot x$
       end;
       $z:=x^{-1}$
end
```

The program has one input variable x, two output variables i, z and one program variable y. Its domain is the set of all roots of the unit. In the loop, y is set to $y\cdot x$ until $y\neq 1$. On the output i is the degree of x, i.e. $x^i=1$ and z is the conjugate $\bar{x}$ of this root x.

EXAMPLE 4.

```
begin $c:=(w_3-w_2)\cdot(z_3-z_1)$;
  if $c \neq 0$ then
  begin  $c:=(w_3-w_1)\cdot(z_3-z_2)\cdot c^{-1}$; $d:=c\cdot z_1$; $b:=w_2\cdot d-w_1\cdot z_2$;
         $d:=d-z_2$; $a:=w_1-w_2\cdot c$; $c:=1-c$;
    if $a\cdot d \neq b\cdot c$ then
    begin
      if $c=0$ then
      begin
        if $d=a$ then
        begin
          if $b=0$ then begin $k:=3$; $m:=1$ end
          else begin $k:=0$; m: $=1$ end
        end else begin $k:=1$; $m:=1$ end
      end else
      begin $a:=a-d$;
```

```
                if $a^2 \neq 4\cdot b\cdot c$ then begin $k := 2$; $m := 0$ end
                else begin $k := 1$; $m := 0$ end
                end
            end else $k := -1$
        end else $k := -1$
end
```

This program is much more complicated. It has six input variables $w_1, w_2, w_3, z_1, z_2, z_3$ and six output variables a, b, c, d, k, m. Let us imagine two circles (proper or improper) on the complex plane C, the first of which is defined by three points w_1, w_2, w_3 and the latter by another three z_1, z_2, z_3. We want to find a homographic function:

$$w(z) = \frac{a\cdot z+b}{c\cdot z+d}$$

which transforms the second circle onto the first one. It is known that such a transformation is defined by the equation:

$$\frac{w-w_1}{w-w_2} : \frac{w_3-w_1}{w_3-w_2} = \frac{z-z_1}{z-z_2} : \frac{z_3-z_1}{z_3-z_2}.$$

Moreover, we want to compute the number of fixed points of this homographic transformation. The problem consists in the analysis of the following equation:

$$c\cdot z^2-(a-d)\cdot z-b = 0.$$

At the exit of the program a, b, c, d are coefficients of $w(z)$, m is 1 if $w(z)$ has a fixed point at infinity, otherwise m is 0. Finally, k is $0, 1$ or 2 if $w(z)$ has k normal fixed points or $k = 3$ if $w(z) = z$ is the identity transformation or $k = -1$ if a transformation does not exist.

EXAMPLE 5. Now, we show how to use the theorems of this section to prove nonprogrammability of some functions and predicates over $\mathfrak{C}$. First, let us consider the predicate $r(z)$—z is real. Is it strongly programmable over $\mathfrak{C}$? If it were, there would exist a program $K(z; x)$ defined on the whole $\mathfrak{C}$ such that, for the output variable x, $x = 0$ iff z is real, otherwise $x = 1$. By Theorem 2 we can assume K to be a loop-free program. Hence the formula $K(z; x)(x = 0)$ would be equivalent to an open formula $a(z)$ such that $\mathfrak{C} \models r(z) \Leftrightarrow a(z)$. Atomic formulas of one variable define only finite sets of complex numbers, so $a(z)$ defines a Boolean combination of finite sets which is finite or cofinite set. Hence it is clear that $a(z)$ cannot define the straight of reals which is neither finite nor cofinite in $\mathfrak{C}$.

Now, consider the following functions:
$\mathrm{re}(z)$ — the real part of z,
$\mathrm{im}(z)$ — the imaginary part of z,
$\bar{z}$ — the conjugate of z,
$|z|$ — the modulus of z.

We have $\mathfrak{C} \models r(z) \Leftrightarrow \mathrm{re}(z) = z$, which shows that $\mathrm{re}(z)$ is not programmable in $\mathfrak{C}$. Next, the programs:

$$u := (z+\bar{z})/2 \quad (\text{computes } \mathrm{re}(z)),$$
$$u := \mathrm{im}(z^2)\cdot(\mathrm{im}(z))^{-1}/2 \quad (\text{computes } \mathrm{re}(z)),$$
$$u := z^{-1}\cdot|z|^2 \quad (\text{computes } \bar{z}),$$

show that none of $\mathrm{im}(z)$, $\bar{z}$, $|z|$ is programmable in $\mathfrak{C}$.

EXAMPLE 6. The problem of programmability over $\mathfrak{C}$ is closely connected with the problem of solvability by radicals. We recall that the radical $\sqrt[n]{z}$ can be treated as a many-valued function which gives for z all solutions of the equation $x^n = z$. First of all, notice that $\sqrt[n]{z}$ is not programmable over $\mathfrak{C}$ for any $n \geqslant 2$. This follows from the fact that algorithmic properties valid in $\mathfrak{C}$ remain valid in $\mathfrak{Q}$ — the field of rationals. The programs over any field transform rational arguments to rational results, hence radicals are not programmable in $\mathfrak{Q}$, which implies that they are not programmable in $\mathfrak{C}$.

So, the radicals augment our primary computational possibilities. If we introduce them to the relational system $\mathfrak{C}$, shall we obtain a class of algorithmically solvable problems much wider than the original one? In particular, we can ask if loops are essential in that new data structure. Looking back at Theorem 1, § II, we find that A_i, B_i ought to be algebraic varieties, nothing more. But the formula

$$f(x_1, \ldots, x_n) = 0,$$

where f is a polynomial and x_i is the radical $\sqrt[n_i]{z_i}$, defines also an algebraic variety in $\mathfrak{C}$, because it is the meet of $\{z \in C^n: f(z) = 0\}$ and $\{(x_i, z_i): z_i^{n_i} = x_i\}$, $i = 1, \ldots, n$.

Hence any program which halts on all initial values of input variables in the field $\mathfrak{C}$ of complex numbers with radicals is strongly equivalent to a loop-free program.

Let us denote this data structure by $\mathfrak{C}\mathfrak{R}$ad. The most important result of nonprogrammable function over $\mathfrak{C}\mathfrak{R}$ad is supplied by the famous Abel–Ruffini's theorem about unsolvability of algebraic equations of degree $\geqslant 5$ by radicals. Moreover, by the same argument as in the case of $\mathfrak{C}$ we can prove that such relations and functions as $\mathrm{re}(z)$, $r(z)$, $\mathrm{im}(z)$,

$\bar{z}$, $|z|$ are not programmable over $\mathfrak{CRad}$. However, there are some functions programmable over $\mathfrak{CRad}$ which are interesting. For example, the well-known algorithms for solution of quadratic, cubic, some symmetric equations and so on are programmable over $\mathfrak{CRad}$ without being programmable over $\mathfrak{C}$.

IV. Programmability in the field of real numbers

Let $\mathfrak{R} \overset{\mathrm{df}}{=} \langle R, +, -, \cdot, ^{-1}, 0, 1, =\rangle$ be the data structure of real numbers. In § III, Theorem 1, we proved "finite covering condition" for the field of complex numbers. As it was mentioned in the remark, the proof remains valid in the case of R. So, we can repeat Theorems 1–3 *mutatis mutandis*.

THEOREM 1. *The field R of real numbers satisfies "finite covering condition".*

THEOREM 2. *There is an effective method of transforming every program K for which $\mathfrak{R} \models K1$ into a loop-free program M such that*

$$\mathfrak{R} \models K \equiv M.$$

THEOREM 3. *The set of algorithmic properties valid in the field $\mathfrak{R}$ is a Π_2^0-complete set.*

In what follows we shall try to find, as in the case of $\mathfrak{C}$, a set of axioms which characterizes all algorithmic properties of $\mathfrak{R}$. It will turn out that in this case the axioms are also of the first order.

DEFINITION 1 (Artin [13]). A field $\mathfrak{F}$ is called *formally real* iff the only relations of the form $x_1^2+\ldots+x_n^2 = 0$ in $\mathfrak{F}$ are those for which $x_1 = x_2 = \ldots = x_n = 0$.

It is immediate that $\mathfrak{F}$ is formally real iff -1 is not a sum of squares of elements of F. If the characteristic of $\mathfrak{F}$ is $p \neq 0$, then $0 = 1^2+\ldots+1^2$ (p terms), hence it is clear that formally real fields are necessarily of characteristic 0.

Let us denote by $\mathscr{P}$ the axioms of formally real fields, i.e. the axioms Φ of fields and the scheme of axioms:

$$(1) \qquad (\forall x_1, \ldots, x_n)(x_1^2+\ldots+x_n^2 \neq -1) \qquad n \geqslant 1.$$

THEOREM 4. *For every algorithmic property φ*

$$\mathfrak{R} \models \varphi \quad \textit{iff} \quad \mathscr{P} \models \varphi.$$

Proof: If-part follows from the fact that $\mathfrak{R}$ is formally real.

Suppose that for a primitive formula φ and a formally real field $\mathfrak{F}$ we have $\mathfrak{F}$ non $\models \varphi$. Then

$$(2) \qquad \mathfrak{F} \models (\exists x_1, \ldots, x_n)\Big(\bigvee_{i\in\omega} a_i(x_1, \ldots, x_n) \wedge \bigwedge_{i\in\omega} \neg\beta_i(x_1, \ldots, x_n)\Big).$$

Let $\mathfrak{F}'$ be the real closure of $\mathfrak{F}'$ (see [13]). (2) is true for $\mathfrak{F}'$. There exists $k \in \omega$ such that the set $\{\alpha_k, \neg\beta_i, i \in \omega\}$ is satisfiable in $\mathfrak{F}'$, hence every finite subset of it is satisfiable in $\mathfrak{F}'$. By Tarski's theorem [13] and by Corollary 3 to Theorem 1, § II, this set is satisfiable in $\mathfrak{R}$, which proves that (2) is also valid for $\mathfrak{R}$. ■

Now, as in the case of $\mathfrak{C}$, we can consider two classes of algorithmic properties. Those which have negative occurrences of loops and are not equivalent to elementary formulas and those loop-positive ones which are equivalent to elementary formulas.

From Theorem 4 it follows that for every algorithmic property φ, if it is valid in $\mathfrak{R}$, then it is valid in any formally real field $\mathfrak{F}\mathfrak{R}$. In particular, all subfields of $\mathfrak{R}$ satisfy this condition.

Let us denote by $\mathfrak{F}\mathfrak{R}_p$ a field which satisfies axioms (1) for $n = 1, 2, \ldots, p$. Notice that there are fields of characteristic zero which are $\mathfrak{F}\mathfrak{R}_p$ not being $\mathfrak{F}\mathfrak{R}_n$ for $n > p$. For example, the field:

$$\{a+b\sqrt{-2}:\ a, b \text{ — rationals}\}$$

is $\mathfrak{F}\mathfrak{R}_1$ and is not $\mathfrak{F}\mathfrak{R}_2$. Indeed, $(a+b\sqrt{-2})^2 = a^2-2b^2+2ab\sqrt{-2}$. If $a^2-2b^2+2ab\sqrt{-2} = -1$, then $a = 0$ or $b = 0$. The first implies $b^2 = 1/2$, the latter $a^2 = -1$. Hence this field is $\mathfrak{F}\mathfrak{R}_1$. On the other hand, $(0+\sqrt{-2})^2+(1+0\cdot\sqrt{-2})^2 = -1$, which proves that it is not $\mathfrak{F}\mathfrak{R}_2$.

Now, let φ be a loop-positive algorithmic property. By Theorem 4 and by the Compactness theorem for the first order logic, if $\mathfrak{R} \models \varphi$, then there exists n such that for $n' \geqslant n$, $\mathfrak{F}\mathfrak{R}_{n'} \models \varphi$.

EXAMPLE 1.

begin while $x^2 = -1$ **do** $x := x$ **end**

The domain of this program is the whole field R and it shows that the fields $\mathfrak{R}$ and $\mathfrak{C}$ are not algorithmically equivalent.

EXAMPLE 2.

begin $x := 1$; **while** $x \neq 0$ **do** $x := x+1$ **end**

Negation of the halting formula of the above program is equivalent to χ — the scheme of axioms of fields of characteristic zero. As in the case of $\mathfrak{C}$, it is valid in $\mathfrak{R}$.

EXAMPLE 3.

```
begin
    $e := 2\cdot b\cdot d\cdot e - a\cdot e^2 - d^2\cdot c$;
    $a := a\cdot c - b^2$; $e := e + f\cdot a$;
    if $e \neq 0$ then $k := 1$ else if $a \neq 0$ then $k := 0$ else $k := 2$
end
```

The program computes the number of centers of symmetry of the algebraic figure defined by the equation:

$$a\cdot x^2+2\cdot b\cdot x\cdot y+c\cdot y^2+2\cdot d\cdot x+2\cdot e\cdot y+f = 0.$$

Input variables are a, b, c, d, e, f, output is k. If $k = 1, 0$, then the figure has k centers of symmetry, otherwise it has an infinite number of them.

EXAMPLE 4 ([1]).

```
begin
  $x := (a+b)\cdot c$   $y := a\cdot(d-c)+x$   $x := x-b\cdot(d+c)$
end
```

Let a, b, c, d be input variables and x, y output variables of our program. It is easily shown that $y = a\cdot d+b\cdot c$ and $x = a\cdot c-b\cdot d$ and, of course, y, x are the real and the imaginary part of $(a+ib)\cdot(c+id)$.

This program computes the product of two complex numbers using only three real multiplications. In what follows we shall repeat the proof [1] of the fact that for evaluating the product of two complex numbers three multiplications in $\mathfrak{R}$ are necessary. We shall base on the following:

THEOREM 5. *Let M be an $r\times p$ matrix with elements from the ring $F[a_1, \ldots, a_n]$ of polynomials of $a_1, \ldots, a_n$ with coefficients in a field F. Suppose that for any two vectors u, v with elements from F, uMv belongs to F iff either $u = 0$ or $v = 0$. Then any computation of Mx, where $x = [x_1, \ldots, x_r]$, requires at least $r+p-1$ multiplications.*

Now, the computation of $(a+ib)\cdot(c+id)$ can be represented as follows:

$$\begin{bmatrix} a & -b \\ b & a \end{bmatrix}\cdot\begin{bmatrix} c \\ d \end{bmatrix}.$$

So, it suffices to prove that if the product

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\cdot\begin{bmatrix} a & -b \\ b & a \end{bmatrix}\cdot\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

is an element of F, then for $y_1, y_2, z_1, z_2 \in F$ either $y_1 = y_2 = 0$ or $z_1 = z_2 = 0$. Since we consider the field $\mathfrak{R}$, we can take as F an arbitrary formally real field FR. The above product is equal to

$$y_1\cdot z_1\cdot a+y_2\cdot z_1\cdot b+y_2\cdot z_2\cdot a-y_1\cdot z_2\cdot b.$$

Thus the coefficients of a and b must be zero, i.e.

$$y_1\cdot z_1+y_2\cdot z_2 = 0 \quad \text{and} \quad y_2\cdot z_1-y_1\cdot z_2 = 0.$$

Suppose $y_1 \neq 0$. Then $z_2 = y_2 \cdot z_1 / y_1$ and substituting this into the second equation we have

$$(y_1^2 + y_2^2) \cdot z_1 = 0.$$

Since $y_1 \neq 0$, we have in the formally real field FR $y_1^2 + y_2^2 \neq 0$. Hence $z_1 = 0$, which immediately implies that $z_2 = 0$.

Now assume $y_1 = 0$. If $y_2 = 0$, the proof is finished. If $y_2 \neq 0$, then we interchange the roles of y_1 and y_2 and we obtain again $z_1 = z_2 = 0$.

The above proof is very close to the original one, but here we emphasize the role of the axiomatization. The facts which are essential and can be taken into consideration in proofs of algorithmic properties in the field $\mathfrak{R}$, are only the axioms of formally real fields.

Example 5. Consider the following functions and relations:

$$x > y, \quad x > 0, \quad x \geqslant y,$$

$$abs(x), \quad sign(x), \quad entier(x).$$

We shall prove that all of them are nonprogrammable in $\mathfrak{R}$. First we prove that $x > 0$ is not programmable. By Theorem 2, if $x > 0$ is programmable in $\mathfrak{R}$, then there exists an open formula $a(x)$ such that

$$\mathfrak{R} \models (\forall x)\big(x > 0 \Leftrightarrow a(x)\big).$$

As in the case of $\mathfrak{C}$, $a(x)$ defines in $\mathfrak{R}$ only finite and cofinite sets. But $\{x: x > 0\}$ is neither finite nor cofinite.

Now, if $x > y$ or $x \geqslant y$ were programmable, then of course $x > 0$ would be programmable. Next,

$$x \geqslant 0 \quad \text{iff} \quad abs(x) = x,$$

hence *abs* is also a nonprogrammable function.

Finally, consider the following programs:

```
begin y: = sign(x);
    if y = 0 ∨ y = 1 then y: = x else y: = −x
end
begin y: = entier(x);
  if x ≠ 0 then
  begin i: = 0; while i = y ∨ i = −y do i: = i+1;
      if i = y then y: = 1 else y: = −1
  end else y: = 0
end
```

The first computes *abs*, the latter *sign*, hence neither *sign* nor *entier* is programmable in $\mathfrak{R}$.

EXAMPLE 6. Consider the field $\mathfrak{R}$ of reals with one additional operation: $\sqrt{x^2+y^2}$, i.e. extracting the positive square root of the sum of two squares.

This system gives the algorithmic basis for the classical geometrical constructions. From the fact that

$$x^2+y^2 = z^2$$

is an algebraic variety in R, it follows that loops are eliminable from programs over this system. Every construction is definable by some finite open formula.

V. Programmability in the ordered field of reals

Let $\mathfrak{RO} \overset{\mathrm{df}}{=} \langle R, +, -, \cdot, ^{-1}, 0, 1, =, < \rangle$ be the data structure of the ordered field of reals. The axiomatization of algorithmic properties in $\mathfrak{RO}$ was given by E. Engeler [7]. Let us denote by Ω the axioms of Archimedean ordered fields, i.e., the axioms of ordered fields which are of the first order and the following axiom of Archimedes:

$$\Omega_0\colon\ (\forall xy)\big(x > 0 \wedge y > 0 \Rightarrow (z := y)\textstyle\bigcup(z := z+y)(x < z)\big).$$

THEOREM 1. *For every algorithmic property* φ:

$$\mathfrak{RO} \models \varphi \quad \textit{iff} \quad \Omega \models \varphi.$$

Proof: The implication from right to left is immediate. Conversely, suppose that $\mathfrak{RO} \models \varphi$ and that $\mathfrak{FO}$ is any Archimedean ordered field. By algebra, [13], $\mathfrak{FO}$ is embeddable in $\mathfrak{RO}$. Hence, φ being a universal formula, it is also valid in $\mathfrak{FO}$. ∎

The first question concerning programmability in $\mathfrak{RO}$ is immediate. Does a theorem on the elimination of loops hold in $\mathfrak{RO}$? The answer is negative. Loops cannot be eliminated from programs over $\mathfrak{RO}$. Namely, consider the predicate: x is a positive integer. This property is strongly programmable in $\mathfrak{RO}$:

$$\mathfrak{RO} \models \textbf{begin } i := 0;\ \textbf{while } i < x \textbf{ do } i := i+1 \textbf{ end } (i = x)$$

iff x is a positive integer.

Now, we can easily show that the above program is not equivalent to any loop-free program. In fact, if it were, then there would be an open formula $\alpha(x)$ defining in $\mathfrak{RO}$ the set of positive integers, which is well known to be impossible (see Tarski [13]).

By Theorem 1, § I, the set of algorithmic properties valid in $\mathfrak{RO}$ has no first order characterization. This shows that the axiom of Archimedes is not equivalent to any set of first order formulas (A. Robinson [22]).

An important property of Archimedean ordered fields examined already in ancient Greece is the so-called exhaustion rule. It comes from Eudoxos and was used by him and Archimedes to prove quite precisely many facts from integral calculus. Before we present the rule formally, let us recall how it was formulated in *Elementa* of Euclid. Consider two quantities x and y, $x > y$. If we subtract from x more than its half, from the difference more than its half and so on, after a finite number of steps we shall obtain the difference less than y.

Suppose for a moment that x is the area of a certain surface. If $x_0, x_1, \ldots$ is an increasing sequence of figures with known areas such that $x - x_i < x/2^i$, then by the exhaustion rule the difference $x - x_i$ becomes arbitrarily small.

From the presentday point of view the exhaustion rule actually says that a sequence $\{x_i\}$ of nonnegative quantities for which $x_i < x/2^i$ converges to zero. In what follows we shall try to underline the role of this rule in the analysis of programmability in $\mathfrak{RO}$.

First of all we characterize the exhaustion rule in terms of algorithmic logic.

Let K be a program and let x be a unique input-output variable of K. If K does not change values of variables t, u, v, z, then $H(K)$:

H(K):
$$(\forall tuy)\big(y > 0 \wedge t > 0 \wedge u > 0 \Rightarrow$$
$$\big((x: = t; z: = y)\bigcap(K; z: = z/2)(0 \leqslant x \leqslant z) \Rightarrow (x: = t)\bigcup K(x < u)\big)\big)$$

is the algorithmic version of the exhaustion rule.

Theorem 2 (Euclid [10]). *For any ordered field* $\mathfrak{F}$

$$\mathfrak{F} \models \Omega_0 \quad \textit{iff} \quad \mathfrak{F} \models H(K) \textit{ for every program } K.$$

Proof: Let us suppose that $\mathfrak{F}$ is Archimedean. We define a sequence $\{x_i\}$ by induction. $x_0 = t > 0$ and $x_{i+1} = K(x_i)$. If $y > 0$, then $0 \leqslant x_i \leqslant y/2^i$. But if $y > 0$ and $u > 0$, then from the assumption that $\mathfrak{F}$ is Archimedean follows $y < u \cdot m$ for a natural m. From the axioms of ordered fields we get $x_m \cdot m \leqslant x_m \cdot 2^m$, because $x_m \geqslant 0$. So $x_m \cdot m \leqslant x_m \cdot 2^m \leqslant y < u \cdot m$ and again by properties of ordered fields $x_m < u$.

Now let us suppose that $\mathfrak{F} \models H(K)$ for every program K. Let K be the program:

$$x := x/2.$$

Consider $y > 0$ and $u > 0$. For $t = y$, by assumption, there exists a natural m such that $y < 2^m \cdot u$. Hence, for $n = 2^m$, $y < u \cdot n$, which proves that $\mathfrak{F}$ is Archimedean. ■

Corollary. *Let us denote by H the scheme of axioms of the form $H(K)$. Then for every algorithmic property φ*

$$\mathfrak{RO} \models \varphi \quad \textit{iff} \quad H \models \varphi.$$

Proof follows immediately from Theorem 2.

EXAMPLE 1.

begin if $x \geqslant 0$ **then** $y := x$ **else** $y := -x$ **end**

This program computes the function *abs* over $\mathfrak{RD}$.

EXAMPLE 2.

```
begin $i := 0$;
  if $x \geqslant 0$ then
      while $i+1 \leqslant x$ do $i := i+1$
  else
      while $i > x$ do $i := i-1$
end
```

The above program computes *entier*.

EXAMPLE 3.

```
begin
    $x := (a+1)/2$
  while $x - a/x \geqslant eps$ do $x := (x+a/x)/2$
end
```

This program for $a > 0$ and $eps > 0$ computes the square root of a with the accuracy eps. We shall prove this fact formally in order to emphasize the role played by the exhaustion rule.

First of all, notice that the above program is equivalent to:

begin $x := eps$ **while** $x \geqslant eps$ **do** M **end**

where M is the loop-free program:

```
begin
  if $x = eps$ then $w := (a+1)/2$ else $w := (w+a/w)/2$;
  $x := w - a/w$
end
```

Here w plays the role of x in the original program and x is equal to the obtained accuracy in every step of the loop, except the first step, of course. We must show that the demanded accuracy eps can be always achieved. By Corollary it is sufficient to show that for certain $y > 0$ and $t = u = eps$:

(1) $$(x := eps; z := y) \bigcap (M; z := z/2) \quad (0 \leqslant x \leqslant z).$$

Then by the exhaustion rule we shall have immediately:

(2) $$(x := eps) \bigcup M (x < eps).$$

So, it remains to prove (1). Set to y the value $a + eps + 1 > 0$.

Further steps of the proof carry as follows:

(3) $$(x := eps;\ z := a + eps + 1) \quad (0 \leqslant x \leqslant z)$$

(immediately from the properties of substitutions),

(4) $\big((a+1)/2\big)^2 \geqslant a \wedge 0 \leqslant (a-1)^2/2/(a+1) \leqslant (a+eps+1)/2$

(from the axioms of ordered fields),

(5) $(x:=eps;\ z:=a+eps+1)(M;\ z:=z/2)$

$$(w^2 \geqslant a \wedge 0 \leqslant x \leqslant z \wedge x = w - a/w)$$

(from (3), (4) and the properties of the program M),

(6) $\big(w^2 \geqslant a \wedge 0 \leqslant (w-a/w) \leqslant z\big)$

$$\Rightarrow \big((w+a/w)/2\big)^2 \geqslant a \wedge 0 \leqslant (w+a/w)/2 - a/(w+a/w)/2 \leqslant z/2\big)$$

(from the axioms of ordered fields),

(7) $(w^2 \geqslant a \wedge 0 \leqslant x \leqslant z \wedge x = w - a/w)$

$$\Rightarrow (M;\ z:=z/2.)(w^2 \geqslant a \wedge 0 \leqslant x \leqslant z \wedge x = w - a/w)$$

(from (6) and the properties of M),

(8) for every natural $i \in \omega$

$$(x:=eps;\ z:=a+eps+1)(M;\ z:=z/2)^i(0 \leqslant x \leqslant z)$$

(By induction on i. For $i=0$ this is (3). For $i=1$ this holds by (5). The inductive step now follows from (7)),

(9) $(x:=eps;\ z:=a+eps+1)\bigcap(M;\ z:=z/2)(0 \leqslant x \leqslant z)$

(Immediately from (8) and the definition of the iterational quantifier $\bigcap$).
Finally, by (2),

(10) $H \models a > 0 \wedge eps > 0 \Rightarrow (x:=eps)\bigcup M(x < eps).$

By analogous but more sophisticated methods one can prove the correctness of many numerical algorithms in $\mathfrak{RD}$ as, for example, the bisection strategy for evaluating zeroes of functions or Simpson's algorithm for integration. Moreover, by Corollary to Theorem 2, all algorithmic properties valid in $\mathfrak{RD}$ are semantical consequences and by the completeness theorem they are also syntactical consequences of axioms of ordered fields and the exhaustion scheme. In numerical practice the general idea is based on the construction of an algorithmically computable sequence $\{x_n\}$ which converges to the wanted quantity x. Then x can be computed with arbitrarily high precision.

The analysis of the correctness formula (10) indicates that the sequence $\{x_n\}$ need not converge, it is completely sufficient if x is a cluster point of $\{x_n\}$. In other words, if for every $eps > 0$ there is x_n such that $abs(x-x_n) < eps$.

EXAMPLE 4. We end this section by giving some examples of non-programmable functions in $\mathfrak{RO}$. Let us consider

$$f(x) = \begin{cases} \sqrt{x} & x \geqslant 0, \\ 0 & x < 0. \end{cases}$$

If f were programmable, then there would exist a program $K(x; y)$ such that

$$\mathfrak{RO} \models x > 0 \Rightarrow K(y^2 = x).$$

This formula must be also valid in every Archimedean field, so for the field $\mathfrak{Q}$ of rationals and $x = 2$ we come to a contradiction.

This argument shows that every function which transforms rationals to irrationals or vice versa is not programmable in $\mathfrak{RO}$, and this concerns almost all standard functions.

VI. Programmability in the field of rationals

Let $\mathfrak{Q} \overset{\mathrm{df}}{=} \langle Q, +, -, \cdot, ^{-1}, 0, 1\rangle$ denote the field of rational numbers. The following lemma allows us to consider the ordered field $\mathfrak{Q}$.

LEMMA 1. *The relation of order $<$ is strongly programmable in $\mathfrak{Q}$.*

Proof: The idea of a program which decides $x > 0$ is very simple. If we can generate all pairs $\langle i, j\rangle$ of natural numbers, then it will be sufficient to verify the conditions $x = j/i$ and $x = -j/i$. Hence, because every rational x must satisfy for some pair $\langle i, j\rangle$ one of the above equations, our program will always halt.

```
begin
    $p: = 2$; $k: = 2$;
    while $p = 2$ do
    begin
        $i: = k$; $j: = 0$;
        while $i \neq 0$ do
        if $x = -j/i$ then $p: = 0$ else
        if $x = j/i$ then $p: = 1$ else
        begin
            $i: = i-1$; $j: = j+1$
        end;
        $k: = k+1$
    end
end
```

At exit $p = 1$ if $x > 0$, otherwise $p = 0$. ∎

The next problem concerns the axiomatization of algorithmic properties in $\mathfrak{Q}$. May it be the same as for the field $\mathfrak{RO}$? If it might, both fields would be algorithmically equivalent and this is impossible by Lemma 1 and Example 5, § IV. It is also quite clear that algorithmic properties in $\mathfrak{Q}$ cannot have the first order characterization. So, the axioms must be infinite and stronger than those of Archimedean ordered fields.

We shall show that $\mathfrak{Q}$ may be characterized up to isomorphism in algorithmic logic. Consider program E:

```
begin
  while $x \neq y$ do
  if $x > y$ then $x := x - y$ else $y := y - x$
end
```

The program E realizes the famous Euclid's algorithm and is known from practice. It computes the greatest common divisor of two integer numbers as well as the ratio of two commeasurable segments and in this latter sense it was used by the learned Greeks.

Let us denote by Γ_0 the following formula:

$$(\forall x, y)(x > 0 \wedge y > 0 \Rightarrow E\mathbf{1}).$$

THEOREM 1. *Let $\mathfrak{F}$ be an ordered field such that $\mathfrak{F} \models \Gamma_0$. Then $\mathfrak{F}$ is isomorphic to $\mathfrak{Q}$.*

Proof: Let x, y be two positive elements of $\mathfrak{F}$. By Euclid's algorithm we have two sequences $\{x_i\}$, $\{y_i\}$ defined as follows:

$$x_0 = x, \quad y_0 = y;$$

$$x_{i+1} = \begin{cases} x_i - y_i & \text{if} \quad x_i > y_i, \\ x_i & \text{otherwise}; \end{cases}$$

$$y_{i+1} = \begin{cases} y_i - x_i & \text{if} \quad y_i > x_i, \\ y_i & \text{otherwise}. \end{cases}$$

By the axiom Γ_0, there exists n such that $x_n = y_n = a$. By induction x_i, y_i are positive for every i. Hence $a > 0$. Now suppose that $x_i = k \cdot a$ and $y_i = m \cdot a$ for some natural numbers k, m. Then

$$x_{i-1} = (k+m)\cdot a, \qquad y_{i-1} = m \cdot a,$$

or

$$x_{i-1} = k \cdot a, \qquad y_{i-1} = (k+m)\cdot a.$$

But $x_n = y_n = a$ and by induction there exist k, m natural such that $x = k \cdot a$ and $y = m \cdot a$. This immediately implies that $x \cdot y^{-1} = (k \cdot 1)\cdot(m \cdot 1)^{-1}$. But $\mathfrak{F}$ is ordered, hence it must be of characteristic zero. This proves that the set of elements of $\mathfrak{F}$ of the form $(k \cdot 1)\cdot(m \cdot 1)^{-1}$ is isomorphic to $\mathfrak{Q}$. So we proved that the ratio $x \cdot y^{-1}$ of two arbitrary elements of $\mathfrak{F}$ is a rational number. Taking $y = 1$, we see that $\mathfrak{F}$ is isomorphic to $\mathfrak{Q}$. ∎

The following lemma shows that the axiom of Euclid implies that of Archimedes. We recall that the converse is not true, because the field $\mathfrak{RD}$ satisfies the latter without satisfying the first axiom.

LEMMA 2. *For any ordered field $\mathfrak{F}$, if $\mathfrak{F} \models \Gamma_0$, then $\mathfrak{F} \models \Omega_0$.*

Proof: Suppose that $\mathfrak{F}$ is not Archimedean. Then there are two elements $x, y > 0$ such that for every natural n, $n \cdot x < y$. But for these x and y Euclid's algorithm does not terminate, because $y - n \cdot x > x$ for every n. ■

THEOREM 2. *The set of algorithmic properties valid in the field $\mathfrak{Q}$ is a Π_2^0-complete set.*

Proof: Primitive formulas

$$(\forall x_1, \ldots, x_n)\big(\bigvee_{i \in \omega} a_i(x_1, \ldots, x_n) \Rightarrow \bigvee_{i \in \omega} \beta_i(x_1, \ldots, x_n)\big)$$

realized in $\mathfrak{Q}$ are by the Tarski–Kuratowski algorithm [23] of Π_2^0-form. We prove that this set is Π_2^0-complete in the same way as in the case of $\mathfrak{C}$. ■

THEOREM 3. *A total function $f\colon Q \to Q$ is programmable in $\mathfrak{Q}$ iff there exist three total recursive functions g, h, j such that*

$$f\big((n-k)/m\big) = \big(g(n) - h(k)\big)/j(m)$$

for all $n, k, m \in \omega$.

Proof: Only-if-part follows from the effectiveness of computation in the field $\mathfrak{Q}$. Having n, k, m we can effectively compute n_1, k_1, m_1 such that $f\big((n-k)/m\big) = (n_1 - k_1)/m_1$.

To show if-part, let us notice that for any given rational x we can easily regain as in Lemma 1 three positive integers r, s, p such that $x = (r-s)/p$. This step is of course programmable in $\mathfrak{Q}$. Now, by assumption, there are three recursive functions g, h, j such that

$$f\big((r-s)/p\big) = \big(g(r) - h(s)\big)/j(p).$$

By Theorem 2, § II, g, h, j are programmable in $\mathfrak{Q}$. ■

From the above theorem it follows that all functions and relations programmable over $\mathfrak{Q}$ are up to isomorphism recursive objects. So, the theory of programmability over $\mathfrak{Q}$ can be reduced to the theory of recursive functions.

References

[1] Aho, A., Hopcroft, J., Ullman, J., *The design and analysis of computer algorithms*, Adison–Wesley 1974.

[2] de Bakker, J. W., *Recursive procedures*, Mathematical Centre Tracts, Amsterdam 1971.

[3] Banachowski, L., *Extended algorithmic logic, descriptions of programs and their correctness*, Notes of lectures held at Mathematical foundations of computer science semester, Stefan Banach International Mathematical Center, Warsaw 1974.

[4] Bell, J., Slomson, A., *Models and ultraproducts: an introduction*, North Holland, Amsterdam 1969.

[5] Blikle, A., *An extended approach to mathematical analysis of programs*, CC PAS Reports 169, Warsaw 1974.

[6] Engeler, E., *Algorithmic properties of structures*, Math. Systems Theory 1 (1967), 183–195.

[7] — *On the solvability of algorithmic problems*, Report ETH Zürich 1973.

[8] Grabowski, M., *The set of all tautologies of the zero-order algorithmic logic is decidable*, Bull. Acad. Pol. Sci., Sér. Sci. Math. Astronom. Phys. 20 (1972), 575–582.

[9] Greibach, S., *Theory of program structures: schemes, semantics verification*, Lecture Notes on Computer Science 36, Springer Verlag 1975.

[10] Heath, T., *A manual of Greek mathematics*, Oxford 1931.

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

[12] Fischer, P., Probert, R., *Efficient procedures for using matrix algorithms*, Proc. 2nd Coll. Autom. Lang. Program., Lecture Notes on Computer Science 14, Springer Verlag.

[13] Jacobson, N., *Lectures in abstract algebra*, Van Nostrand 1951.

[14] Kfoury, D., *Comparing algebraic structures up to algorithmic equivalence*, Proc. 1st Coll. Autom. Lang. Program., North Holland 1972.

[15] Knuth, D., Floyd, R., *Notes on avoiding "go to" statements*, Inf. Proc. Lett. 1 (1971), 23–31.

[16] Kreczmar, A., *Effectivity problems of algorithmic logic*, Proc. 2nd Coll. Autom. Lang. Program., Lecture Notes on Computer Science 14, Springer Verlag.

[17] McCarthy, J., *A basis for mathematical theory of computation*, Computer programming and formal systems, North Holland 1963.

[18] Mazurkiewicz, A., *Recursive algorithms and formal languages*, Bull. Acad. Pol. Sci., Sér. Sci. Math. Astronom. Phys. 20 (1972), 793–799.

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

[20] Paterson, M., *Equivalence problems in a model of computation*, Ph. D. Thesis, University of Cambridge 1967.

[21] Rasiowa, H., *On logical structure of programs*, Bull. Acad. Pol. Sci., Sér. Sci. Math. Astronom. Phys. 20 (1972), 319–324.

[22] Robinson, A., *Introduction to model theory and to the metamathematics of algebra*, North Holland, Amsterdam 1965.

[23] Rogers, H., Jr., *Theory of recursive functions and effective computability*, McGraw-Hill 1967.

[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*, ibidem, 275–278.

[26] Scott, D., *Outline of a mathematical theory of computation*, Proc. of the 4th Annual Princeton Conference on Information Sciences and Systems, Princeton 1970.

[27] Strassen, V., *Gaussian elimination is not optimal*, Numer. Math. 13 (1969), 354–356.

[28] Whitney, H., *Elementary structures of real algebraic varieties*, Ann. of Math. 66 (1957), 545–556.