On the incompleteness of proving partial correctness

By T. GERGELY and M. SZŐTS

To the memory of Professor László Kalmár

1. Introduction

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

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

2. General principles

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

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

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

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

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

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

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

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

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

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

b) The calculus of **L**.

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

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

3. Interpreting programs in relational structures

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

For the definitions of logic see [8].

Definition 1. We define the syntax of the t-type programming language (P_t).

(i) Symbols of the language:

a) Set of the program variable symbols: $Y = \{y_0, y_1, \ldots, y_i, \ldots\}_{i \in \omega}$

b) Function and relation symbols: $Do(t')$ and $Do(t'')$

c) Logical connectives of classical logic: $\{\neg, \wedge\}$

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

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

b) C_c is the set of control commands:

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

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

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

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

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

These sequences are called t-type programs. □
Let $p\in P_t$. The set of variable symbols occuring in p is designated as Y_p, the set of labels labelling the commands of p as I'_p, the set of labels occuring in it's control commands as I''_p. Let l_y the first label not in I'_p, then $I_p=I'_p\cup I''_p\cup\{l_y\}$.

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

0: $y_2\leftarrow 0$;
1: $y_3\leftarrow 1$;
2: IF $y_2=y_1$ THEN 6;
3: $y_2\leftarrow y_2+1$;
4: $y_3\leftarrow y_3\cdot y_2$;
5: IF $y_1=y_1$ THEN 2;

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

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

so

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

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

The language P is minimal in some respect: some kind of assignment and control commands are needed to build programs. We neglected input-output commands, and do not speak about subroutines. The reason of it is not that if

we could not carry on the same investigation having these kind of commands, but that the result would be the same.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$s = \langle\langle 0, [3, 1, 1]\rangle, \langle 1, [3, 0, 1]\rangle, \langle 2, [3, 0, 1]\rangle,$$
$$\langle 3, [3, 0, 1]\rangle, \langle 4, [3, 1, 1]\rangle, \langle 5, [3, 1, 1]\rangle, \langle 2, [3, 1, 1]\rangle,$$
$$\langle 3, [3, 1, 1]\rangle, \langle 4, [3, 2, 1]\rangle, \langle 5, [3, 2, 2]\rangle, \langle 2, [3, 2, 2]\rangle,$$
$$\langle 3, [3, 2, 2]\rangle, \langle 4, [3, 3, 2]\rangle, \langle 5, [3, 3, 6]\rangle, \langle 2, [3, 3, 6]\rangle,$$
$$\langle 6, [3, 3, 6]\rangle\rangle.$$

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

For the following investigations we need some auxilary definitions:

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

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

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

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

Definition 4. The programming language is a triple:

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

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

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

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

that is there should be a recursive set of formulas (Ax_T) which all the formulas of T can be deduced from. So usually we define M_p as Mod (Ax_T).

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

4. Description of semantics in first order logic

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

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

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

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

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

Formally:

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

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

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

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

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

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

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

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

(ii) The effect of the commands:

a, assignment command: $l_i: y_k \leftarrow f(\bar{y})$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(i) For arbitrary minimal relations:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

isfying $\varphi(\bar{x})$ the appropriate trace in $\mathfrak{A}$ terminates and the output $(\bar{b})$ satisfies $\psi(\bar{y}, \bar{x})$. □

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

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

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

5. Provability of partial correctness

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

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

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

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

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

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

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

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

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

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

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

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

Notice that the theorem could formalize the notion of partial correctness only by a second order formula ($Q_1, \ldots, Q_i, \ldots$ stay here for relational variable symbols). So this theorem failed to give a complete calculus to prove partial correctness. The question has arised whether it is possible to give any. Before giving an answer let

us analyse the question itself, that is the notion of completeness. For a given type t and class of intended models M_p we say that for the programming language $\mathbf{P}_t=\langle P_t, M_p, k\rangle$ there is a complete calculus to prove partial correctnes, if we have a calculus which proves (φ, p, ψ) iff p is partially correct w.r.t. φ, ψ in every model $\mathfrak{A}\in M_p$. So the question can be raised only with respect to the similarity type and the model class of the language. We give some propositions and theorems dealing with different cases.

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

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

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

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

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

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

$$
\begin{aligned}
&0\colon\ y_1 \leftarrow 0;\\
&1\colon\ y_2 \leftarrow 0;\\
&\ \ \vdots\\
&n-1\colon\ y_n \leftarrow 0;\\
&n\colon\ \text{IF } \tau_1(\bar{y})=\tau_2(\bar{y}) \text{ THEN } m+1;\\
&\ \ \vdots\\
&m\colon\ \text{IF } y_1=y_1 \text{ THEN } n;
\end{aligned}
$$

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

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

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

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

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

Similar proofs can be created using any undecidable problem.

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

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

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

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

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

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

b. First order derivations listed below:

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

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

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

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

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

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

are corresponded.

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

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

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

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

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

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

By Theorem 5 the following proposition is evident:

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

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

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

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

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

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

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

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

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

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

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

of arithmetic. It is true that fully automatized algorithm to prove partial correctness in every case can not exist, but with human intuition every program can be proved. Speaking about the mechanisation of program verification this argument underlies the necessity of interactive systems.

Abstract

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

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

References

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

(Received May 29, 1978)