Acta Informatica 16, 199–217 (1981)

Acta Informatica

The Clean Termination of Iterative Programs

Andrzej Blikle

Institute of Computer Science, Polish Academy of Sciences
PKiN, P.O. Box 22, 00-901 Warsaw, Poland

Summary. The paper is devoted to a program-correctness concept which captures partial correctness, termination (nonlooping) and clean termination (nonabortion). The underlying proof method offers a one-stage proof of all the three properties. This method is proved consistent and algebraically complete. It is first discussed for the general case of arbitrary possibly nondeterministic iterative programs. Next, this case is restricted to arbitrary deterministic iterative programs and finally to structured programs. The presented approach is compared with partial correctness, total correctness and weakest precondition methods. The concluding example shows the verification of an arithmetical program in machine-bounded arithmetics. As a side effect of the verification procedure one finds input boundary conditions which guarantee clean termination.

1. Introduction

The total correctness of programs [25, 13, 17] was investigated under the assumption that all elementary instructions, like tests and assignments, represent total functions. Since at the level of algorithmic languages this assumption is never satisfied (due to the error handling mechanism) one must be careful in interpreting the meaning of total correctness at that level. In the environment where instructions represent partial functions the total correctness of a program P wrt a precondition c_{pr} and a postcondition c_{po} means that if some input state s satisfies c_{pr} then the execution of P with s terminates and if this is a clean termination then the output state s' satisfies c_{po}. In other words, using total-correctness proof rules [17] one can formally prove the total correctness of a program which possibly terminates by abortion. For instance one can prove that (1.1) is totally correct *wrt* the given pre- and post-conditions despite that for some inputs this program aborts at the attempt to evaluate the boolean expression $a[i]<a[i-1)$ with $i=0$.

This paper was prepared when the author was visiting the Department of Computer Science of the Technical University of Denmark in Lyngby.

0001-5903/81/0016/0199/$03.80

```
{integer i, n & n>0 & integer array A[0:n], a[0:n]}
   i, a:=n, A;
   while a[i]<a[i-1]
      do a[i], a[i-1]:=a[i-1], a[i];                    (1.1)
         i:=i-1
   od
{a is a permutation of A}
```

An ad hoc solution which is frequently proposed as a remedy for this situation (mainly in private discussions) is the inclusion of error messages in the set of states. With such erroneous states we can make all the functions artificially total hence satisfying the total-correctness preassumption. One can assume further that the erroneous states are always propageted to the end of the program and that when used to evaluate a postcondition always return **false.** In such a model total correctness implies clean termination but then the situation is even worser since total-correctness proof rules become inconsistent. Indeed, we can still prove that (1.1) is totally correct although now it is not! The point is that in the total-correctness proof of (1.1) we check the case where $a[i]<a[i-1]$ returns **true** (the correctness of loop body) and the case where $a[i]<a[i-1]$ returns **false** (exit to the postcondition), but we do not check whether $a[i]<a[i-1]$ yields an error message or not. In order to be able to check this we not only have to extend the proof rules but also have to extend the logic of conditions from the classical two-valued to the nonclassical three-valued case. Both these problems are discussed in this paper.

The problem of clean-termination of programs is not new. A fixed-point oriented method for the detection of all termination cases (both clean and "dirty") in iterative programs was suggested by the author in [9] and [3]. Hitchcock and Park [14] gave another treatment of that problem (cf. also de Bakker [1]). An assertion-oriented method for proving total correctness with clean termination was described by Mazurkiewicz [19] in the general case of abstract sequencial processes. The clean termination property was also considered at a more practical level by Sites [23] who studied possible use of Floyd-Natur invariants in the proofs of this property. These ideas were later independently followed by Coleman and Hughes [10] who investigated possible improvements of Hoare [15] PASCAL axiomatic definition.

The method which is described in this paper bases on the general technique of [19] which is brought here to the level of arbitrary (possibly nondeterministic) iterative programs. This method offers one-stage correctness proofs and covers both total correctness and clean termination. It is first discussed in a semantical model of binary relations (Sect. 2) and abstract flow-charts (Sect. 3). In this model the general proof rule is established and proved consistent and algebraically complete (Sect. 4). Since in the case of deterministic flowcharts our rule may be simplified we discuss this as a separate case in Sect. 5. Further simplification is possible if we structure our programs. This is described at the example of a simplified programming language TOY where the method is brought to the syntactical level. In Sect. 7 the method is compared with partial correctness, total correctness and weakest precondition

methods. Section 8 contains an example of program-correctness verification in the environment of machine-bounded arithmetics. As a side-effect of the program correctness proof one finds input boundary conditions which guarantee clean termination. Concluding remarks of Sect. 9 relate our program verification method to a program derivation method [8].

2. Binary Relations

Let S be an arbitrary nonempty set of *states*. Until Sect. 6 we assume nothing about the nature of the states but we may think of them as of computer memory configurations. By a *binary relation* (shortly *relation*) in S we mean any set $R \subseteq S \times S$. We denote $\mathrm{Rel}(S) = \{R \mid R \subseteq S \times S\}$. By I and ϕ we denote respectively the *identity relation* and the *empty relation* in S. Given two relations $R_1, R_2 \in \mathrm{Rel}(S)$ we define their *composition* by

$$R_1 R_2 = \{(a,b) \mid (\exists c)(a R_1 c \,\&\, c R_2 b)\}.$$

This operation is associative, distributive over arbitrary (finite and infinite) unions and monotone in both arguments *wrt* set-theoretical inclusion. It corresponds to a sequential composition of two programs.

Given an arbitrary relation R we define

$$R^0 = I$$
$$R^{n+1} = R R^n \quad \text{for } n \geqq 0$$
$$R^* = \bigcup_{n=0}^{\infty} R^n.$$

The latter operation is called *iteration* (of R) and corresponds to a nondeterministic loop construct with the body R.

The operation of composion may be extended to the case where one of the arguments is a subset of S. Let $R \in \mathrm{Rel}(S)$ and $A \subseteq S$. Then we define

$$AR = \{s \mid (\exists a)(a \in A \,\&\, a R s)\} \text{ – the } \textit{image} \text{ of } A$$
$$RA = \{s \mid (\exists a)(s R a \,\&\, a \in A)\} \text{ – the } \textit{coimage} \text{ of } A.$$

Intuitively AR is the set of all outputs of R generable from inputs in A and RA is the set of all inputs of R which may generate outputs in A. Notice that $s \in RA$ should read "there exists a computation of R which starts from s and terminates in A" rather than "all computations of R which start from s terminate in A". If R represents a nondeterministic program (i.e. R is not a function) then there may exist computations of R which start from s but do not terminate in A (possibly do not terminate at all).

The extended operation of composition is also distributive over arbitrary unions (of both relations and sets), monotone in both arguments, but only weakly associative:

$$A(R_1 R_2) = (AR_1) R_2$$
$$(R_1 R_2) A = R_1 (R_2 A).$$

If ϕ denotes both the empty relation and the empty set then we have

$$\phi G=\phi \quad \text{and} \quad G\phi=\phi$$

independently whether G is a relation or a set.

For any subset $A\subseteq S$ we define a subset of the identity relation

$$I_A=\{(a,a)|a\in A\}.$$

These subsets of identity are used to describe tests in programs. We have the following obvious properties:

$$I_{A\cup B}=I_A\cup I_B \qquad I_AI_A=I_A$$
$$I_{A\cap B}=I_A\cap I_B=I_AI_B \qquad I_AR\subseteq R$$
$$I_AI_B=I_BI_A \qquad RI_A\subseteq R.$$

The use of these relations as the semantic representations of tests requires a few more comments. In the classical logic every condition (predicate) say c, is a total two-valued function and therefore can be unambiguously represented by the set $T_c=\{s\,|\,c(s)=\textbf{true}\}$. The set $F_c=\{s\,|\,c(s)=\textbf{false}\}$ is implicite in the specification of T_c since $F_c=S-T_c$. In programming language conditions (boolean expressions) represent partial two-valued functions (cf. Sect. 1) and therefore must be represented by two sets T_c and F_c. These sets are, of course, disjoint but not necessarily complementary in S. The set $U_c=S-(T_c\cup F_c)$ contains all the states for which c is undefined. This treatment of conditions is essential in our approach. It allows us to describe all these cases of abortion which are due to condition evaluation.

3. Abstract Flowcharts

Abstract flowcharts are general semantic models of iterative programs. Below we describe this concept together with the corresponding operational and fixed-point semantics.

By an *abstract flowchart* we mean a two-dimensional array of relations $\{R_{ij}\}_{i,j=1}^{n}$ with $R_{ij}\in\mathrm{Rel}(S)$. Of course, each such an array represents a full decorated graph where the integers $1,\ldots,n$ are nodes, ordered pairs of integers are edges and where the edge (i,j) is decorated by R_{ij}. Intuitively R_{ij} represent $I/0$ relations of boxes in concrete flowcharts. If a flowchart is deterministic then R_{ij} are functions. If there is no one-step transition between the nodes i and j then $R_{ij}=\phi$. If i is a branching node with the associated condition c and if the *true* branch goes to j and the *false* branch goes to k then $R_{ij}=I_{T_c}$ and $R_{ik}=I_{F_c}$.

With every abstract flowchart $\{R_{ij}\}_{i,j=1}^{n}$ we associate a relation Res called the *resulting relation* of this flowchart. Res is defined in the following standard way. First with every pair of nodes i and j we associate the set Tr_{ij} of decorations of all chains from i to j. The elements of Tr_{ij} are finite strings of relations and are called *i,j-traces*' Each such trace represents a potential

computation history between the control states i and j. The computational effect of a trace $(R_1, \ldots, R_m)$ is, of course, the composition of its successive elements. Denote this composition by

$$C(R_1, \ldots, R_m) = R_1 \ldots R_m$$

and assume that $C(\varepsilon) = I$, where ε is the empty string of relations. The computational effect of the set of traces Tr_{ij} is the alternative (union) of the effects of all its elements. We denote it by Res_{ij} and define by

$$\mathrm{Res}_{ij} = \bigcup \{C(t) | t \in \mathrm{Tr}_{ij}\}.$$

Now, if we assume that 1 is the *initial node* and n is the *terminal node* then we simply set

$$\mathrm{Res} = \mathrm{Res}_{1n}.$$

The resulting relations of flowcharts may also be described using fixed-point techniques. Since we shall need this in the proof of our main theorem of Sect. 4 we recall below some main concepts and facts. Readers who are not interested in proofs may simply skip this part and proceed to Sect. 4.

With every node i of a flowchart we associate two relations

$$\mathrm{Head}_i = \mathrm{Res}_{1i} \quad \text{and} \quad \mathrm{Tail}_i = \mathrm{Res}_{in}.$$

Head_i describes the relationship between input states and the states which momentarily appear in i. This relation can be said to describe *i-initiations* of the flowchart. Tail_i describes the relationship between the momentary states in i and the output states. This relation describes *i-continuations* of the flowchart (cf. the tail functions of Mazurkiewicz [18] and also the continuations of Strachey and Wadsworth [24]). Of course, the composition of Head_i with Tail_i describes the effect of all these $1, n$-traces which pass through i. Therefore

$$\mathrm{Head}_i \, \mathrm{Tail}_i \subseteq \mathrm{Res} \quad \text{and}$$
$$\mathrm{Res} = \mathrm{Head}_n = \mathrm{Tail}_1.$$

Now, we can formulate two symmetrical fixed-point theorems:

Lemma 1. (The initiation fixed-point theorem.) *The family of relations* $\{\mathrm{Head}_i\}_{i=1}^n$ *is the least solution of the set of equations:*

$$\{X_i = X_1 R_{1i} \cup \ldots \cup X_n R_{ni} \cup R_{1i}\}_{i=1}^n. \quad \square$$

Lemma 2. (The continuation fixed-point theorem.) *The family of relations* $\{\mathrm{Tail}_i\}_{i=1}^n$ *is the least solution of the set of equations:*

$$\{X_i = R_{i1} X_1 \cup \ldots \cup R_{in} X_n \cup R_{in}\}_{i=1}^n. \quad \square$$

Both theorems are well known in the literature (cf. [2], or [9]). For proofs in the calculus of relations see [5].

4. The Global Correctness of Programs

In this section we define our concept of correctness, we describe the corresponding proof rule and we prove that this rule is consistent and algebraically complete. We call our correctness a global correctness of a program since it refers to input-output – hence global – properties of a program. This notion was formerly used in [8] in the context of a program-derivation method where it was contrasted to a correctness which refers to both global and local properties of a program.

Let π be an arbitrary program represented by an input-output relation $R \in \mathrm{Rel}(S)$ and let $B, C \subseteq S$ represent respectively the set of possible inputs (the precondition) and the set of expected outputs (the postcondition) of π. We say that π is *globally correct wrt B and C* if

$$B \subseteq R\,C. \tag{4.1}$$

Intuitively, if (4.1) holds then for any state $s \in B$ there exists a successfull execution of π which ends in C. Of course, if π is deterministic, i.e. if R is a function, then (4.1) means that for any state $s \in B$ the unique s-execution of π terminates successfully in C. We say that an abstract flowchart $\{R_{ij}\}_{i,j=1}^{n}$ is globally correct wrt B and C if

$$B \subseteq \mathrm{Res}\,C. \tag{4.2}$$

Theorem 1. (The general proof rule for abstract flowcharts.) *For any abstract flowchart* $\{R_{ij}\}_{i,j=1}^{n}$ *with the resulting relation* Res *and for any two sets* $B, C \subseteq S$ *the inclusion* $B \subseteq \mathrm{Res}\,C$ *holds* **iff** *there exists a family of sets* $\{A_{ij}^{k}\}_{k=1}^{\infty}{}_{i,j=1}^{n}$ *such that:*

$$1^\circ \quad B \subseteq \bigcup_{k=1}^{\infty} \bigcup_{j=1}^{n} A_{1j}^{k}$$

$$2^\circ \quad \{A_{ij}^{k+1} \subseteq R_{ij} \bigcup_{p=1}^{n} A_{jp}^{k}\}_{k=1}^{\infty}{}_{i,j=1}^{n} \tag{4.3}$$

$$3^\circ \quad \{A_{in}^{1} \subseteq R_{in}\,C\}_{i=1}^{n}; \qquad \{A_{ij}^{1} = \phi\}_{i=1\ j\neq n}^{n}. \quad \square$$

Before we proceed to the proof a few remarks are in order. The sets A_{ij}^{k} – which we call *global inductive assertions* (or g.i.a. in short) – represent properties of states and are associated with the edges of our flowchart. This is in contrast to Naur-Floyd inductive assertions, which are associated with nodes, and is the consequence of the requirement of nonabortion in a nondeterministic framework (see Sect. 5 for further discussion). The intuitive interpretation of A_{ij}^{k} is the following: If we activate the flowchart at i with a state $s \in A_{ij}^{k}$, then the computationally next node is j and the execution reaches node n in no more than k steps. Now, 1° means that if we activate the flowchart at 1 with a state in B then the execution reaches n in a finite time. 2° means that if we are already at i with the state $s \in A_{ij}^{k+1}$ then we can execute R_{ij} and get a new state s' in $\bigcup_{p=1}^{n} A_{jp}^{k}$. The fact that s' is in $\bigcup_{p=1}^{n} A_{jp}^{k}$ means that with s' in j we can make

another step to some p. Due to the possible nondeterminism of the flowchart this rule cannot indicate any particular p. Since in each step the upper index k decrements, our execution must terminate in a finite time. 3° means that the last step of every execution leads to n with a state in C. The index k in A_{ij}^k ranges over positive integers and is used, of course, to prove that our program does not loop indefinitely. This can be easily generalized to the well-known technique of *well-founded sets* ([25, 13, 17]). The **if** part of Theorem 1 may be interpreted as the consistency of our method whereas the **only if** part claims sort of an algebraic completeness. This is not a logical completeness since the existence of the family of sets $\{A_{ij}^k\}_{k=1}^{\infty}{}_{i,j=1}^{n}$ does not guarantee that the sets A_{ij}^k may be expressed (are definable) in an appropriate logic. The algebraic completeness indicates therefore that the limitations of our method are only those imposed by the limitations of logic.

Proof of Theorem 1. We start from the **only if** part. Let $B \subseteq \mathrm{Res}\, C$. The required family of g.i.a. is defined as follows:

$$\{A_{in}^1 = R_{in}\, C\}_{i=1}^n; \qquad \{A_{ij}^1 = \phi\}_{i=1}^n \, j \neq n \tag{4.4}$$

$$\{A_{ij}^{k+1} = R_{ij} \bigcup_{p=1}^{n} A_{jp}^k\}_{k=1}^{\infty}{}_{i,j=1}^{n}.$$

By this definition 2° and 3° are satisfied. In order to prove 1° consider the family of relations $\{T_i^k\}_{k=1}^{\infty}{}_{i=1}^{n}$ defined by the equations

$$\{T_i^1 = R_{in}\}_{i=1}^n$$

$$\{T_i^{k+1} = R_{i1}\, T_1^k \cup \ldots \cup R_{in}\, T_n^k\}_{k=1}^{\infty}{}_{i=1}^{n}.$$

By Lemma 2 and by Kleene's fixed point theorem

$$\{\mathrm{Tail}_i = \bigcup_{k=1}^{\infty} \mathrm{T}_i^k\}_{i=1}^n. \tag{4.5}$$

By the induction on k we shall prove

$$\{T_i^k\, C = \bigcup_{j=1}^{n} A_{ij}^k\}_{k=1}^{\infty}{}_{i=1}^{n}. \tag{4.6}$$

For $k=1$, $T_i^1\, C = R_{in}\, C = A_{in}^1 = \bigcup_{j=1}^{n} A_{ij}^1$. For $k \geqq 1$

$$T_i^{k+1}\, C = R_{i1}\, T_1^k\, C \cup \ldots \cup R_{in}\, T_n^k\, C$$

$$= R_{i1} \bigcup_{j=1}^{n} A_{1j}^k \cup \ldots \cup R_{in} \bigcup_{j=1}^{n} A_{nj}^k = A_{i1}^{k+1} \cup \ldots \cup A_{in}^{k+1}.$$

Now, by (4.5), (4.6) and the assumption that $B \subseteq \mathrm{Res}\, C$ we get

$$B \subseteq \mathrm{Tail}_1\, C = \bigcup_{k=1}^{\infty} T_i^k\, C = \bigcup_{k=1}^{\infty} \bigcup_{j=1}^{n} A_{1j}^k$$

which completes the proof.

To prove the **if** part assume that 1°, 2° and 3° are satisfied. First we shall prove by the induction on k, the inclusions

$$\{A_{ij}^k \subseteq \mathrm{Tail}_i\, C\}_{k=1}^{\infty}{}_{i,j=1}^{n}. \tag{4.7}$$

For $k=1$ this is true by 3° since $R_{in} \subseteq \mathrm{Tail}_i$. For $k \geqq 1$, by 2° and by Lemma 2

$$A_{ij}^{k+1} \subseteq R_{ij} \bigcup_{p=1}^{n} A_{jp}^k \subseteq R_{ij}\, \mathrm{Tail}_j\, C \subseteq \mathrm{Tail}_i\, C.$$

Now, by (4.7) and 1°

$$B \subseteq \bigcup_{k=1}^{\infty} \bigcup_{j=1}^{n} A_{1j}^k \subseteq \mathrm{Tail}_1\, C = \mathrm{Res}\, C. \quad \square$$

Given an abstract flowchart $\{R_{ij}\}_{i,j=1}^n$ and two set $B, C \subseteq S$ such that this abstract flowchart is correct wrt them we have many families $\{A_{ij}^k\}_{k=1}^{\infty}{}_{i,j=1}^{n}$ which satisfy (4.3). Any such family is called a *family of global inductive assertions* for $\{R_{ij}\}_{i,j=1}^n$ and (B, C).

Theorem 2. *If* $\{R_{ij}\}_{i,j=1}^n$ *is globally correct* wrt B *and* C *then the set of all corresponding families of global inductive assertions is nonempty and is a complete upper semilattice with componentwise ordering by inclusion. The* 1.u.b. *in this lattice is the componentwise set-theoretical union and the greatest element in the lattice is the family defined by* (4.4). □

A routine proof is left to the reader.

5. Deterministic Flowcharts

The fact that g.i.a. are associated to edges rather than to nodes is the consequence of the nondeterminism of flowcharts. Here we show that for deterministic flowcharts one can reduce the family of g.i.a. to only two dimensions thus associating its elements to nodes.

An abstract flowchart is called *deterministic* if the following three conditions are satisfied:

1) all R_{ij} are (partial) functions,

2) for all $i, j, p \leqq n$ if $j \neq p$ then $R_{ij}S \cap R_{ip}S = \phi$, i.e. in all branchings there is always at most one branch which may be executed in a given state,

3) for all $i \leqq n$, $R_{ni} = \phi$, i.e. there is no transition between the terminal node and any other node, including n.

Lemma 3. *If an abstract flowchart is deterministic, the corresponding* Res *relation is a function.* □

Proof. Let Φ: $[\mathrm{Rel}(S)]^n \to [\mathrm{Rel}(S)]^n$ be a function defined as follows: for any

$$(P_1, \ldots, P_n) \in [\mathrm{Rel}(S)]^n, \qquad \Phi(P_1, \ldots, P_n) = (Q_1, \ldots, Q_n),$$

where

$$\{Q_i = R_{i1}\, P_1 \cup \ldots \cup R_{in}\, P_n \cup R_{in}\}_{i=1}^n.$$

By Lemma 2, the least fixed point of Φ is $\{\mathrm{Tail}_i\}_{i=1}^n$. On the other hand it is easy to show that Φ has the following property (which is the consequence of the determinism of the flowchart):

(*) if $P_1, \ldots, P_{n-1}$ are functions and $P_n = \phi$ then $Q_1, \ldots, Q_n$ are functions and $Q_n = \phi$.

Now, by a simple inductive argument, we show that for all $k \geqq 0$, $\Phi^k(\phi, \ldots, \phi)$ is a vector of functions. Since $\Phi^k(\phi, \ldots, \phi) \subseteq \Phi^{k+1}(\phi, \ldots, \phi)$ for all $k \geqq 1$, the limit of this sequence, which is $\{\mathrm{Tail}_i\}_{i=1}^n = \bigcup_{k=1}^{\infty} \Phi^k(\phi, \ldots, \phi)$, must be a vector of functions. □

Theorem 2. *If* $\{R_{ij}\}_{i,j=1}^n$ *is deterministic then* $B \subseteq \mathrm{Res}\, C$ **iff** *there exists a family of sets* $\{A_i^k\}_{k=1}^{\infty}{}_{i=1}^{n-1}$ *such that*

$$
\begin{aligned}
&1^\circ \quad B \subseteq \bigcup_{k=1}^{\infty} A_1^k \\
&2^\circ \quad \{r_{ij} \cap A_i^{k+1} \subseteq R_{ij} A_j^k\}_{k=1}^{\infty}{}_{i,j=1}^{n-1}; \qquad (5.1) \\
&\qquad \{A_i^{k+1} \subseteq \bigcup_{j=1}^{n-1} r_{ij}\}_{k=1}^{\infty}{}_{i=1}^{n-1} \\
&3^\circ \quad \{A_i^1 \subseteq R_{in} C\}_{i=1}^{n-1};
\end{aligned}
$$

where $r_{ij} = R_{ij} S$ for $i \leqq n, j \leqq n-1$. □

Proof. Let $B \subseteq \mathrm{Res}\, C$. Then by Theorem 1 there exists a family $\{A_{ij}^k\}_{k=1}^{\infty}{}_{i,j=1}^{n}$ which satisfies (4.3). Define the family $\{A_i^k\}_{k=1}^{\infty}{}_{i=1}^{n-1}$ by the equations $\{A_i^k = \bigcup_{j=1}^{n} A_{ij}^k\}_{k=1}^{\infty}{}_{i=1}^{n-1}$. We shall prove that this family satisfies (5.1). Inclusion 1° is obviously satisfied by 1° of (4.3). Inclusion 2° of (4.3) may be written in the form

$$\{A_{ij}^{k+1} \subseteq R_{ij} A_j^k\}_{k=1}^{\infty}{}_{i,j=1}^{n}. \qquad (5.2)$$

By the assumption that the flowchart is deterministic we have $r_{ip} \cap R_{ij} A_j^k =$ **if** $p = j$ **then** $R_{ij} A_j^k$ **else** ϕ. Therefore by (5.2), $r_{ip} \cap A_{ij}^{k+1} = \phi$ for $p \neq j$. On the other hand

$$A_{ij}^{k+1} \subseteq R_{ij} A_j^k \subseteq R_{ij} S = r_{ij}$$

hence $r_{ij} \cap A_{ij}^{k+1} = A_{ij}^{k+1}$. Consequently $A_{ij}^{k+1} = r_{ij} \cap A_i^{k+1}$ which proves 2° of (5.1). Inclusions 3° of (5.1) follow immediately from 3° of (4.3). This completes the **only if** part of the proof.

Now, suppose that there exists a family $\{A_i^k\}_{k=0}^{\infty}{}_{i=1}^{n-1}$ which satisfies (5.1). We define the family $\{A_{ij}^k\}_{k=1}^{\infty}{}_{i,j=1}^{n}$ by the following equations:

$$\{A_{ij}^k = r_{ij} \cap A_i^k\}_{k=2}^{\infty}{}_{i=1}^{n-1}{}_{j=1}^{n} \qquad (5.3)$$

$$\{A_{nj}^k = \phi\}_{k=1}^{\infty}{}_{j=1}^{n} \qquad (5.4)$$

$$\{A_{ij}^1 = \phi\}_{i=1}^{n-1}{}_{j=1}^{n-1}; \qquad \{A_{in}^1 = A_i^1\}_{i=1}^{n-1}. \qquad (5.5)$$

By the determinism of the flowchart we have $r_{in} \cap \bigcup_{i=1}^{n-1} r_{ij} = \phi$ hence by 2° of (5.1) and by (5.3)

$$\{A_{in}^{k+1} = \phi\}_{k=1}^{\infty}{}_{i=1}^{n-1}. \tag{5.6}$$

Again by (5.3) and the second set of inclusions 2° of (5.1) we have $\{\bigcup_{j=1}^{n} A_{ij}^{k} = A_i^k \cap \bigcup_{j=1}^{n} r_{ij} = A_i^k\}_{k=2}^{\infty}{}_{i=1}^{n-1}$. By (5.5) $\{\bigcup_{j=1}^{n} A_{ij}^{1} = A_i^1\}_{i=1}^{n-1}$. Hence

$$\{A_i^k = \bigcup_{j=1}^{n} A_{ij}^k\}_{k=1}^{\infty}{}_{i=1}^{n-1}.$$

This proves 1° of (4.3) by 1° of (5.1). By 3° of (5.1) by (5.5) and (5.4) we have 3° of (4.3). By 2° of (5.1) we have 2° of (4.3) for $i,j \leqq n-1$. By (5.6) and (5.5) we have the remaining cases of 2° of (4.3). □

6. Global Correctness Associated to a Language

In order to carry out the semantical analysis of concrete programs one must associate a programming language with a proof method. Below we define a simplified programming language TOY and we show how to establish the corresponding global-correctness proof rules. These rules are then referred to in Sect. 7 where we compare global correctness proof techniques with those for partial and total correctness. An example of program verification follows in Sect. 8.

TOY contains the following four syntactical categories: IDE – of *identifiers*, EXP – of *expressions* with a subcategory CON – of *conditions* (boolean expressions) and INS – of *instructions*. We assume that the sets IDE, EXP and CON have been defined in some standard way and we set the following BNF grammar for instructions:

INS ::= **skip** | **if** CON **fi** | {IDE} := {EXP}
| INS; INS
| **if** CON **then** INS **else** INS **fi**
| **while** CON **do** INS **od.**

We add to this grammar the usual context-sensitive restriction that the string {IDE} is repetition free and of the same length as the string {EXP}.

In order to define the semantics of TOY assume that we are given an abstract data type with the (possibly heterogenous) set of values D and an associated set of operations Op. By a *state* we mean any total function s: IDE $\underset{t}{\to} D$ and by

$$S = \text{IDE} \underset{t}{\to} D$$

we denote the set of all states. The totality of states means that all variables are global. Now we define the function of semantics which is denoted by [ ]

and which has the following specification:

$$[\,]: \mathrm{IDE} \underset{t}{\rightarrow} S \underset{t}{\rightarrow} D$$
$$[\,]: \mathrm{EXP} \underset{t}{\rightarrow} S \underset{p}{\rightarrow} D$$
$$[\,]: \mathrm{INS} \underset{t}{\rightarrow} S \underset{p}{\rightarrow} S.$$

Of course, p stands for *partial*. Assume that [ ] has been defined for EXP – hence also for CON – in some standard way. The fact that [ ] maps expressions into partial functions reflects the possibility of abortion. The denotational definition of [ ] in INS is given below. x, c, E and IN stand for an identifier, a condition, an expression and an instruction respectively.

(1) $[\mathbf{skip}] = I_S$
(2) $[\mathbf{if}\ c\ \mathbf{fil}] = \{(s,s) | [c](s) = \mathrm{true}\} \subseteq I_S$
(3) $[x_1, \ldots, x_n := E_1, \ldots, E_n]$
$= \{(s_1, s_2) | [E_i](s_1)$ all defined and
$s_2(x_i) = [E_i](s_1)$ and
$s_2(y) = s_1(y)$ for $y \neq x_i\}$
(4) $[\mathrm{IN}_1; \mathrm{IN}_2] = [\mathrm{IN}_1][\mathrm{IN}_2]$
(5) $[\mathbf{if}\ c\ \mathbf{then}\ \mathrm{IN}_1\ \mathbf{else}\ \mathrm{IN}_2\ \mathbf{fi}] = [\mathbf{if}\ c\ \mathbf{fi}][\mathrm{IN}_1] \cup [\mathbf{if} \sim c\ \mathbf{fi}][\mathrm{IN}_2]$
(6) $[\mathbf{while}\ c\ \mathbf{do}\ \mathrm{IN}\ \mathbf{od}] = ([\mathbf{if}\ c\ \mathbf{fi}][\mathrm{IN}])^* [\mathbf{if} \sim c\ \mathbf{fi}]$.

For the sake of the definition of correctness we introduce an auxiliary notation. For any condition c by

$$\{c\} = \{s | [c](s) = \mathrm{true}\}$$

we denote the set of all states which satisfy c. This is exactly the set T_c as defined in Sect. 2. We say that c_1 implies c_2, in symbols $c_1 \Rightarrow c_2$ **if** $\{c_1\} \subseteq \{c_2\}$. An instruction IN is called *globally correct* wrt a precondition c_{pr} and *a* postcondition c_{po}, which we denote by $c_{\mathrm{pr}} \| \mathrm{IN} \|_g c_{\mathrm{po}}$, if

$$\{c_{\mathrm{pr}}\} \subseteq \mathrm{IN}\{c_{\mathrm{po}}\}. \tag{6.1}$$

The corresponding proof rules are theorems similar to Theorems 1 and 2 in Sect. 4 and Sect. 5. In order to formulate them in a possibly transparent form we shall use diagrams

$$\left|\begin{array}{l} X \\ \hline \downarrow Y \end{array}\right. \quad \text{and} \quad \left|\begin{array}{l} \uparrow X \\ \hline \downarrow Y \end{array}\right.$$

which we read **if** X **then** Y and X **iff** Y respectively

$$\left|\begin{array}{l} \uparrow c_{\mathrm{pr}} \Rightarrow c_{\mathrm{po}} \\ \hline \downarrow c_{\mathrm{pr}} \| \mathbf{skip} \|_g c_{\mathrm{po}} \end{array}\right. \tag{6.2}$$

$$\left|\begin{array}{l} \uparrow c_{\mathrm{pr}} \Rightarrow c\ \&\ c_{\mathrm{po}} \\ \hline \downarrow c_{\mathrm{pr}} \| \mathbf{if}\ c\ \mathbf{fi} \|_g c_{\mathrm{po}} \end{array}\right. \tag{6.3}$$

$$\left|\begin{array}{l} \uparrow c_{\mathrm{pr}} \Rightarrow c_{\mathrm{po}}(E_1/x_1, \ldots, E_n/x_n)\ \&\ E_1 = !\ \&\ \ldots\ \&\ E_n = ! \\ \hline \downarrow c_{\mathrm{pr}} \| x_1, \ldots, x_n := E_1, \ldots, E_n \|_g c_{\mathrm{po}} \end{array}\right. . \tag{6.4}$$

Here, by $c_{\mathrm{po}}(E_1/x_1, \ldots, E_n/x_n)$ we denote the result of the substitution of x_i by E_i in c_{po} and by $E=!$ we denote the boolean expression which is true whenever E is defined.

$$\frac{\begin{array}{ll}1^\circ & c_{\mathrm{pr}} \,\|\mathrm{IN}_1\|_g\, c_a \\ 2^\circ & c_a \,\|\mathrm{IN}_2\|_g\, c_{\mathrm{po}}\end{array}}{c_{\mathrm{pr}} \,\|\mathrm{IN}_1; \mathrm{IN}_2\|_g\, c_{\mathrm{po}}} \tag{6.5}$$

$$\frac{\begin{array}{ll}1^\circ & c_{\mathrm{pr}} \Rightarrow c \vee \sim c \\ 2^\circ & c_{\mathrm{pr}} \,\&\, c \,\|\mathrm{IN}_1\|_g\, c_{\mathrm{po}} \\ 3^\circ & c_{\mathrm{pr}} \,\&\sim c \,\|\mathrm{IN}_2\|_g\, c_{\mathrm{po}}\end{array}}{c_{\mathrm{pr}} \,\|\textbf{if } c \textbf{ then } \mathrm{IN}_1 \textbf{ else } \mathrm{IN}_2 \textbf{ fi}\|_g\, c_{\mathrm{po}}} \tag{6.6}$$

$$\frac{\begin{array}{ll}1^\circ & c_{\mathrm{pr}} \Rightarrow c_a \\ 2^\circ & c_a \Rightarrow c \vee \sim c \\ 3^\circ & (c_a \,\&\, c \,\&\, E=z) \,\|\mathrm{IN}\|_g (c_a \,\&\, E<z) \\ 4^\circ & c_a \,\&\sim c \Rightarrow c_{\mathrm{po}}\end{array}}{c_{\mathrm{pr}} \,\|\textbf{while } c \textbf{ do } \mathrm{IN} \textbf{ od}\|_g\, c_{\mathrm{po}}} \tag{6.7}$$

In the last rule we assume that the expression E, which we call the *termination expression* has the following property: $[E]: \{c_a\} \to W$ is a total function and W is a well-founded set partially ordered by the relation "$<$". We also assume that z remains constant in IN.

Rules (6.5) and (6.7) involve only single assertions; rule (6.6) involves no assertions whatsoever. This contrasts, of course, with the general case of Theorem 2 (an infinite set of assertions) and is the consequence of the simplicity of structured programs. Moreover in (6.7) we have replaced a (one-dimensional) infinite family by the pair consisting of c_a and E. The condition c_a – which we call *loop assertion* – represents the union of all elements of this family and E represents the upper indexing. The same technique, which is of course well known, could be used to modify Theorem 1 and 2.

The proofs of our rules are very simple. As an example, we show the proof of (6.7). Let s be a state which satisfies c_{pr}. Then by 1° and 2° s satisfies either $c_a \,\&\sim c$ or $c_a \,\& c$. In the former case s satisfies c_{po} by 4° hence $s \in [\textbf{if } c \textbf{ fi}]\{c_{\mathrm{po}}\} \subseteq ([\textbf{if } c \textbf{ fi}][\mathrm{IN}])^*[\textbf{if } \sim c \textbf{ fi}]\{c_{\mathrm{po}}\}$ and we are done. In the latter case we can construct by 3° a finite sequence of states $s_1, \ldots, s_n$ called an *s-computation* such that: (1) $s=s_1$, (2) $s_i[\textbf{if } c \textbf{ fi}][\mathrm{IN}]s_{i+1}$ for $i \leqq n$, (3) $n \geqq 2$. In the set of all such computations there must be unextendable computations since otherwise the wellfoundness of the range of $[E]$ would be violated. Let then $s_1, \ldots, s_n$ be unextendable. By 3° and 2° this implies that s_n must satisfy $c_a \,\&\sim c$, hence it must satisfy c_{po} which completes the proof.

It should be noticed that (6.5) and (6.7) have only **if** arrows. The **iff** extension is, of course, possible at the semantic ground, but cannot be expected whenever we have to express our assertions c_a and termination expressions E in a reasonable language (cf. the discussion which follows Theorem 1 in Sect. 4).

7. The Comparison with the Other Concepts of Correctness

In this section we discuss the relationships between global correctness on one hand and partial correctness, total correctness and Dijkstra's weakest preconditions on the other. Let us start from partial correctness.

A program π represented by an $I/0$ relation R is called *partially correct* wrt a precondition B and a postcondition C if

$$BR \subseteq C.$$

This reflects the usual understanding of partial correctness (cf. [25, 21, 13] 15). In the general nondeterministic case, partial and global correctness are incomparable since in partial correctness we make a statement about all the outputs of π whereas in global correctness we only guarantee the existence of an output with the required property. Of course, for deterministic programs global correctness always implies partial correctness, e.i. if R is a function then $B \subseteq RC$ implies $BR \subseteq C$.

In view of this implication one may ask if there is any relationship between the family of Floyd-Naur inductive assertions and our g.i.a. This is discussed below. As is well known [2] the **if** part of the following theorem describes the Floyd-Naur method of proving partial correctness.

Theorem 3. *If* $\{R_{ij}\}_{i,j=1}^{n}$ *is deterministic then* $B\ \mathrm{Res} \subseteq C$ **iff** *there exists a family of sets* $\{A_i\}_{i=1}^{n-1}$ *such that*

$$\begin{aligned} &1^\circ \quad B \subseteq A_1 \\ &2^\circ \quad \{A_i R_{ij} \subseteq A_j\}_{i,j=1}^{n-1} \\ &3^\circ \quad \{A_i R_{in} \subseteq C\}_{i=1}^{n-1}. \quad \square \end{aligned} \tag{7.1}$$

The proof is immediate. The **if** part by a simple induction on the length of a trace. The **only if** part by setting $A_1 = B\,\mathrm{Head}_1 \cup B$ and $A_i = B\,\mathrm{Head}_i$ for $i = 2, \ldots, n-1$.

Let the family of global inductive assertions from Theorem 2 be called a *small family of global inductive assertions* and the family from Theorem 3 be called *small family of partial inductive assertions* (p.i.a. for short).

Theorem 4. *Let* $\{R_{ij}\}_{i,j=1}^{n}$ *be deterministic and globally correct* wrt *some* $B, C \subseteq S$. *If* $\{A_i^k\}_{k=1}^{\infty}{}_{i=1}^{n-1}$ *is a corresponding small family of global inductive assertions, then* $\left\{\bigcup_{k=1}^{\infty} A_i^k\right\}_{i=1}^{n-1}$ *is a corresponding small family of partial inductive assertions.* $\square$

Proof. In this proof we use two general facts about relations. Let $R_1, R_2 \in \mathrm{Rel}(S)$ and let $A \subseteq S$:

(i) if R_1 is a function, then $(R_1 A) R_1 \subseteq A$,
(ii) if $R_1 S \cap R_2 S = \phi$, then $(R_1 A) R_2 = \phi$.

Easy proofs are left to the reader. Now, let $\{A_i^k\}_{k=1}^{\infty}{}_{i=1}^{n-1}$ satisfy (5.1). Denote by $A_i = \bigcup_{k=1}^{\infty} A_i^k$ for $i \leqq n-1$. Performing the sidewise summation on j we get from

the inequalities 2° of (5.1):

$$\{A_i^{k+1} \subseteq R_{i1} A_1^k \cup \ldots \cup R_{in-1} A_{n-1}^k\}_{k=1}^{\infty}{}_{i=1}^{n-1}.$$

Performing the sidewise summation on k above and adding 3° of (5.1) we get

$$\{A_i \subseteq R_{i1} A_1 \cup \ldots \cup R_{in-1} A_{n-1} \cup R_{in} C\}_{i=1}^{n-1}.$$

Therefore, by (i) and (ii) we get

$$\{A_i R_{ij} \subseteq (R_{ij} A_j) R_{ij} \subseteq A_j\}_{i=1}^{n-1}{}_{j=1}^{n-1}$$
$$\{A_i R_{in} \subseteq (R_{in} C) R_{in} \subseteq C\}_{i=1}^{n-1}.$$

This proves 2° and 3° of (7.1). The remaining condition 1° of (7.1) is obvious from 1° of (5.1). □

According to this theorem one may try to search for a small family of g.i.a. in two steps: (1) finding a small family of p.i.a. and then (2) splitting it into a small family of g.i.a. This method contains, however, a trap since not every small family of p.i.a. may be splitted into a small family of g.i.a. The point is that in splitting p.i.a. into g.i.a. we enrich the former by the mechanism for proving non-looping but we keep unchanged their usuability for proving nonabortion. This can be easily explained on the example of a stright-line program. Consider

$$\begin{aligned} &x := x-1; \\ &y := 1/x. \end{aligned} \tag{7.2}$$

This program is partially correct wrt to the precondition c_{pr}: $x>1$ and the postcondition c_{po}: $y \geqq 0$. We can prove this using the intermediate p.i.a. c: $x \geqq 0$. In order to prove global correctness, which is also the case, our g.i.a. must be stronger: $\bar{c}$: $x>0$.

The next case to consider is total correctness as defined in [17]. This concept has been formalized under the assumption that all functions (expressions) and predicates (boolean expressions) are total. With this assumption the effect of abortion is nonexistent and therefore all we want to prove about a program is that it is partially correct, and that it does not loop indefinitely. This property is reflected in Manna and Pnueli's axioms. If we bring them to the world where the totality of expressions is not the case, then all we can prove is partial correctness plus non-looping. For instance (7.2) can be proved totally correct wrt the precondition $x \geqq 1$ and the postcondition $y \geqq 0$ although for $x=1$ this program aborts. If we wish to capture the error of abortion we have to reformulate the axioms in a way which leads to global correctness (Sect. 6). In the case of structured programs it results in the following modifications (the fact that Manna and Pnueli use two-argument rather than one-argument postconditions is of technical character only and not discussed here).

(1) The setting of the assignment axiom (6.4):

$$\frac{c_{\mathrm{pr}} \Rightarrow c_{\mathrm{po}}(E_1/x_1, \ldots, E_n/x_n) \& E_1 = ! \& \ldots \& E_n = !}{c_{\mathrm{pr}} \| x_1, \ldots, x_n := E_1, \ldots, E_n \|_g \, c_{\mathrm{po}}}$$

(2) The modification of **if-then-else-fi** and **while-do-od** inference rules by adding the requirements 1° in (6.6) and 2° in (6.7).

Since total correctness captures only half of termination it cannot be given an algebraic definition similar to these for partial and total correctness. Such a definition would require an extended algebra of relations where we could distinguish between two erroneous states: error message and infinite computation (cf. [4]).

Regarding Dijkstra's weakest preconditions [11] and [12] their definition in the general nondeterministic case can only be given in an extended algebra of relations (cf. [22]). This is due to the requirement that *all* computations of a nondeterministic program terminate cleanly. For deterministic programs, however, this concept easily relates to global correctness. The weakest precondition of a function F and a postcondition C is the greatest set B such that F is globally correct wrt B and C. The greatest B such that $B \subseteq FC$ is, of course, FC. Hence

$$wp(F, C) = FC.$$

Using this definition one can easily prove Dijkstra's axioms (see [5]). Also Dijkstra's liberal weakest precondition (all computations which terminate must terminate in C) can be defined as the greatest set B which satisfies partial correctness requirement $BF \subseteq C$ for given F and C. This leads to the following formula (cf. [2]):

$$lwp(F, C) = \{s \mid \{s\} F \subseteq C\}.$$

8. An Example of Program Verification

Consider a program similar to that discussed by Coleman and Hughes [10]:

$$\begin{aligned} &\mathrm{IN}: \mathrm{IN}_1: x, y := 0, 1; \\ &\mathrm{IN}_2: \textbf{while } y \leqq n \\ &\qquad \textbf{do } x, y := x+1,\ y+2x+3 \textbf{ od} \end{aligned} \tag{8.1}$$

This program computes the integer square root of n and stores the result in x. We shall assume that the program is executed on a machine where integers are defined only within the interval $\langle -b, B\rangle$ with $-1, 0, 1 \in \langle -b, B\rangle$. We also assume that this machine is equipped with an overflow and underflow detection mechanism which aborts program execution whenever one of these cases occurs. The evaluation of $y \leqq n$ is implemented by the comparison of $y-n$ with zero and therefore may also be subject to abortion. Now we want to prove that our program computes $E(\sqrt{n})$ and we also want to find the constraints on n which would guarantee that the program does not abort. With this we come to a rather subtle problem of choosing an adequate mathematical model where our proof might be carried out. The first candidate seems to be the bounded integer arithmetics with partial arithmetical operations and re-

lations. This model, although mathematically acceptable is technically inconvenient since in the proof we may wish to go beyond the machine arithmetics, e.g. in claiming inequalities like $n \leqq \max$ which must be satisfied but need not be evaluable on the machine. We assume therefore that the machine arithmetics is embedded in the abstract (unbounded) abstract arithmetics. Now, we must very carefully distinguish between machine and abstract operations and relations. Let $+$, $-$, $\leqq$, etc. belong to machine arithmetics and $[+]$, $[-]$, $[\geqq]$, etc. be their abstract extentions. For better readability we are bracketing whole formulas. E.g. $[n<(x+1)^2]$ is a boolean expression evaluated in the abstract arithmetics.

We shall prove that (8.1) is globally correct wrt the following precondition and postcondition:

$$c_{\mathrm{pr}}\colon\ [0 \leqq n \leqq \max]$$
$$c_{\mathrm{po}}\colon\ [x^2 \leqq n < (x+1)^2\ \&\ y=(x+1)^2\ \&\ x \geqq 0.$$

We intend to find the appropriate value of max as a sideeffect of the proof. Of course, we assume that $\max \geqq B$. In the first step (rule (6.5)) we set c_1: $0 \leqq n \leqq \max\ \&\ x=0\ \&\ y=1$ and we obviously have $c_{\mathrm{pr}} \| \mathrm{IN}_1 \|_g c_1$. Now, in order to prove $c_1 \| \mathrm{IN}_2 \|_g c_{\mathrm{pr}}$ we set

$$c_a\colon\ [0 \leqq n \leqq \max\ \&\ 0 \leqq x \leqq B\ \&\ 1 \leqq y \leqq B\ \&\ x^2 \leqq n\ \&\ y=(x+1)^2]$$
$$E\colon\ [n-x^2]$$

where the underlying well-founded set is, of course, $(\{0,1,\ldots\},[<])$. Following rule (6.7) we prove:

1° $c_1 \Rightarrow c_a$ which is obviously true

2° $c_a \Rightarrow y \leqq n \vee \sim y \leqq n$

which is true as long as c_a implies $[-b \leqq y-n \leqq B]$. This, of course, depends on the value of max which must be chosen such that the minimal value of $[y-n]$, which is $[1-\max]$, is not less than $[-b]$. This gives us the first restriction on max:

$$[\max \leqq b+1]$$
$$3^\circ\quad c_a\ \&\ [y \leqq n\ \&\ n-x^2=z] \tag{8.2}$$
$$\Rightarrow (c_a\ \&\ [n-x^2<z])(x/[x+1],\ y/[y+2x+3]).$$

We first perform the required substitution:

$$[0 \leqq n \leqq \max\ \&\ 0 \leqq x+1 \leqq B\ \&\ 1 \leqq y+2x+3 \leqq B\ \&$$
$$(x+1)^2 \leqq n\ \&\ y+2x+3=((x+1)+1)^2\ \& \tag{8.3}$$
$$n-(x+1)^2<z].$$

Now, we have to prove that $c_a \,\&\, [y \leqq n \,\&\, n - x^2 = z]$ implies (8.3). Since

(i) the condition $[y+2x+3=(x+2)^2]$ in (8.3) may be simplified to $[y=(x+1)^2]$ and $[1 \leqq y+2x+3 \leqq B]$ in (8.3) may be replaced by $[1 \leqq (x+2)^2 \leqq B]$;

(ii) the condition $[y=(x+1)^2 \,\&\, y \leqq n]$ in the premise of 3° implies $[(x+1)^2 \leqq n]$ in (8.3);

(iii) the condition $[n-x^2=z]$ in the premise of 3° implies $[n-(x+1)^2<z]$ in (8.3)

it remains to be proved that the premise of 3° implies the restrain condition

$$[0 \leqq n = \max \,\&\, 0 \leqq x+1 \leqq B \,\&\, 1 \leqq (x+2)^2 \leqq B]. \tag{8.4}$$

This again, depends on max. The maximal value of x which still satisfies the premise of 3° is such that $[(x+1)^2 \leqq n < (x+2)^2 \,\&\, x \geqq 0]$ which simply means that $[x = E(\sqrt{n}) - 1]$. With this value of x (8.4) must be satisfied for the maximal value of n. This leads to the inequality

$$[(E(\sqrt{\max}+1)^2 \leqq B] \tag{8.3}$$

which may be solved numerically for any concrete B.

$$4° \quad c_a \,\&\, \sim y \leqq n \Rightarrow c_{\mathrm{po}}.$$

This is true since $\sim y \leqq n \Rightarrow [y>n]$. This step adds no new requirements on max. The final requirement on max is therefore

$$[\max \leqq b+1 \,\&\, E(\sqrt{\max}+1)^2 \leqq B].$$

9. Concluding Remarks

The reader may feel a bit irritated by our enexplained cleverness in guessing the assertion c_a in the proof of Sect. 8. We omitted any comments at that place since the ability of guessing assertions is a necessary prerequisite in any assertion-oriented proof method whether it deals with partial, total or global correctness. On the other hand, the method of global correctness has been thought primarily as a mathematical framework for correct program derivation (cf. [6] and [8]). In the latter method programs are derived – i.e. constructed and refined – along with the extended specifications consisting of a precondition, a postcondition and of all the assertions which are necessary in the proof of global correctness. All program development rules are sound, i.e. they always lead from correct into correct programs. This soundness principle requires a very careful choice of the underlying logic of conditions. The mathematical requirement is that this logic be three-valued. It is also technically convenient if it obeys the rule adopted by Jensen and Wirth [16] in the definition of PASCAL that the evaluation of condition may be discontinued as soon as its value is ascertained. For instance, if in the evaluation of $c_1 \,\&\, c_2$ the value of c_1 is **false** then $c_1 \,\&\, c_2$ is given the value **false** even if the value of c_2 is

undefined. With such a rule we can easily correct the program of Sect. 1 in replacing its **while** condition by $i>0 \& a[i]<a[i-1]$. A logic with the mentioned property has been described in [20] and has been successfully used in the mentioned program derivation method. At this point we disagree with the opinion expressed by Coleman and Hughes [10] that such a logic leads to unnecessarily complicated preconditions.

Another point which requires a few words of comments is our use of unary postconditions (sets of states) as opposed to binary postconditions (relations on states) used by Manna and Pnueli. This difference is purely technical. Whenever we wish to refer in a postcondition to the initial values of some variables we introduce constants which represent the initial values. In the program of Sect. 8 such a constant was n.

Acknowledgements: The reported research was partly supported by the Polish Academy of Sciences under grant MR I/3-04.1.1 and partly by the Danish State Natural- and Technical Sciences Research Foundation under grant no. 511-16000.

References

1. Bakker de, J.W.: Mathematical Theory of Program Correctness, Englewoods Cliffs: Prentice-Hall Int., 1980
2. Bakker de, J.W., Meertens, L.G.L.T.: On the completness of the inductive assertion method. J. Comput. System Sci. **II**, 323–357 (1975)
3. Blikle, A.: An algebraic approach to mathematical theory of programs. CCPAS Reports 119 (1973)
4. Blikle, A.: Proving programs by δ-relations. Formalization of semantics of programming languages and writing of Compilers (Proc. Symp. Frankfurt/Oder 1974). Elektron. Informationsverarbeit. Kybernetik **11**, 267–274 (1975)
5. Blikle, A.: A comparative review of some program-verification methods. In: Mathematical Foundations of Computer Science 1977 (Proc. 6th Symp. Tatranska Lomnica 1977). (J. Gruska, ed.) Lecture Notes in Computer Sciences, Vol. 53, pp. 17–33. Berlin-Heidelberg-New York: Springer 1977
6. Blikle, A.: Assertion programming. In: Mathematical Foundations of Computer Science (Proc. 8th Symposium. Olomouc 1979) (J. Becvar ed.) Lecture Notes in Computer Sciences, Vol. 74. Berlin-Heidelberg-New York: Springer 1979
7. Blikle, A.: On correct program development. (Proc. 4th Int. Conf. on Software Engineering, Sept. 1979 Munich, pp. 164–173) IEEE Catalog No. 79 CH 1479-5 C, 1979b
8. Blikle, A.: On the development of correct specified programs. IEEE Transactions on Software Engineering (to appear in 1981)
9. Blikle, A., Mazurkiewicz, A.: An algebraic approach to the theory of programs, algorithms, languages and recursiveness. Math. Found. Comp. Sci. (Proc. Symp. Warsaw-Jablonna 1972) Warsaw 1972
10. Coleman, D., Hughes, J.W.: The clean termination of PASCAL programs. Acta Informat. **11**, 195–210 (1979)
11. Dijkstra, E.W.: Guarded commands, non-determinancy and a calculus for the derivation of programs. (Proc. 1975 Int. Conf. Reliable Software) Comm. ACM **18**, 453–457 (1975)
12. Dijkstra, E.W.: A discipline of programming. Englewood Cliffs: Prentice Hall 1976
13. Floyd, R.W.: Assigning meanings to programs. Appl. Math. Comput. **19**, 19–32 (1967)
14. Hitchcock, P., Park, D.: Induction rules and termination proofs, in: Automata, Languages and Programming (Proc. IRIA Symp. 1972, M. Nivat, ed.) Amsterdam: North Holland 1973
15. Hoare, C.A.R.: An axiomatic definition of the programming language PASCAL. (International Symposium on Theoretical Programming). Lecture Notes in Computer Science, Vol. 5. Springer: Berlin-Heidelberg-New York 1974

16. Jensen, K., Wirth, N.: PASCAL User Manual. 2nd ed. Springer: Berlin-Heidelberg-New York 1975
17. Manna, Z., Pnueli, A.: Axiomatic approach to total correctness of programs. Acta Informat. **3**, 243–263 (1974)
18. Mazurkiewicz, A.: Proving algorithms by tail functions, Information and Control **18**, 220–226 (1971)
19. Mazurkiewicz, A.: Proving properties of processes. Algorytmy **11**, 5–22 (1974)
20. McCarthy, J.: A basis for a mathematical theory of computation. (Western Joint Computer Conference, May 1961) (P. Braffort, D. Hirschberg, eds.), Computer Programming and Formal Systems. Amsterdam, North Holland: 1967, pp. 33–70
21. Naur, P.: Proof of algorithms by general snapshots. BIT **6**, 310–316 (1966)
22. Roever de, W.P.: Dijkstra's predicate transformer, nondeterminism, recursion and termination. Mathematical Foundation of Computer Science 1976 (Proc. 5th. Symp. Gdansk, September 1976) (A. Mazurkiewicz, ed.) Lecture Notes in Computer Sciences, Vol. 45, pp. 472–481. Berlin-Heidelberg-New York: Springer 1976
23. Sites, R.L.: Proving that computer programs terminate cleanly. STAN-CS-74-418, 1974
24. Strachey, C., Wadworth, C.P.: Continuations, a mathematical semantics for handling full jumps. Technical Monograph PRG-11, Oxford 1974
25. Turing, A.M.: On checking a large routine. Report of a Conference on High Speed Automatic Calculating Machines, pp. 67–69, University Mathematical Laboratory, Cambridge 1949

Received May 15, 1980