Theoretical Computer Science 8 (1979) 73–87
© North-Holland Publishing Company

RECURSIVE ASSERTIONS ARE NOT ENOUGH – OR ARE THEY?*

Krzysztof R. APT†
Mathematical Centre, 2e Boerhaavestraat 49, Amsterdam, The Netherlands

Jan A. BERGSTRA
Institute of Applied Mathematics and Computer Science,
University of Leiden, Wassenaarseweg 80, Leiden, The Netherlands

Lambert G.L.T MEERTENS
Mathematical Centre, 2^e Boerhaavestraat, 49, Amsterdam, The Netherlands

Communicated by Albert Meyer
Received October 1977;
Revised June 1978

Abstract. Call a set of assertions $\mathscr{A}$ *complete* (with respect to a class of programs $\mathscr{S}$) if for any p, $q \in \mathscr{A}$ and $S \in \mathscr{S}$, whenever $\{p\}S\{q\}$ holds, then all intermediate assertions can be chosen from $\mathscr{A}$. This paper is devoted to the study of the problem which sets of assertions are complete in the above sense. We prove that any set of recursive assertions containing *true* and *false* is not complete. We prove the completeness for while programs of some more powerful assertions, e.g. the set of recursively enumerable assertions. Finally, we show that by allowing the use of an 'auxiliary' coordinate, the set of recursive assertions is complete for while programs.

1. Introduction

Two important methods that are used to establish the partial correctness (correctness without regard to termination) are the inductive assertion method of Floyd [6] and the axiomatic method of Hoare [10]. These two methods are closely related; in particular, both use intermediate assertions to express or derive local correctness properties.

A *global* correctness property $\{p\}S\{q\}$ will in practice have *recursive*[1] assertions p and q. The precondition p will usually be some simple condition on the input

* This publication is registered as Report IW 92/77 of the Mathematical Centre.

† Present address: The faculty of Economics, Erasmus University, P.O. Box 1738, Rotterdam, Netherlands.

[1] The word 'recursive' is used in this article in two senses: that of Recursion Theory (meaning 'effectively computable') and that of Program Schemes (meaning 'applied in its definition'). Which meaning is intended will be clear from the context; here of course, the first one is implied.

variables, or even 'true'. Similarly, one may expect that the postcondition q can be checked effectively by inspection of the output variables. A natural conjecture then is that all intermediate assertions may also be chosen recursive. This is of relevance, e.g., in view of proposals for 'assert statements' (see e.g., [12]), where **assert** B, where B is a Boolean expression, supposedly signals an error if B evaluates to false. Such a B should be chosen such that it could have served as intermediate assertion in a correctness proof for the intended program. Since the value of B must be effectively computable, this assertion is recursive. The present paper addresses the question which sets of assertions are sufficiently large to allow the intermediate assertions to be chosen from them. It will be shown that the set of recursive assertions does not suffice, so the above conjecture is false.

This question is one particular aspect of the completeness problem for proof methods, i.e., the problem whether a given method can be used to prove any true proposition from the class to which it pertains. Various results concerning completeness have been obtained, both for the method of Floyd and that of Hoare. For Floyd's method we only mention the papers [1], [2], [9] and [11]. Hoare's method is based on formal deduction systems to derive sentences of the form $\{p\}S\{q\}$, where S is a program from a given programming language and p and q are formulas from a given first-order language of assertions. For this method incompleteness threatens at every turn. We shall briefly review some of the problems and approaches to suppress 'uninteresting' forms of incompleteness.

To start with, there is the relative weakness of formal deduction systems compared to the power of computing systems. Even under rather general assumptions any axiomatizable deduction system H is incomplete. Take, e.g., the language of Peano arithmetic as assertion language. Since a sentence $\{\textbf{true}\}$ skip $\{p\}$ is true iff p is true, we conclude immediately from Gödel's first Incompleteness Theorem that H is incomplete. Now the language of Peano arithmetic is rather powerful, but a restriction to a simpler assertion language is of no help, as the following diagonalization argument shows. Suppose that the class of programs $\mathcal{S}$ under consideration is such that every partial recursive function can be computed by a program from $\mathcal{S}$. (Several extremely simple classes of programs with this property have been exhibited in the literature.) Let H be a formal system to derive asserted statements for $\mathcal{S}$. One can construct a program $S \in \mathcal{S}$ which, for input $i = n$, generates all proofs in H and halts iff it finds $H \vdash \{\textbf{true}\}\ i := n;\ P_n\ \{\textbf{false}\}$, where P_n stands for the nth program in $\mathcal{S}$ according to some enumeration. Such an S diverges for input $i = n$ iff $H \nvdash \{\textbf{true}\}$ $i := n;\ P_n\ \{\textbf{false}\}$. S has itself a number, say n_S. Now $\{\textbf{true}\}\ i := n_S;\ P_{n_S}\ \{\textbf{false}\}$ holds iff $P_{n_S} = S$ diverges for input $i = n_S$, i.e., iff $H \nvdash \{\textbf{true}\}\ i := n_S;\ P_{n_S}\{\textbf{false}\}$, so H is not complete.

A way to overcome this inherent weakness of the axiomatic method has been indicated by Cook [4]. Add to the Hoare system an oracle that can supply answers to questions of the form $\models p$, i.e., 'is p true?' for all first order formulas p (in some given structure with some fixed interpretation). This oracle is incorporated in the system by the

$$\text{Rule of Consequence:}\quad \frac{\models p' \to p, \{p\}S\{q\}, \models q \to q'}{\{p'\}S\{q'\}}.$$

This rule by itself still leaves room for rather inessential forms of incompleteness. For example, Wand [15] exhibits a particular structure for which the necessary intermediate assertions are not first-order definable. This problem had also been tackled by Cook by defining a notion of 'expressiveness' for the assertion language, and restricting the question of completeness to structures with expressive assertion languages. Using as definition for expressiveness of a language L the requirements

(i) for any assertion p from L and any program S the strongest postcondition $\mathrm{sp}(p, S)$ is definable in L, and

(ii) the equality predicate is in the language,

Cook succeeded in showing the completeness of a Hoare system for a language of (essentially) while programs. Gorelick [7] extended this result to a class of programs with recursive procedures. (Following Clarke [5], to prove these results one could also replace the above two requirements by the single one that the weakest precondition be definable.)

Clarke [5] finally reached along this road an incompleteness result: for a programming language with global variables, 'static scope' and recursive procedures with procedure parameters, he proved the incompleteness of any Hoare system, by using a structure with two elements and an expressive assertion language (in the sense of Cook).

For the purpose of the present paper we take the standard model of Peano arithmetic as the underlying structure. As an immediate consequence, the problem of expressiveness disappears if one allows all first-order definable assertions. However, we want to restrict the set of assertions and to ask the question for which sets one obtains completeness.

It is convenient to consider this problem within a relational framework (see, e.g., [2]). We shall view a program as a set of initial and final states, and an assertion as the set of states 'satisfying' it. This approach corresponds to the method of Floyd, but the results are readily translated to Hoare's method (assuming an oracle), where the class of programs under consideration corresponds to while programs.

2. Preliminaries

Throughout the paper, $\mathcal{V} = \{v_1, v_2, \ldots\}$ stands for a finite, nonempty set of 'variables'. A state is a (total) mapping $\mathcal{V} \to \mathcal{N}$, where $\mathcal{N}$ denotes the set of natural numbers. Letters $\sigma, \tau, \ldots$ are used for states. $\mathcal{U}$ denotes the set of all states. A program is a binary relation over the state space $\mathcal{U}$, i.e. a set of pairs of (initial and final) states. An assertion is a subset of $\mathcal{U}$. Programs are denoted by $S, S_1, S_2, \ldots,$ and assertions by $p, q, r, \ldots$.

The fact that $\mathcal{V}$ is finite is merely a matter of convenience. With suitably amended definitions, all theorems remain valid if $\mathcal{V}$ is infinite (which is obvious if one assumes that each particular program uses only a finite number of variables).

Definition 2.1.

$\{p\}S\{q\}$ iff $\forall\sigma, \tau[(\sigma \in p \wedge \sigma S\tau) \rightarrow \tau \in q]$;

$\mathrm{wp}(S, q) = \{\sigma : \forall\tau[\sigma S\tau \rightarrow \tau \in q]\}$;

$\mathrm{sp}(p, S) = \{\tau : \exists\sigma[\sigma \in p \wedge \sigma S\tau]\}$;

$\neg p = \mathcal{U} \backslash p = \{\sigma : \sigma \notin p\}$;

$S_1; S_2 = \{(\sigma, \tau) : \exists\sigma'[\sigma S_1\sigma' \wedge \sigma' S_2\tau]\}$;

$p ** S = \{(\sigma, \tau) : \exists n \exists\sigma_0, \ldots, \sigma_n[\sigma = \sigma_0 \wedge \sigma_n = \tau \wedge \forall i < n[\sigma_i \in p \wedge \sigma_i S\sigma_{i+1}]]\}$;

$p * S = \{(\sigma, \tau) : \sigma(p ** S)\tau \wedge \tau \in \neg p\}$

The form $p * S$ defines, of course, the meaning of a while loop '**while** p **do** S **od**'.

Two obvious but important properties of the notions introduced are:

$$\{p\}S\{q\} \text{ iff } p \subseteq \mathrm{wp}(S, q);$$

$$\{p\}S\{q\} \text{ iff } \mathrm{sp}(p, S) \subseteq q.$$

Note that 'wp' is the weakest precondition for partial correctness. Termination is not implied.

Corollary. *If* $p' \subseteq p$, $\{p\}S\{q\}$ *and* $q \subseteq q'$, *then* $\{p'\}S\{q'\}$.

Definition 2.2. Let f be a partial function from $\mathcal{N}$ into $\mathcal{N}$. We say that S *computes* f if

$$S = \{(\sigma, \tau) : f(\sigma(v_1)) = \tau(v_1) \wedge \forall v \in \mathcal{V}[v \neq v_1 \rightarrow \sigma(v) = \tau(v) = 0]\}.$$

Throughout the paper, $\mathcal{B}$ stands for a set of recursive assertions (the 'conditionals') which is closed under the operations $\cap$ and $\neg$, and $\mathcal{S}$ stands for a class of programs that satisfies the following three properties:

(a) if $S_1, S_2 \in \mathcal{S}$ then $S_1; S_2 \in \mathcal{S}$
($\mathcal{S}$ is closed under sequencing);

(b) if $b \in \mathcal{B}$ and $S \in \mathcal{S}$ then $b * S \in \mathcal{S}$
($\mathcal{S}$ is closed under repetition over the conditionals);

(c) every unary partial recursive function is computed by some program from $\mathcal{S}$
($\mathcal{S}$ has the full power of recursion theory).

Finally, $\mathscr{A}$ stands throughout the paper for a set of assertions which contains at least $\emptyset$ ('false') and $\mathscr{U}$ ('true'), so that trivial completeness is excluded.

We adopt now the following completeness definition:

Definition 2.3. $\mathscr{A}$ is *complete* for $(\mathscr{S}, \mathscr{B})$ if for all $p, q \in \mathscr{A}$, $b \in \mathscr{B}$ and $S, S_1, S_2 \in \mathscr{S}$ the following three requirements are satisfied:

(i) $p \cap b \in \mathscr{A}$;

(ii) if $\{p\}S_1; S_2\{q\}$, then, for some $r \in \mathscr{A}$, $\{p\}\ S_1\{r\}$ and $\{r\}S_2\{q\}$;

(iii) if $\{p\}b * S\{q\}$, then, for some $r \in \mathscr{A}$, $p \subseteq r$, $\{r \cap b\}S\{r\}$ and $r \cap \neg b \subseteq q$.

So, informally speaking, $\mathscr{A}$ is complete for $(\mathscr{S}, \mathscr{B})$ if for every $p, q \in \mathscr{A}$ and $S \in \mathscr{S}$ the truth of $\{p\}S\{q\}$ can be verified using only intermediate assertions from $\mathscr{A}$.

It may seem that we unduly omitted conditional statements from our definition. Note, however, that if $\mathscr{A}$ is complete for $(\mathscr{S}, \mathscr{B})$ and (informally) $\{p\}$ '**if** b **then** S_1 **else** S_2, **fi**' $\{q\}$, with $p, q \in \mathscr{A}$, $b \in \mathscr{B}$ and $S_1, S_2 \in \mathscr{S}$, then the necessary intermediate assertions $p \cap b$ and $p \cap \neg b$ are already elements of $\mathscr{A}$ by virtue of requirement (i).

3. Incompleteness for recursive assertions

For the result of this section, we rely heavily on a theorem due to Mostowski (see [8]):

Theorem 3.1. *There exist two disjoint recursively enumerable subsets X and Y of $\mathscr{N}$, such that for no recursive Z both $X \subseteq Z$ and $Y \cap z = \emptyset$.*

Through the remainder of this section X and Y will stand for two such sets.

Definition 3.1. For $n \in \mathscr{N}$ and $A \subseteq \mathscr{N}$:

$$[\![n]\!] = \lambda v \in \mathscr{V}[\text{if } v = v_1 \text{ then } n \text{ else } 0];$$

$$[\![A]\!] = \{[\![n]\!] : n \in A\}.$$

Theorem 3.2. *If $\mathscr{A}$ is a set of recursive assertions, then $\mathscr{A}$ is incomplete for $(\mathscr{S}, \mathscr{B})$.*

Proof. We exhibit the existence of programs $S_1, S_2 \in \mathscr{S}$ for which $\{\mathscr{U}\}S_1; S_2\{\emptyset\}$, but, for no recursive assertion r, $\{\mathscr{U}\}S_1\{r\}$ and $\{r\}S_2(\emptyset\}$.

Let f_1 be a total function enumerating X, and let S_1 be a program computing f_1. We have, for arbitrary $n \in \mathscr{N}$, $[\![n]\!] \in \text{sp}(\mathscr{U}, S_1)$ iff $\exists m[f_1(m) = n]$, that is, iff $n \in X$. Since obviously $\text{sp}(\mathscr{U}, S_1) \subseteq [\![\mathscr{N}]\!]$, $\text{sp}(\mathscr{U}, S_1) = [\![X]\!]$.

Let f_2 be defined by

$$f_2(n) = \begin{cases} 0 \text{ if } n \in Y, \\ \text{divergent otherwise.} \end{cases}$$

Let S_2 be a program computing f_2. We have, for arbitrary $m \in \mathcal{N}$, $[\![m]\!] \in \mathrm{wp}(S_2, \emptyset)$ iff $\forall n[\neg f_2(m) = n]$, that is, iff $n \notin Y$, so $\mathrm{wp}(S_2, \emptyset) \cap [\![\mathcal{N}]\!] = [\![\neg Y]\!]$, implying $[\![\neg Y]\!] \subseteq \mathrm{wp}(S_2, \emptyset)$. Since $X \cap Y = \emptyset$, $X \subseteq \neg Y$, so $[\![X]\!] \subseteq [\![\neg Y]\!]$. Now we have $\mathrm{sp}(\mathcal{U}, S_1) = [\![\neg Y]\!] \subseteq \mathrm{wp}(S_2, \emptyset)$, so clearly $\{\mathcal{U}\}S_1;\ S_2\{\emptyset\}$.

Now assume that $\{\mathcal{U}\}S_1\{r\}$ and $\{r\}S_2\{\emptyset\}$. Then $[\![X]\!] = \mathrm{sp}(\mathcal{U}, S_1) \subseteq r$ and $r \cap [\![Y]\!] \subseteq \mathrm{wp}(S_2, \emptyset) \cap [\![Y]\!] = \emptyset$. The set Z defined by $[\![Z]\!] = r \cap [\![\mathcal{N}]\!]$ satisfies $X \subseteq Z$ and $Y \cap Z = \emptyset$ and is, by theorem 3.1, not recursive. Consequently, r is not recursive either.

In our opinion, this result shows that assert statements have only limited applicability. It might be argued, however, that the notion of *partial* correctness – essential to our proofs – is not the proper one to consider here, and that the conditional of an assert statement should also express termination. Although termination cannot be dealt with by Floyd's (nor by Hoare's) method, it is not difficult to show that this suggestion does not even save the assert statement in the simple case of deterministic programs. For, if $\{p\}S_1;\ S_2\{q\}$ holds in the sense of total correctness, an intermediate r would have to satisfy

$$\mathrm{sp}(p, S_1) \subseteq r \subseteq \mathrm{wp}(S_2, q) \cap \neg\mathrm{wp}(S_2, \emptyset),$$

where the addition $\neg\mathrm{wp}(S_2, \emptyset)$ expresses termination. Now, let Z be an arbitrary recursively enumerable, but not recursive, subset of $\mathcal{N}$, and let S_1 and S_2, respectively, compute a total function with range Z and a partial function with domain Z. Clearly, $\{\mathcal{U}\}S_1;\ S_2\{\mathcal{U}\}$ in the sense of total correctness. Since $\mathrm{sp}(\mathcal{U}, S_1) = \neg\mathrm{wp}(S_2, \emptyset) = [\![Z]\!]$, r must be equal to $[\![Z]\!]$, which is not recursive.

4. More powerful assertions

Having established that the set of recursive assertions has insufficient power for completeness, we now turn to more powerful classes. It is clearly fruitless to hope for completeness proofs without additional assumptions about the class of programs.

Definition 4.1. A program S is *normal* if (the set which is) S is recursively enumerable.

Observe that this is a quite normal property for programs indeed; it certainly holds for all programs corresponding to computational processes.

Lemma 4.1.

(a) *If S is a normal program and $b \in \mathcal{B}$, then*
*b ** S is a normal program.*

(b) *If S is a normal program and p is recursively enumerable, then*
$\mathrm{sp}(p, S)$ *is recursively enumerable.*

(c) *If S is a normal program and $\neg q$ is recursively enumerable, then $\neg \mathrm{wp}(S, q)$ is recursively enumerable.*

Proof.

(a) Since b is recursive and S is recursively enumerable, $b ** S = \{(\sigma, \tau) : \exists n \exists \sigma_0, \ldots, \sigma_n[\sigma = \sigma_0 \wedge \sigma_n = \tau \wedge \forall i < n[\sigma_i \in b \wedge \sigma_i S \sigma_{i+1}]]\}$ is recursively enumerable.

(b) Since p and S are recursively enumerable, $\mathrm{sp}(p, S) = \{\tau : \exists \sigma[\sigma \in p \wedge \sigma S \tau]\}$ is recursively enumerable.

(c) Since S and $\neg q$ are recursively enumerable, $\neg \mathrm{wp}(S, q) = \{\sigma : \exists \tau[\sigma S \tau \wedge \tau \in \neg q]\}$ is recursively enumerable.

Theorem 4.1. *If $\mathscr{S}$ is a class of normal programs and $\mathscr{A} = \{p : p$ is recursively enumerable$\}$, then $\mathscr{A}$ is complete for $(\mathscr{S}, \mathscr{B})$.*

Proof. We shall verify each of the three requirements from definition 2.3.

First, if $p \in \mathscr{A}$ and $b \in \mathscr{B}$, then, since b is recursive, $p \cap b$ is recursively enumerable, so $p \cap b \in \mathscr{A}$.

Next, suppose $\{p\}S_1; S_2\{q\}$ for some $S_1, S_2 \in \mathscr{S}$ and $p, q \in \mathscr{A}$. Take $r = \mathrm{sp}(p, S_1)$. Clearly, $\{p\}S_1\{r\}$ and $\{r\}S_2\{q\}$. By (b) of lemma 4.1, r is recursively enumerable, so $r \in \mathscr{A}$.

Finally, suppose $\{p\}b * S\{q\}$ for some $S \in \mathscr{S}$, $b \in \mathscr{B}$ and $p, q \in \mathscr{A}$. Take $r = \mathrm{sp}(p, b ** S)$. It is easy to verify that r is a proper invariant, i.e., that $p \subseteq r$, $\{r \cap b\}S\{r\}$ and $r \cap \neg b \subseteq q$. By (a) and (b) of lemma 4.1, r is recursively enumerable, so $r \in \mathscr{A}$.

Theorem 4.2. *If $\mathscr{S}$ is a class of normal programs and $\mathscr{A} = \{p : \neg p$ is recursively enumerable$\}$, then $\mathscr{A}$ is complete for $(\mathscr{S}, \mathscr{B})$.*

Proof. Left to the reader. (Hint: take $\mathrm{wp}(S_2, q)$ and $\mathrm{wp}(b * S, q)$ as intermediate assertions.)

A natural question is whether the intersection or the union of the sets of assertions considered in the last two theorems is complete for $(\mathscr{S}, \mathscr{B})$. The intersection of these two sets is the set of recursive assertions which is incomplete by the results of the previous section. As we shall see in the next section the union of them is also incomplete for $(\mathscr{S}, \mathscr{B})$.

5. Arithmetical assertions

Let A be a subset of $\mathscr{N}^n$ $(n > 0)$. Recall that A is called $\Sigma_0^0(\Pi_0^0)$ if it is recursive. A is called $\Sigma_{k+1}^0(\Pi_{k+1}^0)$ where $k \geqslant 0$ if for some B being a $\Pi_k^0(\Sigma_k^0)$ subset of $\mathscr{N}^{n+1}$ and all

$\sigma \in \mathcal{N}^n$

$$\sigma \in A \leftrightarrow \exists x[(\sigma, x) \in B]$$
$$(\sigma \in A \leftrightarrow \forall x[(\sigma, x) \in B]).$$

A is called Δ_n^0 ($n \geqslant 0$) if it is both Σ_n^0 and Π_n^0. A is called *arithmetical* if it is Σ_n^0 for some n. It is well known that a set is Σ_1^0 iff it is recursively enumerable. This implies that a set is Δ_1^0 iff it is recursive. It is clear now what we mean by saying that a set of states (i.e. an assertion) or a set of k-tuples of states ($k>0$) is Σ_n^0, Π_n^0, Δ_n^0 or arithmetical.

The following easy facts about Σ_n^0, Π_n^0 or Δ_n^0 sets (see for them e.g. [14]) will be needed below.

Lemma. *Let $A, B \subseteq \mathcal{N}^k$ where $k>0$.*
(a) *A is Σ_n^0 iff $\mathcal{N}^k \backslash A$ is Π_n^0.*
(b) *if A and B are Σ_n^0 then so are $A \cup B$ and $A \cap B$.*
(c) *if A is Σ_n^0 then the set $\{(\sigma_1, \ldots, \sigma_{k-1}) : \exists \sigma_k[(\sigma_1, \ldots, \sigma_k) \in A]\}$ is Σ_n^0, as well.*
(d) *if A is Σ_n^0 then A is Σ_m^0, Π_m^0 and Δ_m^0 for any $m>n$.*
By $\Sigma_n^0(\Pi_n^0)(\Delta_n^0)$ we denote, from now on, the set of all $\Sigma_n^0(\Pi_n^0)(\Delta_n^0)$ assertions. We shall also use the following facts about Σ_n^0, Π_n^0 or Δ_n^0 assertions.

Lemma 5.1. *Let S be a normal program.*
(a) *if $p \in \Sigma_n^0$ ($n \geqslant 1$) then* $\mathrm{sp}(p, S) \in \Sigma_n^0$*, too.*
(b) *if $q \in \Pi_n^0 (n \geqslant 1)$ then* $\mathrm{wp}(S, q) \in \Pi_n^0$*, too.*
(c) *if $q \in \Sigma_n^0$ ($n \geqslant 2$) and S is a deterministic program then* $\mathrm{wp}(S, q) \in \Sigma_n^0$*, too.*

Proof. (a), (b) By definition. (c) Since S is deterministic, we have for all states σ

$$\sigma \in \mathrm{wp}(S, q) \leftrightarrow \forall \tau[\neg \sigma S \tau] \vee \exists \tau[\sigma S \tau \wedge \tau \in q]$$

which shows that $\mathrm{wp}(S, q) \in \Sigma_n^0$.

At first we shall prove the theorem we promised in the last section. We have to make a very mild assumption about $\mathcal{S}$, namely that it contains the program $[v_2 := 0]$ corresponding to the statement $v_2 := 0$. Thus for any two states σ and τ

$$\sigma[v_2 := 0]\tau \leftrightarrow \tau(v_2) = 0 \wedge \forall v \in \mathcal{V}[v \neq v_2 \rightarrow \tau(v) = \sigma(v)].$$

Theorem 5.1. *If $[v_2 := 0] \in \mathcal{S}$ and $\mathcal{A} = \Sigma_1^0 \cup \Pi_1^0$, then $\mathcal{A}$ is incomplete for $(\mathcal{S}, \mathcal{B})$.*

Proof. Let A be a Σ_1^0 nonrecursive subset of the even natural numbers and let B be a Π_1^0 nonrecursive subset of the odd natural numbers. Then $C = A \cup B$ is a set which is neither Σ_1^0 or Π_1^0. Indeed, we have for all $x \in \mathcal{N}$

$$x \in A \leftrightarrow x \in C \wedge x \text{ is even}$$

$$x \in B \leftrightarrow x \in C \wedge x \text{ is odd,}$$

so if C were Σ_1^0 then B would be Σ_1^0 and if C were Π_1^0 then A would be Π_1^0.

Let f be the following partial recursive function:

$$f(x) = \begin{cases} x \text{ if } x \notin B \\ \text{divergent otherwise} \end{cases}$$

and let S be a program which computes f. Thus $S = \{(\sigma, \sigma) : \sigma \in [\![\neg B]\!]\}$.

We prove now that

$$\mathrm{wp}(S, [\![A]\!]) \cap [\![\mathcal{N}]\!] = [\![C]\!]. \tag{1}$$

We have for any $n \in \mathcal{N}$

$$\begin{aligned} [\![n]\!] \in \mathrm{wp}(S, [\![A]\!]) &\leftrightarrow \forall\sigma[\neg[\![n]\!]S\sigma) \vee \exists\sigma[[\![n]\!]S\sigma \wedge \sigma \in [\![A]\!]] \\ &\leftrightarrow n \in B \vee \exists m[[\![n]\!]S[\![m]\!] \wedge [\![m]\!] \in [\![A]\!]] \\ &\leftrightarrow n \in B \vee ([\![n]\!] \in [\![A]\!]) \\ &\leftrightarrow n \in A \cup B \\ &\leftrightarrow n \in C, \end{aligned}$$

so indeed (1) holds.

For any state σ, if $\sigma(v_1) \in C$, then either $\sigma \in [\![C]\!]$, in which case by (1) $\sigma \in \mathrm{wp}(S, [\![A]\!])$, or σ is not of the form $[\![n]\!]$ for any n, in which case $\forall\tau[\neg\sigma S\tau]$, i.e. $\sigma \in \mathrm{wp}(S, [\![A]\!])$, as well. This shows that

$$\{\sigma : \sigma(v_1) \in C\} \subseteq \mathrm{wp}(S, [\![A]\!]). \tag{2}$$

C is a Σ_2^0 set, so for some D which is Π_1^0

$$x \in C \leftrightarrow \exists y\,[(x, y) \in D].$$

Let $p = \{\sigma : (\sigma(v_1), \sigma(v_2)) \in D\}$. Then $p \in \Pi_1^0$.

We show now that

$$\{p\}[v_2 := 0]; S\{[\![A]\!]\} \tag{3}$$

but for no assertion $q \in \Sigma_1^0 \cup \Pi_1^0$

$$\{p\}[v_2 := 0]\{q\} \quad \text{and} \quad \{q\}S\{[\![A]\!]\}. \tag{4}$$

At first observe that

$$\begin{aligned}&\mathrm{sp}(p,[v_2:=0])=\{\tau:\exists\sigma[\sigma\in p\wedge\sigma[v_2:=0]\tau]\}=\\&\{\tau:\exists\sigma[(\sigma(v_1),\sigma(v_2))\in D]\wedge\tau(v_2)=0\wedge\forall v\in\mathcal{V}L\,v\neq v_2\rightarrow\tau(v)=\sigma(v)]]\}=\\&\{\tau:\exists x[(\tau(v_1),x)\in D]\wedge\tau(v_2)=0\}=\\&\{\tau:\tau(v_1)\in C\wedge\tau(v_2)=0\}.\end{aligned}$$

Thus, by (2) $\mathrm{sp}(p,[v_2:=0])\subseteq\mathrm{wp}(S,[\![A]\!])$, which means that (3) holds.

Suppose now that for some assertion q (4) holds. Then $\mathrm{sp}(p,[v_2:=0])\subseteq q$ and $q\subseteq\mathrm{wp}(S,[\![A]\!])$, so

$$[\![C]\!]=\mathrm{sp}(p,[v_2:=0])\cap[\![\mathcal{N}]\!]\subseteq q\cap[\![\mathcal{N}]\!]\subseteq\mathrm{wp}(S,[\![A]\!])\cap[\![\mathcal{N}]\!]=[\![C]\!]$$

i.e. $[\![C]\!]=q\cap[\![\mathcal{N}]\!]$. Since C is not Σ_1^0 or Π_1^0, $q\notin\Sigma_1^0\cup\Pi_1^0$.

Using lemma 5.1 we can easily extend the results of section 4 to the sets Σ_n^0 and Π_n^0. Using lemma 5.1 (a) and following the proof of Theorem 4.1 we obtain that Σ_n^0 $(n\geqslant1)$ is complete for $(\mathcal{S},\mathcal{B})$ under the assumption that $\mathcal{S}$ is a class of normal programs. Also due to lemma 5.1 (b) we obtain that if $\mathcal{S}$ is a class of normal programs then Π_n^0 $(n\geqslant1)$ is complete for $(\mathcal{S},\mathcal{B})$. As a corollary we have: if $\mathcal{S}$ is a class of normal programs then the set of all arithmetical assertions is complete for $(\mathcal{S},\mathcal{B})$. If $\mathcal{S}$ is a class of deterministic normal programs then due to lemma 5.1 (c) for every $n\geqslant2$ Δ_n^0 and $\Sigma_n^0\cup\Pi_n^0$ are complete for $(\mathcal{S},\mathcal{B})$.

6. Completeness for recursive assertions

From the result of section 3 we learned that recursive assertions are not sufficient to obtain completeness. This fact is connected with a phenomenon (difficult to define formally) of loss of information about the program in question. Both the assertion method and the Hoare axiomatic method are concerned only with the input-output behaviour of a given program and not with the history of computation resulting from the execution of S. In this section we show that, by allowing the use of an 'auxiliary' coordinate, the set of recursive assertions is complete for while programs. This result is obtained by using that coordinate to provide information for limiting possible initial states to a finite set.

We extend the domain $\mathcal{V}$ to $\mathcal{V}^+$ by adding a fresh variable u, and we denote, for $\sigma\in\mathcal{U}$ and $x\in\mathcal{N}$, the extended state $\lambda v\,[\text{if } v\in\mathcal{V} \text{ then } \mathcal{N} \text{ then } \sigma(v) \text{ else } x]$ by $\sigma\&x$ and the extended state space by $\mathcal{U}^+$. Programs and assertions on the extended state space will in general be denoted by letters bearing a superscript $+$.

For $p\subseteq\mathcal{U}$, we write $p^\uparrow$ for $\{\sigma\&x:\sigma\in p,\,x\in\mathcal{N}\}$. We denote $\{b^\uparrow:b\in\mathcal{B}\}$ by $\mathcal{B}^+$.

Definition 6.1. S^+ is a *faithful extension* of S if

$$\forall\sigma, \tau\, \forall x[\sigma S\tau \leftrightarrow \exists y[\sigma\&x\, S^+\, \tau\&y]].$$

The relevance of this definition will become clear in the light of the following lemma, especially part (c).

Lemma 6.1.
(a) *If S_1^+ is a faithful extension of S_1 and S_2^+ is a faithful extension of S_2, then $S_1^+; S_2^+$ is a faithful extension of $S_1; S_2$.*
(b) If *S_3^+ is a faithful extension of S_3, then, for any $b \in \mathscr{B}$, $b^\uparrow * S_3^+$ is a faithful extension of $b * S_3$.*
(c) *If S^+ is a faithful extension of S, then $\forall p, q[\{p\}S\{q\} \leftrightarrow \{p^\uparrow\}S^+\{q^\uparrow\}]$.*

Proof. The verification of (a) and (b) is straightforward from the definitions of ';' and '*' and is therefore omitted. As for (c), suppose first $\{p\}S\{q\}$, that is, $\forall\sigma, \tau[(\sigma \in p \,\&\, \sigma S\tau) \rightarrow \tau \in q]$. We must prove $\{p^\uparrow\}S^+\{q^\uparrow\}$, that is, $\forall\sigma, x, \tau, y[(\sigma\&x \in p^\uparrow \wedge \sigma\&x\, S^+\, \tau\&y) \rightarrow \tau\&y \in q^\uparrow]$. If $\sigma\&x \in p^\uparrow$, then $\sigma \in p$. Also, if $\sigma\&x\, S^+\, \tau\&y$, then $\sigma S\tau$, since S^+ is a faithful extension of S. From $\sigma \in p$ and $\sigma S\tau$ we have $\tau \in q$, and therefore $\tau\&y \in q^\uparrow$. Next, suppose $\{p^\uparrow\}S^+\{q^\uparrow\}$. Let x be some arbitrary element of $\mathcal{N}$ (e.g. 0). If $\sigma \in p$, then $\sigma\&x \in p^\uparrow$. Also, if $\sigma S\tau$, then there exists a y such that $\sigma\&x\, S^+\, \tau\&y$, since S^+ is a faithful extension of S. From $\sigma\&x \in p^\uparrow$ and $\sigma\&x\, S^+\, \tau\&y$ we have $\tau\&y \in q^\uparrow$, and therefore $\tau \in q$.

Definition 6.2. A class of programs $(\mathscr{S}, \mathscr{B})$ is *well-founded* if $S = \bigcup_{k=0}^{\infty} \mathscr{S}_k$, where

$$\begin{cases} \mathscr{S}_0 \text{ is some class of recursive (i.e. } \Delta_0^0\text{) programs,} \\ \mathscr{S}_{k+1} \subseteq \mathscr{S}_k \cup \{S_1; S_2 : S_1, S_2 \in S_k\} \cup \{b * S_3 : b \in B, S_3 \in \mathscr{S}_k\}. \end{cases}$$

Remark. A well-founded class of programs consists of normal programs only.

Theorem 6.1. *If $(\mathscr{S}, \mathscr{B})$ is well-founded, then there exists a class of programs $\mathscr{S}^+$ such that*
(i) *each $S \in \mathscr{S}$ has a faithful extension $S^+ \in \mathscr{S}^+$;*
(ii) *if $\mathscr{A}$ is the set of recursive assertions from the extended state space of $\mathscr{S}^+$, then $\mathscr{A}$ is complete for $(\mathscr{S}^+, \mathscr{B}^+)$.*

Proof. For a state σ let $|\sigma|$ stand for $1 + \Sigma_{v \in \mathscr{V}} \sigma(v)$ and for an extended state α let $|\alpha|$ stand for $1 + \Sigma_{v \in \mathscr{V}^+} \alpha(v)$.

Since $(\mathscr{S}, \mathscr{B})$ is well-founded, we can write $\mathscr{S} = \bigcup_{k=0}^{\infty} \mathscr{S}_k$, as in definition 6.2. We first construct, for $S = \mathscr{S}_0$,

$$S^\uparrow = \{(\sigma\&x, \tau\&(x + |\sigma|) : \sigma, \tau \in \mathscr{U}, x \in \mathcal{N}, \sigma S\tau\}.$$

From this definition it is obvious that $S^\uparrow$ is a faithful extension of S and that $S^\uparrow$ is recursive.

The complete class $\mathcal{S}^+$ is then defined by $\mathcal{S}^+ = \bigcup_{k=0}^{\infty} \mathcal{S}_k^+$, where

$$\begin{cases} \mathcal{S}_0^+ = \{S^\uparrow : S \in \mathcal{S}_0\}, \\ \mathcal{S}_{k+1}^+ = \mathcal{S}_k^+ \cup \{S_1^+ ; S_2^+ : S_1^+, S_2^+ \in \mathcal{S}_k^+\} \cup \{b^+ * S_3^+ : b^+ \in \mathcal{B}^+, S_3^+ \in \mathcal{S}_k^+\}. \end{cases}$$

Clearly, $\mathcal{S}^+$ is closed under sequencing and repetition. It is trivially proved from this construction of $\mathcal{S}^+$ (by induction on k and using lemma 6.1 (a) and (b)) that indeed each $S \in \mathcal{S}$ has at least one faithful extension $S^+ \in \mathcal{S}^+$.

To establish the second claim consider the least class of programs $\mathcal{T}$ such that

(i) $\mathcal{S}_0^+ \subseteq \mathcal{T}$;

(ii) if $T, T_1, T_2 \in \mathcal{T}_1$, $b^+ \in \mathcal{B}^+$, then $T_1; T_2 \in \mathcal{T}$, $b^+ ** T \in \mathcal{T}$, $b^+ * T \in \mathcal{T}$.

Clearly, $\mathcal{S}^+ \subseteq \mathcal{T}$. To conclude the proof it is enough to show that for all $p^+ \in \mathcal{A}$ and $T \in \mathcal{T}$, $\text{sp}(p^+, T)$ is recursive, since, as in the proof of theorem 4.1, the forms $\text{sp}(p^+, S^+)$ and $\text{sp}(p^+, b^+ ** S^+)$ already suffice for providing the necessary intermediate assertions. A straightforward induction argument shows that for all $T \in \mathcal{T}$ and $\alpha, \beta \in \mathcal{U}^+$

$$(\Delta) \qquad [\alpha T \beta \wedge \alpha \neq \beta] \rightarrow |\alpha| < |\beta|.$$

Hence, if $T, T_1, T_2 \in \mathcal{T}$, $b^+ \in \mathcal{B}^+$ and $p^+ \subseteq \mathcal{U}^+$ then, for all $\alpha, \beta \in \mathcal{U}^+$,

(i) $\alpha T_1; T_2 \beta \leftrightarrow \exists \gamma[|\gamma| \leqslant |\beta| \wedge \alpha T_1 \gamma \wedge \gamma T_2 \beta]$;

(ii) $\alpha b ** T\beta \leftrightarrow \exists n < |\beta| \exists \gamma_0, \ldots, \gamma_n[|\gamma_0| < \cdots < |\gamma_n| = |\beta| \wedge \alpha = \gamma_0 \wedge \gamma_n = \wedge$
$\forall i < n[\gamma_i T \gamma_{i+1} \wedge \gamma_i \in b^+]]$;

(iii) $\alpha b^+ * T\beta \leftrightarrow \alpha b^+ ** T\beta \wedge \beta \notin b^+$;

(iv) $\beta \in \text{sp}(p^+, T) \leftrightarrow \exists \alpha[|\alpha| \leqslant |\beta| \wedge \alpha \in p^+ \wedge \alpha T \beta]$.

(Note that in (ii), for $\alpha \notin b^+$, we always have $n = 0$). (i), (ii) and (iii) together imply that all $T \in \mathcal{T}$ are recursive. This together with (iv) implies that if $p^+ \subseteq \mathcal{U}^+$ is recursive and $T \in \mathcal{T}$ then $\text{sp}(p^+, T)$ is recursive, which concludes the proof. Observe that the property (Δ) was crucial to the proof, as this allowed to bound the extential quantifiers in (i), (ii) and (iv).

7. Translation to Hoare's method

We shall briefly dwell on the question how these results can be translated to Hoare's method. Within Hoare's framework assertions are formulas, so it is awkward to talk about recursive assertions. A much more suitable class to consider is that of assertions which contain only bounded quantifiers. Call such formulas BQ formulas. For the purpose of the subsequent discussion we assume that the assertion language L is an extension of the language L_P of Peano arithmetic, such that each symbol of L which is not in L_P can be defined in L_P by a BQ formula.

Call a formula a Σ_1^0 formula if it is of the form $\exists z_1, \ldots, \exists z_n[\phi]$, where ϕ is a BQ formula. Observe that each BQ formula defines a Δ_0^0 set, but not every Δ_0^0 set is defined by a BQ formula. On the other hand, a set is Σ_1^0 iff it can be defined by a Σ_1^0 formula.

Let $\mathcal{S}$ be the class of while programs and let $\mathcal{S}_0 \subset \mathcal{S}$. We call a class of assertions $\mathcal{A}$ complete for $\mathcal{S}_0$ if for every p, $1 \in \mathcal{A}$ and $S \in \mathcal{S}_0$ $\models \{p\}S\{q\}$ iff $\{p\}S\{q\}$ can be proved in the usual Hoare proof system using only assertions from $\mathcal{A}$. The latter we denote by $\mathcal{A} \vdash \{p\}S\{q\}$.

The proof of theorem 3.3 shows that the class $\mathcal{BQ}$ of BQ formulas is incomplete for $\mathcal{S}$. On the other hand, as theorem 4.1 shows, the class of Σ_1^0 formulas is complete for $\mathcal{S}$.

In the translation from Floyd's to Hoare's method, the auxiliary coordinate used in the last section will appear as an auxiliary variable which preserves some information on the history of computation. A similar use of auxiliary variables has been made by Clint [3] to prove the correctness of programs with coroutines and by Owicki [13] for parallel programs.

A program S' is a faithful extension of a program S if its input-output behaviour on the variables of S is the same as that of S, but it can in addition use auxiliary variables. Observe that it is possible to construct the particular class of faithful extensions $\mathcal{S}^+$ corresponding to that defined in the proof of theorem 6.1. The proof of theorem 6.1 shows that $\mathcal{BQ}$ is complete for $\mathcal{S}^+$. To get completeness of $\mathcal{BQ}$ for $\mathcal{S}$ one has now to add to Hoare's system the following proof rule which links S^+ with S.

Rule. Let $S' \in \mathcal{S}$ be such that auxiliary variables appear in S' only in assignments $z := t$, where z is an auxiliary variable. If S is obtained from S' by deleting from S' all assignments to auxiliary variables and p and q do not contain auxiliary variables, then

$$\frac{\{p\}S'\{q\}}{\{p\}S\{q\}}.$$

This rule is also formulated in [13], where it was used in the proof system for the verification of parallel programs. Observe that by the construction each $S \in \mathcal{S}$ is obtained from $S^+ \in \mathcal{S}^+$ by deleting from S^+ all assignments to auxiliary variables. The same result can also be obtained by adding to Hoare's system the following curious rule

$$\frac{\{p\}S\{q\}}{\{p[e/z]\}S\{q\}}, \text{ where } z \text{ is an auxiliary variable.}$$

(As usual, $p[e/z]$ stands for the result of substituting e for z in p). Denote the resulting system by G. The above rule is obviously not sound in the usual technical sense, but it appears to be sound in the sense that only true sentences of the form $\{p\}S\{q\}$ can be derived in G, *provided* that p and q do not contain auxiliary variables.

To prove the last claim, assume for simplicity that only finitely many variables are used and that z is the only auxiliary variable. Call a sentence $\{p\}S\{q\}$ ***semi-valid*** if

$$\forall\sigma, \tau\ \forall x[\sigma\&x \in p \wedge \sigma S\tau \rightarrow \exists y[\tau\&y \in q]],$$

where assertions and programs are identified with their meanings and the notation is that of the last section.

It is easy to see that the axioms of G are semi-valid and that all proof rules of G preserve the semi-validity of sentences. So if $\{p\}S\{q\}$ can be derived in G, then it is semi-valid. If, in addition, p and q do not contain the auxiliary variable, then $\{p\}S\{q\}$ is true.

Due to the completeness of $\mathscr{BQ}$ for $\mathscr{S}^+$, to prove that $\mathscr{BQ}$ is complete for $\mathscr{S}$ with respect to G it is now sufficient to prove that $\mathscr{BQ} \vdash \{p\}S^+\{q\}$ implies $\mathscr{BQ} \vdash_G \{p\}S\{q\}$.

The proof proceeds by induction on the length of S and only the case of an assignment statement needs explanation. If S is of the form $v := t$, then S^+ is of the form $z := s;\ v := t$. $\mathscr{BQ} \vdash \{p\}z := s;\ v := t\{q\}$ implies that for some r $\mathscr{BQ} \vdash \{p\}z := s\{r\}$ and $\mathscr{BQ} \vdash \{r\}v := t\{q\}$. Thus $\models p \rightarrow r[s/z]$ and $\mathscr{BQ} \vdash_G \{r\}v := t\{q\}$. By the new rule $\mathscr{BQ} \vdash_G \{r[s/z]\}v := t\{q\}$, so by the consequence rule $\mathscr{BQ} \vdash_G \{p\}v := t\{q\}$.

Theorem 3.3 indicates a way to construct a program S such that $\mathscr{BQ} \vdash_G \{\mathbf{true}\}S\{\mathbf{false}\}$ (which implies that $\{\mathbf{true}\}S\{\mathbf{false}\}$ is true) but $\mathscr{BQ} \nvdash \{\mathbf{true}\}S\{\mathbf{false}\}$.

Auxiliary variables have also been used in [1] and [9] to obtain completeness of Floyd's method for recursive program schemes. The results of these papers indicate a way to extend the present notion of completeness of a set of assertions from the class of while programs to the class of recursive program schemes by allowing assertions from an extended state space. It would be interesting to investigate whether the completeness results proved in this paper can then be extended to the latter class.

Note added in proof. It was brought to our attention that similar results, in particular our Theorems 3.2, 4.1 and 5.1, have been proved in: I.A. Lomazova, O složnosti induktivnyh usloviĭ dlja verifikacii arifmetičeskih programm (On the complexity of inductive assertions for the verification of arithmetical programs), in: *Materialy Wsesojuznoĭ Naučnoĭ Studenčeskoĭ Konferencii*, Matematika (Novosibirsk State University, Novosibirsk, 1978) 85–94.

References

[1] K.R. Apt and L.G.L.T. Meertens, Completeness with finite systems of assertions for recursive program schemes, Report IW 84/77, Mathematical Centre, Amsterdam, (1977).

[2] J.W. De Bakker and L.G.L.T. Meertens, On the completeness of the inductive assertion method, *J. Comput. System Sci.* **11** (3) (1975) 323–257.

[3] M. Clint, Program proving: coroutines, *Acta Informat.* **2** (1) (1973) 50–62.

[4] S.A. Cook, Soundness and completeness of an axiom system for program verification, *SIAM J. Comput.* **7** (1) (1978) 70–90.

[5] E.M. Clarke, Programming language constructs for which it is impossible to obtain good Hoare-like axioms, *Proc. 4th ACM Symp. on Principles of Programming Languages*, (1977) 10–20.

[6] R.W. Floyd, Assigning meanings to programs, in: J.T. Schwartz, ed., Mathematical Aspects of Computer Science, *Proc. Symp. Applied Math.* (American Math. Soc., Providence) Vol. 19 (1967) 19–32.

[7] G.A. Gorelick, A complete axiomatic system for proving assertions about recursive and non-recursive programs, Technical Report 75, University of Toronto (1975).

[8] A. Grzegorczyk, A. Mostowski and C. Ryll-Nardzewski, The classical and the ω-complete arithmetic, *J. Symbolic Logic* **23** (1958) 1888–205.

[9] D. Harel, A. Pnueli and J. Stavi, Completeness issues for inductive assertions and Hoare's method, Technical Report, Tel-Aviv University (1976).

[10] C.A.R. Hoare, An axiomatic basis for programming language constructs, *C. ACM* **12** (1969) 576–580.

[11] Z. Manna, The correctness of programs, *J. Comput. System Sci.* **3** (1969) 119–127.

[12] D. Matuszek, The case for the assert statement, SIGPLAN Notices (1976) 36–37.

[13] S. Owicki, A consistent and complete system for the verification of parallel programs, *Proc. 8th ACM Symp. Theory Comput.* (1976) 73–86.

[14] J.R. Shoenfield, Mathematical logic (Addison-Wesley, New York, 1967).

[15] M. Wand, A new incompleteness result for Hoare's system, *J. ACM* **25** (1) (1978) 168–175.