stichting
mathematisch
centrum

AFDELING INFORMATICA IW 92/77 NOVEMBER
(DEPARTMENT OF COMPUTER SCIENCE)

K.R. APT, J.A. BERGSTRA & L.G.L.T. MEERTENS

RECURSIVE ASSERTIONS ARE NOT ENOUGH - OR ARE THEY?

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

AMS(MOS) subject classification scheme (1970): 68A05

ACM-Computing Reviews-categories: 5.24

Recursive assertions are not enough - or are they? [*]

by

K.R. Apt, J.A. Bergstra[**] & L.G.L.T. Meertens

ABSTRACT

Call a set of assertions A *complete* (with respect to a class of programs S) if for any $p, q \in A$ and $S \in S$, whenever $\{p\} S \{q\}$ holds, then all intermediate assertions can be chosen from 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 <u>true</u> and <u>false</u> 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.

KEY WORDS & PHRASES: *partial correctness, intermediate recursive assertions, completeness, while programs.*

[*] This report will be submitted for publication elsewhere

[**] Institute of Applied Mathematics and Computer Science
University of Leiden

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 (FLOYD [5]) and the axiomatic method of Hoare (HOARE [6]). 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* assertions p and q. The precondition p will usually be some simple condition on the input 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. MATUSZEK [10]) where assert B, with B some 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 by MANNA [9], DE BAKKER & MEERTENS [2] and APT & MEERTENS [1]. 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 $\{\underline{\text{true}}\}\ \text{skip}\{p\}$ 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 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 $H \vdash \{\underline{\text{true}}\}\ i := n;\ P_n\ \{\underline{\text{false}}\}$, where P_n stands for the n-th program in $\mathcal{S}$ according to some enumeration. Such an S diverges for input $i = n$ iff $H \nvdash \{\underline{\text{true}}\}\ i := n;\ P_n\ \{\underline{\text{false}}\}$. S has itself a number, say n_S. Now $\{\underline{\text{true}}\}\ i := n_S;\ P_{n_S}\{\underline{\text{false}}\}$ holds iff $P_{n_S} = S$ diverges for input $i = n_S$, i.e., iff $H \nvdash \{\underline{\text{true}}\}\ i := n_S;\ P_{n_S}\{\underline{\text{false}}\}$, so H is not complete.

A way to overcome this inherent weakness of the axiomatic method has been indicated in COOK [3]. 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

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

This rule by itself still leaves room for rather inessential forms of incompleteness. For example, WAND [12] exhibits a particular structure for which the necessary intermediate assertions are not first-order definable. This problem had been taken care of 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 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 [6] extended this result to a class of programs with recursive procedures. (Following CLARKE [4], to prove these results one could also replace the above two requirements by the single one that the weakest precondition be definable.)

CLARKE [4] 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., DE BAKKER & MEERTENS [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, $V = \{v_1, v_2, \ldots\}$ stands for a finite, nonempty set of "variables". A *state* is a (total) mapping $V \to N$, where N denotes the set of natural numbers. Letters $\sigma, \tau, \ldots$ are used for states. U denotes the set of all states. A *program* is a binary relation over the state space U, i.e. a set of pairs of (initial and final) states. An *assertion* is a subset of U. Programs are denoted by $S, S_1, S_2, \ldots$, and assertions by $p, q, r, \ldots$.

The fact that V is finite is merely a matter of convenience. With suitably amended definitions, all theorems remain valid if 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 $\forall\sigma,\tau[(\sigma\in p \wedge \sigma S\tau) \rightarrow \tau\in q]$;

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

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

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

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

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

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

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:

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

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

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 N into N. We say that S *computes* f if

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

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

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

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

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

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

We adopt now the following completeness definition:

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

(i) $p \cap b \in A$;

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

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

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

It may seem that we unduly omitted conditional statements from our definition. Note, however, that if A is complete for (S,B) and (informally) $\{p\}$ "if b then S_1 else S_2 fi" $\{q\}$, with $p, q \in A$, $b \in B$ and $S_1,S_2 \in S$, then the necessary intermediate assertions $p \cap b$ and $p \cap \neg b$ are already elements of 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 (GRZEGORCZYK & al. [7]):

THEOREM 3.1. *There exist two disjoint recursively enumerable subsets* X *and* Y *of* 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 $n \in N$ and $A \subseteq N$:

$$[n] = \{\sigma\colon \sigma(v_1) = n\};$$
$$[A] = \{\sigma\colon \sigma(v_1) \in A\};$$
$$[\![n]\!] = \lambda v{\in}V[\text{if } v = v_1 \text{ then } n \text{ else } 0];$$
$$[\![A]\!] = \{[\![n]\!]\colon n \in A\}.$$

Note that $[n]$, $[A]$ and $\llbracket A \rrbracket$ are assertions, whereas $\llbracket n \rrbracket$ is a state.

First, an incompleteness result is proved that is based on a construction involving repetition. We must assume something about the class $\mathcal{B}$, namely that it contains some conditionals of a very simple nature.

THEOREM 3.2. *If* $\mathcal{A}$ *is a set of recursive assertions and* $\{[0],[1]\} \subseteq \mathcal{B}$, *then* $\mathcal{A}$ *is incomplete for* $(\mathcal{S},\mathcal{B})$.

Proof. We exhibit the existence of a program $S \in \mathcal{S}$ such that $\{[0]\} \neg [1] * S\{\emptyset\}$, but, for no recursive assertion r, $[0] \subseteq r$, $\{r \cap \neg [1]\}\ S\{r\}$ and $r \cap [1] \subseteq \emptyset$, thus contradicting the third requirement of the completeness definition. Observe that for this case the characterization of an invariant r can be simplified to $[0] \subseteq r \subseteq \neg [1]$ and $\{r\}S\{r\}$.

Assume (without loss of generality) that $0 \in X$ and $1 \in Y$. Let g be a total recursive function enumerating X without repetition (so $X = \{g(0), g(1), \ldots\}$). We may also assume that $g(0) = 0$. Define f by

$$f(n) = \begin{cases} g(k+1), \text{ where } n = g(k), \text{ if } n \in X, \\ 1 \text{ if } n \in Y, \\ \text{divergent otherwise.} \end{cases}$$

Clearly, f is a partial recursive function. Let S be a program computing f. Observe that the application of f on an argument from X, say $g(k)$, yields the value $g(k+1)$, which is again an element of X. So $\{[X]\}S\{[X]\}$. Moreover, $[0] \subseteq [X] \subseteq \neg [1]$ (since $1 \in Y$ and X and Y are disjoint). It follows that indeed $\{[0]\} \neg [1] * S\{\emptyset\}$.

Now assume $[0] \subseteq r \subseteq \neg [1]$ and $\{r\}S\{r\}$. From $\llbracket g(0) \rrbracket = \llbracket 0 \rrbracket \in [0] \subseteq r$ and $\{r\}S\{r\}$ we find $\llbracket g(1) \rrbracket \in r$, $\llbracket g(2) \rrbracket \in r, \ldots$, so $\llbracket X \rrbracket \subseteq r$. From $r \subseteq \neg [1]$ and $\{r\}S\{r\}$ we find $r \subseteq wp(S, \neg [1]) \subseteq \neg [Y]$, so $r \cap \llbracket Y \rrbracket \subseteq r \cap [Y] = \emptyset$. The set Z defined by $\llbracket Z \rrbracket = r \cap \llbracket \mathcal{N} \rrbracket$ satisfies $\llbracket X \rrbracket \subseteq \llbracket Z \rrbracket$ and $\llbracket Z \rrbracket \cap \llbracket Y \rrbracket = \emptyset$, so $X \subseteq Z$ and $Z \cap Y = \emptyset$. By theorem 3.1, Z is not recursive, so neither is r. □

Slightly surprising, a stronger incompleteness result (without additional assumption on $\mathcal{B}$) can be proved without explicit use of repetition.

(Of course, the property of $\mathcal{S}$ that it has the full power of recursion theory implicitly assumes repetition or its like.)

THEOREM 3.3. *If* $\mathcal{A}$ *is a set of recursive assertions, then* $\mathcal{A}$ *is incomplete for* $(\mathcal{S},\mathcal{B})$.

Proof. We exhibit the existence of programs $S_1,S_2 \in \mathcal{S}$ for which $\{\mathcal{U}\}S_1;S_2\{\emptyset\}$, but, for no recursive assertion r, $\{\mathcal{U}\}S_1\{r\}$ 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 $n \in \mathcal{N}$, $\llbracket n\rrbracket \in sp(\mathcal{U},S_1)$ iff $\exists m[f_1(m)=n]$, that is, iff $n \in X$. Since obviously $sp(\mathcal{U},S_1) \subseteq \llbracket \mathcal{N}\rrbracket$, $sp(\mathcal{U},S_1) = \llbracket X\rrbracket$.

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 $m \in \mathcal{N}$, $\llbracket m\rrbracket \in wp(S_2,\emptyset)$ iff $\forall n[\neg f_2(m)=n]$, that is, iff $n \notin Y$, so $wp(S_2,\emptyset) \cap \llbracket \mathcal{N}\rrbracket = \llbracket \neg Y\rrbracket$, implying $\llbracket \neg Y\rrbracket \subseteq wp(S_2,\emptyset)$. Since $X \cap Y = \emptyset$, $X \subseteq \neg Y$, so $\llbracket X\rrbracket \subseteq \llbracket \neg Y\rrbracket$. Now we have $sp(\mathcal{U},S_1) = \llbracket X\rrbracket \subseteq \llbracket \neg Y\rrbracket \subseteq wp(S_2,\emptyset)$, so clearly $\{\mathcal{U}\}S_1;S_2\{\emptyset\}$.

Now assume that $\{\mathcal{U}\}S_1\{r\}$ and $\{r\}S_2\{\emptyset\}$. Then $\llbracket X\rrbracket = sp(\mathcal{U},S_1) \subseteq r$ and $r \cap \llbracket Y\rrbracket \subseteq wp(S_2,\emptyset) \cap \llbracket Y\rrbracket = \emptyset$. The set Z defined by $\llbracket Z\rrbracket = r \cap \llbracket \mathcal{N}\rrbracket$ satisfies $X \subseteq Z$ and $X \cap Z=\emptyset$ 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 save the assert statement. For, if $\{p\}S_1;S_2\{q\}$ in the sense of total correctness, an intermediate r would have to satisfy

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

where the addition $\neg\, wp(S_2,\emptyset)$ expresses termination. Now, let Z be an arbitrary recursively enumerable, but not recursive, subset of N, and let S_1 and S_2, respectively, compute a total function with range Z and a partial function with domain Z. Clearly, $\{U\}S_1;S_2\{U\}$ in the sense of total correctness. Since $sp(U,S_1) = \neg wp(S_2,\emptyset) = [\![Z]\!]$, 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 $b \in B$, then
$b{*}{*}S$ is a normal program.

(b) If S is a normal program and p is recursively enumerable, then
$sp(p,S)$ is recursively enumerable.

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

PROOF.

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

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

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

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

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

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

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

Finally, suppose $\{p\}b*S\{q\}$ for some $S \in S$, $b \in B$ and $p, q \in A$. Take $r = sp(p,b**S)$. 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 $r \in A$. □

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

Proof. Left to the reader. (Hint: take $wp(S_2,q)$ and $wp(b*S,q)$ 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 (S,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 (S,B).

5. ARITHMETICAL ASSERTIONS

Let A be a subset of N^n $(n>0)$. Recall that A is called $\Sigma^0_0(\Pi^0_0)$ if it is recursive. A is called $\Sigma^0_{k+1}(\Pi^0_{k+1})$ where $k \geq 0$ if for some B being a $\Pi^0_k(\Sigma^0_k)$ subset of N^{n+1} and all $\sigma \in N^n$

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

A is called Δ^0_n $(n\geq 0)$ if it is both Σ^0_n and Π^0_n. A is called *arithmetical* if it is Σ^0_n for some n. It is well known that a set is Σ^0_1 iff it is recursively enumerable. This implies that a set is Δ^0_1 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^0_n, Π^0_n, Δ^0_n or arithmetical.

The following easy facts about Σ^0_n, Π^0_n or Δ^0_n sets (see for them e.g. SHOENFIELD [11]) will be needed below.

LEMMA. *Let* A, B $\subseteq N^k$ *where* k > 0.

(a) A *is* Σ^0_n *iff* $N^k \backslash A$ *is* Π^0_n.

(b) *if* A *and* B are Σ^0_n *then so are* $A \cup B$ *and* $A \cap B$.

(c) *if* A *is* Σ^0_n *then the set* $\{(\sigma_1,\ldots,\sigma_{k-1})\colon \exists\sigma_k[(\sigma_1,\ldots,\sigma_k) \in A]\}$ *is* Σ^0_n, *as well.*

(d) *if* A *is* Σ^0_n *then* A *is* Σ^0_m, Π^0_m *and* Δ^0_m *for any* m > n.

By $\Sigma^0_n(\Pi^0_n)(\Delta^0_n)$ we denote, from now on, the set of all $\Sigma^0_n(\Pi^0_n)(\Delta^0_n)$ assertions. We shall also use the following facts about Σ^0_n, Π^0_n or Δ^0_n assertions.

LEMMA 5.1. *Let* S *be a normal program.*

(a) *if* p *is a* Σ^0_n *assertion* (n≥1) *then* sp(p,S) *is* Σ^0_n, *too.*

(b) *if* q *is a* Π^0_n *assertion* (n≥1) *then* wp(S,q) *is* Π^0_n, *too.*

(c) *if* q *is a* Σ^0_n *assertion* (n≥2) *and* S *is a deterministic program then* wp(S,q) *is* Σ^0_n, *too.*

Proof. (a), (b) By definition.

(c) Since S is deterministic, we have for all states σ

$$\sigma \in wp(S,q) \leftrightarrow \forall\tau(\neg\sigma S\tau) \vee \exists\tau(\sigma S\tau \wedge \tau\epsilon q)$$

which shows that wp(S,q) is Σ^0_n.

At first we shall prove the theorem we promised in the last section. We have to make a very mild assumption about 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

$$\sigma[v_2{:=}0]\tau \leftrightarrow \tau(v_2) = 0 \wedge \forall i{\le}n_0[i \neq 2 \rightarrow \tau(v_i) = \sigma(v_i)].$$

<u>THEOREM 5.1</u>. *If* $[v_2:=0] \in S$ *and* $A = \Sigma^0_1 \cup \Pi^0_1$ *then* A *is incomplete for* (S,B).

Proof. Let A be a Σ^0_1 nonrecursive subset of the even natural numbers and let B be a Π^0_1 nonrecursive subset of the odd natural numbers. Then $C = A \cup B$ is a set which is neither Σ^0_1 or Π^0_1. Indeed, we have for all $x \in N$

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

so if C were Σ^0_1 then B would Σ^0_1 and if C were Π^0_1 then A would be Π^0_1.

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 $S = \{(\sigma,\sigma)\colon \sigma \in \llbracket \neg B \rrbracket\}$.

We prove now that

$$wp(S,\llbracket A \rrbracket) \cap \llbracket N \rrbracket = \llbracket C \rrbracket. \qquad (1)$$

We have for any $n \in N$

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

so indeed (1) holds.

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

$$\{\sigma:\sigma(v_1)\in C\} \subseteq wp(S,\llbracket A \rrbracket). \qquad (2)$$

C is a Σ^0_2 set, so for some D which is Π^0_1

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

Let $p = \{\sigma : (\sigma(v_2), \sigma(v_2)) \in D\}$. Then p is a Π^0_1 assertion.

We show now that

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

but for no assertion $q \in \Sigma^0_1 \cup \Pi^0_1$

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

At first observe that

$$\begin{aligned}
&sp(p,[v_2 := 0]) = \{\tau : \exists\, \sigma[\, \sigma \in p \wedge \sigma[v_2 := 0]\tau\,]\} = \\
&\{\tau : \exists \sigma[(\sigma(v_1), \sigma(v_2)) \in D] \wedge \tau(v_2) = 0 \wedge \forall i < n_0 [i \neq 2 \rightarrow \tau(v_i) = \sigma(v_i)]]\} = \\
&\{\tau : \exists x[(\tau(v_1), x) \in D] \wedge \tau(v_2) = 0\} = \\
&\{\tau : \tau(v_1) \in C \wedge \tau(v_2) = 0\}.
\end{aligned}$$

Thus by (2) $sp(p,[v_2 := 0]) \subseteq wp(S, [\![A]\!])$ which means that (3) holds. Suppose now that for some assertion q (4) holds. Then $sp(p,[v_2 := 0]) \subseteq q$ and $q \subseteq wp(S, [\![A]\!])$, so

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

i.e. $[\![C]\!] = q \cap [\![N]\!]$.
Since C is not Σ^0_1 or Π^0_1, $q \notin \Sigma^0_1 \cup \Pi^0_1$. □

Using the lemma 5.1 we can easily extend results of the section 4 to the sets Σ^0_n and Π^0_n. Using lemma 5.1 (a) and following the proof of the theorem 4.1 we obtain that Σ^0_n $(n \geq 1)$ 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^0_n $(n \geq 1)$ 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 $n \geq 2$ Δ^0_n and $\Sigma^0_n \cup \Pi^0_n$ is complete for $(\mathcal{S}, \mathcal{B})$.

6. COMPLETENESS FOR RECURSIVE ASSERTIONS

From the result of the 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 whole 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 keep track of the history of computation.

We extend the domain $\mathcal{V}$ by adding a fresh variable u, and we denote, for $\sigma \in \mathcal{U}$ and $x \in \mathcal{N}$, the extended state $\lambda v[\text{if } v \in \mathcal{V} \text{ then } \sigma(v) \text{ else } x]$ 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 $\{\sigma\&x\colon \sigma\in p,\ x\in\mathcal{N}\}$. We denote $\{b^{\uparrow}\colon b\in\mathcal{B}\}$ by $\mathcal{B}^+$.

<u>DEFINITION 6.1.</u> 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).

<u>LEMMA 6.1.</u>

(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* $b \in \mathcal{B}$, $b^{\uparrow} * S_3^+$ *is a faithful extension of* $b * S_3$.

(c) *If* S^+ *is a faithful extension of* S*, then*

$$\forall p,q[\{p\}S\{q\} \leftrightarrow \{p^{\uparrow}\}S^+\{q^{\uparrow}\}].$$

<u>Proof</u>. 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 $(\mathcal{S},\mathcal{B})$ is *well-founded* if $\mathcal{S} = \bigcup_{k=0}^{\infty} \mathcal{S}_k$, where

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

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

THEOREM 6.1. *If* $(\mathcal{S},\mathcal{B})$ *is well-founded, then there exists a class of programs* $\mathcal{S}^{+}$ *such that*

(i) *each* $S \in \mathcal{S}$ *has a faithful extension* $S^{+} \in \mathcal{S}^{+}$*;*

(ii) *if* A *is the set of recursive assertions from the extended state space of* $\mathcal{S}^{+}$*, then* A *is complete for* $(\mathcal{S}^{+},\mathcal{B}^{+})$.

Proof. We construct $\mathcal{S}^{+}$ by using the fresh variable of the extended state, intuitively speaking, to keep track of the history of computation (which notion, however, is not properly definable in a relational framework). Let $<\sigma_1,\ldots,\sigma_n>$ stand for an encoding in $\mathcal{N}$ of the sequence $\sigma_1,\ldots,\sigma_n$, $n \geq 0$, $\sigma_i \in \mathcal{U}$, with the following properties:

(i) the set $\{<\sigma_1,\ldots,\sigma_n>\colon n \geq 0,\ \sigma_i \in \mathcal{U}\}$ of all encodings is exactly $\mathcal{N}$;

(ii) for $x \in \mathcal{N}$, the function $|x| = n$, where $x = <\sigma_1,\ldots,\sigma_n>$, is total recursive;

(iii) for $n \in \mathcal{N}$ and $\sigma \in \mathcal{U}$, the function $x^{\frown}\sigma = <\sigma_1,\ldots,\sigma_n,\sigma>$, where $x = <\sigma_1,\ldots,\sigma_n>$, is total recursive;

(iv) for $y \in \mathcal{N}$, with $|y| \geq 1$, the functions $\mathrm{pre}(y) = x$ and $\mathrm{last}(y) = \sigma$, where $y = x^{\frown}\sigma$, are total recursive.

The actual construction of such an encoding is left to the reader. Only properties (i)-(iv) are of relevance for our purpose.

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

$$S^{\uparrow} = \{(\sigma\&x,\ \tau\&x^\frown\sigma) \colon \sigma,\tau \in \mathcal{U},\ x \in \mathcal{N},\ \sigma S\tau\}.$$

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

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

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 $S^+ \in \mathcal{S}^+$. The proof of the second part of the claim of the theorem is more intricate. It is tackled by proving that, for arbitrary $p^+ \subseteq U^+$, $b^+ \in \mathcal{B}^+$ and $S^+ \in \mathcal{S}^+$, $sp(p^+,S^+)$ and $sp(p^+,b^+{*}{*}S^+)$ are recursive if p^+ is recursive. Since these forms already suffice for providing the necessary intermediate assertions, as in the proof of theorem 4.1, the claim then follows.

In the sequel of the proof p^+ stands for an arbitrary *recursive* assertion $\subseteq \mathcal{U}^+$.

First we prove a lemma that, in spite of its seeming simplicity, contains the essential step:

LEMMA 6.2. *If, for some* $S^+ \in \mathcal{S}^+$, $sp(p^+,S^+)$ *is recursive for all recursive* p^+, *then for all* $b^+ \in \mathcal{B}^+$, $sp(p^+,b^+{*}{*}S^+)$ *is recursive.*

Proof. If $\tau\&y \in p^+$ then clearly $\tau\&y \in sp(p^+,b^+{*}{*}S^+)$. Also, if for some σ' and x'

$$\sigma'\&x' \in sp(p^+,b^+{*}{*}S^+) \wedge \sigma' \epsilon b \wedge \tau\&y \in sp(\{\sigma'\&x'\},\ S^+) \qquad (1)$$

then

$$\sigma'\&x'(S^+)\ \tau\&y, \text{ so } \tau\&y \in sp(p^+,b^+{*}{*}S^+).$$

We prove now that, conversely, if $\tau\&y \in sp(p^+,b^+{*}{*}S^+)$ then either $\tau\&y \in p^+$ or for some σ' and x' (1) holds.

Assume that $\tau\&y \in sp(p^+,b^+{*}{*}S^+)$. By the definition of sp, for some $\sigma\&x$ $\sigma\&x \in p$ and $\sigma\&x(b^+{*}{*}S^+)\tau\&y$. By the definition of $b^+{*}{*}S^+$ there exists a sequence $\sigma_0\&x_0,\ldots,\sigma_n\&x_n$ $(n\geq 0)$ such that

$$\sigma_0\&x_0 = \sigma\&x \wedge \sigma_n\&x_n = \tau\&y \wedge \forall i<n[\sigma_i\&x_i \in b^+ \wedge \sigma_i\&x_i(S^+)\sigma_{i+1}\&x_{i+1}].$$

Obviously each $\sigma_i\&x_i \in sp(p^+,b^+{*}{*}S^+)$. There are two possibilities.

If all $\sigma_i\&x_i$ are equal then $\tau\&y = \sigma\&x$, so $\tau\&y \in p^+$. Otherwise, let $\sigma_j\&x_j$ be the last extended state different from $\tau\&y$. Then we have $\sigma_j\&x_j(S^+)\tau\&y$. By the construction of S^+ y is of the form $x_j{}^\frown\sigma_j{}^\frown\nu_1{}^\frown\ldots{}^\frown\nu_i$ for some $i < |y|$. We also have $\sigma_j \in b$ and $\tau\&y \in sp(\{\sigma_j\&x_j\},S^+)$. Thus we have in fact, shown that

$$\tau\&y \in sp(p^+,b^+{*}{*}S^+) \leftrightarrow \tau\&y \in p^+ \vee \exists i<|y|[\sigma'\&x' \in sp(p^+,b^+{*}{*}S^+)$$
$$\wedge\ \sigma'\in b \wedge \tau\&y \in sp(\{\sigma'\&x'\},S^+),$$
$$\text{where } y = x'{}^\frown\sigma'{}^\frown\nu_1{}^\frown\ldots{}^\frown\nu_i \text{ for some } \nu_1,\ldots,\nu_i \in \mathcal{U}].$$

It is easily seen from this formulation, by induction on $|y|$, that the question $\tau\&y \in sp(p^+,b^+{*}{*}S^+)$ admits of a decision procedure, so $sp(p^+,b^+{*}{*}S^+)$ is recursive. □

We now proceed with the proof of the theorem, and use induction on k to show that for $S^+ \in \mathcal{S}_k^+$, $sp(p^+,S^+)$ (and, therefore, $sp(p^+,b^+{*}{*}S^+)$) is recursive.

(Basis) Let $S^+ \in \mathcal{S}_0^+$. By the construction of $\mathcal{S}_0^+$, S^+ is a faithful extension of some program $S \in \mathcal{S}_0$, and $\tau\&y \in sp(p^+,S^+)$ iff $|y| \geq 1 \wedge (\sigma\&x \in p^+ \wedge \sigma S\tau$, where $y = x^\frown\sigma)$. Since $\mathcal{S}_0$ consist of recursive programs only, $sp(p^+, S^+)$ is recursive.

(Induction Step) Let the recursivity of $sp(p^+,S^+)$ already have been established for all $S^+ \in \mathcal{S}_k^+$. Let now $S^+ \in \mathcal{S}^+_{k+1} = \mathcal{S}_k^+ \cup \{S_1^+;S_2^+\colon S_1^+,S_2^+ \in \mathcal{S}_k^+\} \cup \{b^+ * S_3^+\colon b^+ \in \mathcal{B}^+, \mathcal{S}_3^+ \in \mathcal{S}_k^+\}$.

Case A: $S^+ \in \mathcal{S}_k^+$. Then $sp(p^+,S^+)$ is already known to be recursive.

Case B: $S^+ = S_1^+;S_2^+$, where S_1^+ and $S_2^+ \in S_k^+$. Then $sp(p^+,S^+) = sp(sp(p^+,S_1^+),S_2^+)$, which is clearly recursive.

Case C: $S^+ = b^+*S_3^+$, where $S_3^+\in S_k^+$. Then $sp(p^+,S^+) = sp(p^+,b^+**S_3^+) \cap \neg b^+$, which, due to lemma 6.2, is recursive. □

We shall briefly dwell on the question how these results could be translated to Hoare's method. The use of auxiliary variables corresponds to the use of auxiliary variables that are not used by the program. 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. For the use of this notion in a formal deduction system this criterion has to be reformulated in syntactic terms. If we now add to the usual Hoare-like proof system the following proof rule

$$\frac{\{p\}S^+\{q\}}{\{p\}S\{q\}} \qquad \text{provided that q does not contain auxiliary variables,}$$

then for any recursive assertions p and q which do not contain auxiliary variables, we have $\models \{p\}S\{q\}$ if and only if $\vdash \{p\}S\{q\}$, where the latter can be proved using only recursive assertions. Examination of the proof of theorem 6.1 suggests that the same result can be obtained without introduction of faithful extensions by adding instead the curious rule

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

(As usual p[e/z] stands from the result of substituting e for z in p.) This 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 by its use, *provided* that p and q do not contain auxiliary variables.

This result is not too surprising if one looks at it from another point of view. We have proved in section 4 completeness for the class of Σ_1^0 assertions. Each Σ_1^0 set can be defined by a formula of the form $\exists z_1,\ldots,z_n[\varphi]$, where φ contains only bounded quantifiers. Call such a formula a Σ_1^0 formula. Assume now that $\models$ {p}S{q}, where p and q are recursive. By the Σ_1^0 completeness there exists a proof of {p}S{q} which uses only Σ_1^0 assertions. If in each of the assertions - apart from p and q - used in the proof we erase the existential quantifiers, we obtain a sequence of formulas which can easily be transformed into a proof in our new system. This sequence is the proof, apart from the places where the rule of consequence has been used. With not too much trouble the use of the rule of consequence can now be replaced by the use of our new rule. Theorem 3.3 indicates a way to construct a program S such that $\models$ {true} S {false}, but {true} S {false} can not be proved in the usual Hoare-like system using only recursive assertions. It can be proved using only recursive assertions with the help of the above proof rule.

Auxiliary variables have also been used in APT & MEERTENS [1] to show completeness of Floyd's method for recursive program schemes. The results of that paper 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.

REFERENCES

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

[2] DE BAKKER, J.W. & L.G.L.T. MEERTENS, *On the completeness of the inductive assertion method,* Journal of Computer and System Sciences, vol. 11, No. 3, pp. 323-357 (1975).

[3] COOK, S.A., *Axiomatic and interpretive semantics for an Algol fragment,* Technical Report no. 79, University of Toronto (1975).

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

[5] FLOYD, R.W., *Assigning meanings to programs*, in "*Mathematical Aspects of Computer Science*" (J.T. Schwartz, Ed.), pp. 19-32, Proceedings of a Symposium in Applied Mathematics, Vol. 19, American Math. Soc., Providence (1967).

[6] GORELICK, G.A., *A complete axiomatic system for proving assertions about recursive and non-recursive programs*, Technical Report no. 75, University of Toronto (1975).

[7] GRZEGORCZYK, A. & A. MOSTOWSKI & C. RYLL-NARDZEWSKI, *The classical and the ω-complete arithmetic*, Journal of Symbolic Logic 23, pp. 188-205 (1958).

[8] HOARE, C.A.R., *An axiomatic basis for programming language constructs*, C. ACM 12, pp. 576-580 (1969).

[9] MANNA, Z., *The correctness of programs*, Journal of Computer and System Sciences vol. 3, pp. 119-127 (1969).

[10] MATUSZEK, D., *The case for the assert statement*, SIGPLAN Notices, pp. 36-37, August (1976).

[11] SHOENFIELD, J.R., *Mathematical logic*, Addison-Wesley, New York (1967).

[12] WAND, M., *A new incompleteness result for Hoare's system*, 8th Annual Symposium on Theory of Computing, pp. 87-91, (1976).