Acta Informatica 15, 219–232 (1981)

Acta Informatica

Recursive Assertions and Parallel Programs

Krzysztof R. Apt

The Faculty of Economics, Erasmus University, P.O. Box 1738, Rotterdam, The Netherlands

Summary. We prove that recursive assertions are enough for proofs of parallel programs considered in Owicki and Gries [7]. In other words, we prove that for any parallel program S and recursive assertions p and q if $\{p\}\ S\{q\}$ is true under the standard interpretation in natural numbers then all intermediate assertions needed in the proof can be chosen recursive. Finally, we show that if auxiliary variables are used only as program counters then the above result does not hold.

1. Introduction

While discussing various proof systems for correctness of programs the natural question arises: how complicated are the intermediate assertions needed for a correctness proof of a given program? In our previous paper Apt et. al. [1] we studied the problem whether recursive (i.e. effectively computable) assertions are sufficient for proofs of (partial) correctness of while programs. We showed that there exists a while program S and recursive assertions p and q such that $\{p\}\ S\{q\}$ is true under the standard interpretation in natural numbers, but this cannot be proved in the usual Hoare's proof system using recursive intermediate assertions only. On the other hand we succeeded in showing that recursive assertions are sufficient for proofs of correctness of while programs if auxiliary variables are allowed to be used.

In this paper we consider a similar question within the context of parallel programs. We restrict our attention to the general parallel language (GPL) and the proof system for it studied in Owicki [5]. We whow that recursive assertions are enough for proofs of partial correctness of programs from GPL within Owicki's system. The same result holds for the class $\mathscr{BQ}$ of assertions – those which are built up using bounded quantifiers only.

The paper is organized as follows. You are now reading the introduction. In the next section we discuss Owicki's proof system. In section 3 we define an operational semantics for programs from GPL and in section 4 we give a proof

0001-5903/81/0015/0219/$02.80

of the relative completeness of the proof system. Finally in section 5, by a careful analysis of the proof in section 4, we show completeness with respect to recursive assertions. The last section of the paper is devoted to a comparison of Owicki's proof method with that of Lamport [4] in which auxiliary variables are used only as program counters. These two methods when used for proving partial correctness of GPL programs are equivalent – they are both sound and relatively complete. We prove that Lamport's method is incomplete with respect to recursive assertions thus showing that these two methods are not any longer equivalent when restricted to recursive assertions.

2. Preliminaries

In the paper in addition to the usual while programs we consider programs of the form $S_1 \| \ldots \| S_n$ denoting a parallel composition of programs $S_1, \ldots, S_n$. The component programs $S_1, \ldots, S_n$ are while programs but additionally, within the context of parallel composition, programs of the form **await** b **then** S, where S is a while program, are allowed. Informally, a program executes an **await**-statement iff with its turn to execute the boolean expression b is evaluated to **true.** S is then executed as an indivisible operation. We disallow the nested use of parallel composition and of the **await**-statement. The above programs are subsequently called GPL programs.

Let L be a first order extension of the language L_P of Peano arithmetic. We assume that L is interpreted in the domain of natural numbers in such a way that all symbols of L_P get assigned to it the standard meaning and the other symbols can be defined by formulae of L_P with bounded quantifiers only.

By assertion we mean a formula of L. The only non-logical symbols used in programs are those of L. By a state we mean a function from variables into natural numbers with a finite domain. States are denoted by letters σ, τ with possible subscripts; $\mathrm{dom}(\sigma)$ denotes the domain of σ.

The above (standard) interpretation of L embodies the usual interpretation of assertions and meaning of while programs. We write $\models p(\sigma)$ to denote the fact that the assertion p is true when its free variables lying in $\mathrm{dom}(\sigma)$ get assigned meaning provided by the state σ. For example $\models (x=0 \wedge z=z)((x, 0))$ holds. If for all states $\sigma \models p(\sigma)$ holds then we say that p is true.

The meaning of a while program S is a partial function $\mathscr{M}(S)$ from states to states defined in the usual way. If not all variables of S are in $\mathrm{dom}(\sigma)$ then $\mathscr{M}(S)(\sigma)$ is undefined.

Owicki [5] introduced a Hoare-like proof system for partial correctness of GPL programs. She proved soundness of the proof system for all interpretations and (relative) completeness with respect to the standard interpretation in the natural numbers. The latter result is not explicitly proved but it is an immediate consequence of the related completeness result proved is section 6 of Owicki [5] or in Owicki [6]. Since we are interested here only in the issue of completeness, we restricted our attention to the standard interpretation in the natural numbers.

The proof system studied in Owicki [5] and Owicki and Gries [7] is an extension of the usual Hoare-like proof system H for partial correctness of while programs.

Let H^+ denote the extension of H by the following rule
await rule

$$\frac{\{p \wedge b\}\ S\{q\}}{\{p\}\ \textbf{await}\ b\ \textbf{then}\ S\{q\}}$$

Before formulating the (meta) rule concerning parallel composition, it is useful to note the following lemma.

By a *normal* subprogram of a parallel program or its component we mean a subprogram which is not a proper subprogram of an **await**-statement. We write $\vdash_{H^+} \{p\}\,S\{q\}$ to denote the fact that $\{p\}\,S\{q\}$ can be proved in H^+ using (for the consequence rule) *any* assertions which are true.

Lemma 1. *Let S be a component of a parallel program. Let $S_1, \ldots, S_k$ be the list of all normal subprograms of S. Then $\vdash_{H^+} \{p\}\,S\{q\}$ iff there exist assertions* $\mathrm{pre}(S_i)$ *and* $\mathrm{post}(S_i)$ *for $i = 1, \ldots, k$ such that*

(a) *if S_i is* **await** b **then** R *then*
$\models \{\mathrm{pre}(S_i) \wedge b\}\ R\{\mathrm{post}(S_i)\}$;

(b) *the following assertions are true*

(i) $p \rightarrow \mathrm{pre}(S)$, $\mathrm{post}(S) \rightarrow q$,

(ii) $\mathrm{pre}(S_i) \rightarrow \mathrm{post}(S_i)[t/x]$ *if S_i is* $x := t$,

(iii) $\mathrm{pre}(S_i) \rightarrow \mathrm{pre}(S_j)$, $\mathrm{post}(S_j) \rightarrow \mathrm{pre}(S_l)$, $\mathrm{post}(S_l) \rightarrow \mathrm{post}(S_i)$
if S_i is S_j; S_l,

(iv) $\mathrm{pre}(S_i) \wedge b \rightarrow \mathrm{pre}(S_j)$, $\mathrm{pre}(S_i) \wedge \neg b \rightarrow \mathrm{pre}(S_l)$, $\mathrm{post}(S_j) \rightarrow \mathrm{post}(S_i)$,
$\mathrm{post}(S_l) \rightarrow \mathrm{post}(S_i)$ *if S_i is* **if** b **then** S_j **else** S_l **fi**,

(v) $\mathrm{pre}(S_i) \rightarrow \mathrm{post}(S_j)$, $\mathrm{post}(S_j) \wedge b \rightarrow \mathrm{pre}(S_j)$, $\mathrm{post}(S_j) \wedge \neg b \rightarrow \mathrm{post}(S_i)$
if S_i is **while** b **do** S_j **od.**

Proof. See (essentially) Owicki [5]. □

This lemma shows that when discussing proofs in H^+ it will be sufficient to restrict attention to assertions $\mathrm{pre}(S_i)$ and $\mathrm{post}(S_i)$ satisfying the conditions listed in (a) and (b).

Definition 1. The proofs of $\{p_1\}\,S_1\{q_1\}, \ldots, \{p_n\}\,S_n\{q_n\}$ in H^+ are *interference free* if for all normal subprograms R of S_i and R_1 of S_j $(i \neq j)$ such that R_1 is an **await**-statement or an assignment

$$\vdash_{H^+} \{\mathrm{pre}(R) \wedge \mathrm{pre}(R_1)\}\ R_1\{\mathrm{pre}(R)\}$$
$$\vdash_{H^+} \{\mathrm{post}(R) \wedge \mathrm{pre}(R_1)\}\ R_1\{\mathrm{post}(R)\}.$$

The above conditions state, informally speaking, that R_1 "preserves" the pre- and post-assertions of R.

We are finally in position to state the desired proof rule concerning parallel composition.

rule of parallel programs

$$\frac{\text{the proofs of } \{p_1\}\,S_1\{q_1\}, \ldots, \{p_n\}\,S_n\{q_n\} \text{ are interference free}}{\{p_1 \wedge \ldots \wedge p_n\}\ S_1 \| \ldots \| S_n\{q_1 \wedge \ldots \wedge q_n\}}.$$

The last rule needed in the proof system is that of reduction.

reduction rule

Let $\mathscr{AV}$ be a set of variables which do not appear free in p or q and appear in S only in assignments of the form $x:=t$ where $x \in \mathscr{AV}$. Let S' be obtained from S by deleting all assignments to the variables of $\mathscr{AV}$ and subsequently by replacing some of the subprograms of the form **await true then** $x:=t$ by $x:=t$. Then

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

We denote above proof system by O.

In the subsequent considerations we shall need to code finite sequences of natural numbers by natural numbers; $\langle a_1, \ldots, a_n \rangle$ will stand for a code of the sequence $a_1, \ldots, a_n$. If $a = \langle a_1, \ldots, a_n \rangle$ then by definition $a^\frown c = \langle a_1, \ldots, a_n, c \rangle$. $\langle\ \rangle$ denotes the code of the empty sequence. We shall say that a sequence $b_1, \ldots, b_n$ *majorizes* a sequence $a_1, \ldots, a_m$ if $a_1, \ldots, a_m$ is pointwise smaller or equal than a subsequence of $b_1, \ldots, b_n$. We shall implicitly assume various properties of the functions "$\langle \ldots \rangle$" and "$^\frown$" like their definability by expressions of L, injectivity and monotonicity in the sense that if $b_1, \ldots, b_n$ majorizes $a_1, \ldots, a_m$ then $\langle a_1, \ldots, a_m \rangle \leqq \langle b_1, \ldots, b_n \rangle$. The proofs of these properties can be found in Shoenfield [8].

3. Semantics for Parallel Programs

In order to determine the complexity of assertions used in proofs of GPL programs we have to dispose with a simple, elementary semantics of parallel programs. A suitable candidate is a slight variant of the operational semantics introduced in Hennessy and Plotkin [3].

We define first the relation $\langle S_1, \sigma \rangle \to \langle S_2, \tau \rangle$ for S_1, S_2 being components of a parallel program and states σ, τ. It is convenient to allow the empty program E. The intuitive meaning of $\langle S_1, \sigma \rangle \to \langle S_2, \tau \rangle$ is: executing S_1 one step in a state σ leads to a state τ with S_2 being remainder of S_1 still to be executed (S_2 is E if S_1 terminates in τ). We assume that for any program S $E; S = S; E = S$.

We define the above relation by the following clauses.

(i) $\langle x:=t, \sigma \rangle \to \langle E, \mathscr{M}(x:=t)(\sigma) \rangle$

(ii) $\langle$**if** b **then** S_1 **else** S_2 **fi**, $\sigma \rangle \to \langle S_1, \sigma \rangle$ if $\models b(\sigma)$

(iii) $\langle$**if** b **then** S_1 **else** S_2 **fi**, $\sigma \rangle \to \langle S_2, \sigma \rangle$ if $\models \neg b(\sigma)$

(iv) $\langle$**while** b **do** S **od**, $\sigma \rangle \to \langle S;$ **while** b **do** S **od**, $\sigma \rangle$ if $\models b(\sigma)$

(v) $\langle$**while** b **do** S **od**, $\sigma \rangle \to \langle E, \sigma \rangle$ if $\models \neg b(\sigma)$

(vi) $\langle$**await** b **then** $S, \sigma \rangle \to \langle E, \mathscr{M}(S)(\sigma) \rangle$ if $\models b(\sigma)$

(vii) if $\langle S_1, \sigma \rangle \to \langle S_2, \tau \rangle$ then
$\langle S_1; S, \sigma \rangle \to \langle S_2; S, \tau \rangle$.

By a *history* we mean a sequence $(i_1, \sigma_1), \ldots, (i_k, \sigma_k)$ where $k \geqq 0$, for each $j = 1, \ldots, k$ $i_j \in \{1, \ldots, n\}$ and σ_j is a state. The empty history is denoted by ε. We use the letter h with a subscript to denote a history; $h_1{}^\frown h_2$ denotes a concatenation of histories h_1 and h_2.

If S_1, S_2 are parallel programs then by induction on the length of h we define the relation

$\langle S_1, \sigma\rangle \xrightarrow{h} \langle S_2, \tau\rangle$.

(i) $\langle S_1, \sigma\rangle \xrightarrow{\varepsilon} \langle S_1, \sigma\rangle$

(ii) if $\langle S_i, \sigma\rangle \to \langle S_i', \tau\rangle$ then

$\langle S_1 \| \ldots \| S_n, \sigma\rangle \xrightarrow{(i,\sigma)} \langle S_1 \| \ldots \| S_{i-1} \| S_i' \| S_{i+1} \| \ldots \| S_n, \tau\rangle$

(iii) if $\langle S_1, \sigma\rangle \xrightarrow{h_1} \langle S_2, \sigma_0\rangle$ and $\langle S_2, \sigma_0\rangle \xrightarrow{h_2} \langle S_3, \tau\rangle$ then

$\langle S_1, \sigma\rangle \xrightarrow{h_1 \cap h_2} \langle S_3, \tau\rangle$.

Finally, we define the meaning of a parallel programs by putting
$\mathcal{N}(S_1 \| \ldots \| S_n)(\sigma) = \{\tau$: for some history $h \langle S_1 \| \ldots \| S_n, \sigma\rangle \xrightarrow{h} \langle \underbrace{E \| \ldots \| E}_{n \text{ times}}, \tau\rangle\}$.

Of course, we could have provided the above definition without using histories. The reasons for using them will become clear in the proof of the completeness theorem.

We extend $\mathcal{N}$ to provide a meaning for all GPL programs in a standard way. For a GPL program S $\mathcal{N}(S)$ is a total function from states into the power set of all states.

For a GPL program S and assertions p, q we say that $\{p\}\, S\{q\}$ is *true* ($\models \{p\}\, S\{q\}$) if

$$\forall \sigma, \tau [\models p(\sigma) \wedge \tau \in \mathcal{N}(S)(\sigma) \to \models q(\tau)].$$

4. Relative Completeness of *O*

Owicki [5] proved soundness of O. In particular if $\vdash_O \{p\}\, S\{q\}$ then $\models \{p\}\, S\{q\}$. The relative completeness of O (with respect to the set of all true assertions about natural numbers) means the converse implication: if $\models \{p\}\, S\{q\}$ then $\vdash_O \{p\}\, S\{q\}$. Before presenting a completeness proof we introduce the following useful notion.

Let S be a component of a parallel program and let S' be a normal subprogram of S. By induction on the structure of S we define a program **after**(S', S). Informally speaking, **after**(S', S) is a remainder of S still to be executed just after the execution of the subprogram S' terminated and **before**(S', S), defined by

$$\textbf{before}(S', S) \equiv S'; \textbf{after}(S', S),$$

is a remainder of S still to be executed just before the execution of the subprogram S' has started.

If $S' \equiv S$ (which is the case when S is an assignment or an **await**-statement) then **after**$(S', S) \equiv E$. Otherwise

(i) if S is **if** b **then** S_1 **else** S_2 **fi** then
after$(S', S) =$ **after**(S', S_i) where S' is a subprogram of S_i ($i = 1$ or 2),

(ii) if S is **while** b **do** S_1 **od** then
after$(S', S) =$ **after**(S', S_1); S,

(iii) if S is $S_1; S_2$ then if S' is a subprogram of S_1 then
after(S', S)=**after**$(S', S_1); S_2$
and otherwise
after(S', S)=**after**(S', S_2).

We now provide a proof of relative completeness of O which will be needed later. In the proof we make use of the fact that the proof system H is relatively complete for while programs. It is a special case of the completeness results proved by Cook [2] and Owicki [6].

The proof proceeds by induction on the structure of the programs. The only interesting case is that of parallel programs.

Assume that

(1) $\models \{p\}\, S\{q\}$

where S is of the form $S_1 \| \ldots \| S_n$. Let $x_1, \ldots, x_k$ be a list of all variables occurring in S. Denote $\langle x_1, \ldots, x_k\rangle$ by $\bar{x}$ and let z and u be some new variables.

We transform each program S_i $(i=1, \ldots, n)$ into another program S_i^* by replacing successively

(i) each assignment $y := t$ being a normal subprogram of S_i by
await true then $z := z^\frown\langle i, \bar{x}\rangle; y := t$,

(ii) each **await**-statement **await** b **then** R in S_i by
await b **then** $z := z^\frown\langle i, \bar{x}\rangle; R$,

(iii) each assignment $y := t$ within an **await**-statement by
$u := u + x_1 + \ldots + x_k + z;\ y := t$.

The last step is only needed for the proofs in the next section.

Denote $S_1^* \| \ldots \| S_n^*$ by S^*. We now prove

(2) $\vdash_O \{p\}\, z := \langle\rangle;\ u := 0;\ S^*\{q\}$.

The proof is quite long and takes the rest of this section.

Assume that

$$\langle S^*, \tau\rangle \xrightarrow{h} \langle S', \sigma\rangle$$

for some states τ, σ defined on $x_1, \ldots, x_k, z, u$, history h and parallel program S'. Each element of h is associated with an evaluation of a boolean expression or an execution of an assignment or **await**-statement. Let $(j_1, \sigma_1), \ldots, (j_m, \sigma_m)$ be the subsequence of h associated with the executions of an assignment or **await**-statement. We say that h is *coded* by the natural number $\langle\langle j_1, y_1\rangle, \ldots, \langle j_m, y_m\rangle\rangle$, where for each $i=1, \ldots, m$ $y_i = \langle\sigma_i(x_1), \ldots, \sigma_i(x_k)\rangle$.
Note that one number can code more than one history.

Let now R be a normal subprogram of S_i^* $(i=1, \ldots, n)$. Let pre(R) and post(R) be the assertions such that for all states σ defined on $x_1, \ldots, x_k, z, u$

$\models \text{pre}(R)(\sigma) \leftrightarrow \exists\, \tau, S_1', \ldots, S_n', h[\models (p \wedge z = \langle\rangle \wedge u = 0)(\tau)$ and
$\langle S^*, \tau\rangle \xrightarrow{h} \langle S_1' \| \ldots \| S_n', \sigma\rangle$,
where S_i' is **before**(R, S_i^*) and
h is a history coded by $\sigma(z)$]

$\models \text{post}(R)(\sigma) \leftrightarrow \exists\, \tau, S_1', \ldots, S_n', h[\models (p \wedge z = \langle\rangle \wedge u = 0)(\tau)$ and
$\langle S^*, \tau\rangle \xrightarrow{h} \langle S_1' \| \ldots \| S_n', \sigma\rangle$,
where S_i' is **after**(R, S_i^*) and
h is a history coded by $\sigma(z)$].

It can be shown that the above assertions exist. It is an immediate consequence of lemma 3 proved in the next section.

We now prove

(3) $\vdash_O \{\text{pre}(S_1^*) \wedge \ldots \wedge \text{pre}(S_n^*)\}\ S^*\{\text{post})S_1^*) \wedge \ldots \wedge \text{post}(S_n^*)\}$.

To this end we show that the pre- and post-assertions above defined satisfy the conditions (a) and (b) of lemma 1 concerning the proofs of $\{\text{pre}(S_i^*)\}\ S_i^*\{\text{post}(S_i^*)\}$ in $H^+(i=1,\ldots,n)$ and also those related to interference freedom. The former is a straightforward but tedious task and we leave it to the reader. The proof of interference freedom is less straightforward.

Let σ be a given state defined over $x_1, \ldots, x_k, z, u$ and let $i_1, \ldots, i_l$ be a subsequence of $1, \ldots, n$.

Call a list of programs $R_{i_1}, \ldots, R_{i_l}$ $(\sigma, i_1, \ldots, i_l)$-*reachable* of for some programs R_j for $j \in \{1, \ldots, n\} - \{i_1, \ldots, i_l\}$,

$$\langle S^*, \tau\rangle \xrightarrow{h} \langle R_1 \| \ldots \| R_n, \sigma\rangle$$

for some state τ such that $\models (p \wedge z = \langle\rangle \wedge u = 0)(\tau)$ and history h coded by $\sigma(z)$.

Merging Lemma. *If each* R_{i_j} *for* $j=1, \ldots, l$ *is* (σ, i_j)*-reachable then* $R_{i_1}, \ldots, R_{i_l}$ *is* $(\sigma, i_1, \ldots, i_l)$*-reachable.*

The proof is given in the appendix.

Let now R be a normal subprogram of S_i^* and let **await** b **then** R_1 be an **await**-statement from S_j^* $(i \neq j)$. We prove

(4) $\models \{\text{pre}(R) \wedge \text{pre}(\textbf{await } b \textbf{ then } R_1) \wedge b\}\ R_1\{\text{pre}(R)\}$

(5) $\models \{\text{post}(R) \wedge \text{pre}(\textbf{await } b \textbf{ then } R_1) \wedge b\}\ R_1\{\text{post}(R)\}$.

Assume

(6) $\models (\text{pre}(R) \wedge \text{pre}(\textbf{await } b \textbf{ then } R_1) \wedge b)(\sigma)$

for some state σ.

By the definition of the pre-assertions and by the merging lemma there exists a state τ and programs $S_1', \ldots, S_n'$ such that $\models (p \wedge z = \langle\rangle \wedge u = 0)(\tau)$ and $\langle S^*, \tau\rangle \xrightarrow{h} \langle S_1' \| \ldots \| S_n, \sigma\rangle$, where S_i is **before** (R, S_i), S_j is **before** (**await** b **then** R_1, S_j) and h is a history coded by $\sigma(z)$.

Suppose now that $\mathcal{M}(R_1)(\sigma) = \sigma_0$. By the definition of the "$\rightarrow$" relation

$$\langle S_1' \| \ldots \| S_n', \sigma\rangle \xrightarrow{(j,\sigma)} \langle S_1' \| \ldots \| S_{j-1}' \| \textbf{after}(\textbf{await } b \textbf{ then } R_1, S_j) \| S_{j+1}' \| \ldots \| S_n', \sigma_0\rangle$$

because $\models b(\sigma)$.

To show that $\models \text{pre}(R)(\sigma_0)$ it is now sufficient to show that $\sigma_0(z)$ codes the history $h^\frown(j, \sigma)$.

By the definition of S_j^* R_1 is of the form $u := u + x_1 + \ldots + x_k + z;\ z := z^\frown\langle j, \bar{x}\rangle;$ R_2 where z is not free in R_2. Thus

$$\sigma_0(z) = \mathcal{M}(z := z^\frown\langle j, \bar{x}\rangle)(\sigma),$$

so

$$\sigma_0(z)=\sigma(z)^\frown\langle j, y_j\rangle,$$

where $y_j=\langle\sigma(x_1), \ldots, \sigma(x_k)\rangle$, i.e. $\sigma_0(z)$ indeed codes $h^\frown(j, \sigma)$.

The proof of (5) is analogous and is left to the reader.

By the relative completeness of H we now can replace in (4) and (5) "$\models$" by "$\vdash_H$". Applying the await rule we get

$\vdash_{H^+}\{\text{pre}(R)\wedge\text{pre}(\textbf{await } b \textbf{ then } R_1)\}$ **await** b **then** $R_1\{\text{pre}(R)\}$

$\vdash_{H^+}\{\text{post}(R)\wedge\text{pre}(\textbf{await } b \textbf{ then } R_1)\}$ **await** b **then** $R_1\{\text{post}(R)\}$.

Since there are no assignments being normal subprograms of S_j^*, the above concludes the proof of interference freedom of the proofs of $\{\text{pre}(S_i^*)\}\ S_i^*\{\text{post}(S_i^*)\}$ in H^+ $(i=1, \ldots, n)$.

This proves (3). To prove (2) it is now sufficient to show

(7) $\quad\models p\wedge z=\langle\rangle\wedge u=0\rightarrow\text{pre}(S_1^*)\wedge\ldots\wedge\text{pre}(S_n^*)$

and

(8) $\quad\models\text{post}(S_1^*)\wedge\ldots\wedge\text{post}(S_n^*)\rightarrow q.$

The proof of (7) is obvious in view of the fact that by the definition of "$\rightarrow$" we have $\langle S^*, \sigma\rangle\xrightarrow{\varepsilon}\langle S^*, \sigma\rangle$.

To prove (8) we note first that (1) implies

(9) $\quad\models\{p\}\ S^*\{q\}.$

This is a consequence of the easy to prove fact that the converse of the reduction rule is sound.

Suppose now that

(10) $\quad\models(\text{post}(S_1^*)\wedge\ldots\wedge\text{post}(S_n^*))(\tau)$

for some state τ.

By the definition of the post-assertions and the merging lemma there exists a state σ such that $\models p(\sigma)$ and

$$\langle S^*, \sigma\rangle\xrightarrow{h}\underbrace{\langle E\|\ldots\|E, \tau\rangle}_{n\text{ times}},$$

where h is a history coded by $\tau(z)$.

By definition $\tau\in\mathcal{M}(S^*)(\sigma)$, so by (9) $\models q(\tau)$, which proves (8).

This proves (2). Using now the reduction rule we get from (2) $\vdash_O\{p\}\ S\{q\}$ which concludes the proof of relative completeness of O.

It is worthwile to note that the definition of interference freedom can be slightly simplified. The definition we use here is that of Owicki [5]. In Owicki and Gries [7] the second clause of interference freedom is replaced by

$$\vdash_{H^+}\{q_i\wedge\text{pre}(R_1)\}\ R_1\{q_i\}$$

thus reducing the number of formulae to be proved. Obviously our results are not affected by such a change of the definition.

5. Complete Assertion Classes

Let $\mathscr{A}$ be a class of assertions. We say that $\mathscr{A}$ is *complete* if for all $p, q \in \mathscr{A}$ and GPL programs S if $\models \{p\}\, S\{q\}$ then $\{p\}\, S\{q\}$ can be proved in O using only the assertion from $\mathscr{A}$.

The relative completeness of O means that the set of all assertions is complete. We now prove the following theorem showing completeness of two other classes of assertions.

Theorem. *The classes of recursive assertions and $\mathscr{BQ}$ are both complete.*

The proof takes the rest of this section. At first we show that the pre- and post-assertions defined in section 4 are recursive if p is. To this end we need a couple of lemmata.

Definition 2. We say that a program S' is *derived* from S^* if S' is of the form $S_1'' \| \ldots \| S_n''$, where for $i = 1, \ldots, n$ either $S_i'' \equiv \mathbf{before}(R, S_i^*)$ or $S_i'' \equiv \mathbf{after}(R, S_i^*)$ for a normal subprogram R of S_i.

Note that if for some τ, σ and h $\langle S^*, \tau\rangle \xrightarrow{h} \langle S', \sigma\rangle$ then S' is derived from S^*.

Lemma 2. *The relation V defined by*
$V(S', S'', \sigma, \tau, i, m) \equiv S'$ is derived from S^*,
$\langle S', \tau\rangle \xrightarrow{(i,\tau)} \langle S'', \sigma\rangle$ and (i, τ) is coded by m
is recursive.

Proof. To check whether S' is derived from S^* is straightforward. To determine whether

$$\langle S', \tau\rangle \xrightarrow{(i,\tau)} \langle S'', \sigma\rangle$$

for given S', S'', σ, τ and i is straightforward if (i, τ) is associated with an evaluation of a boolean expression or an execution of an assignment (which can be determined by an effective procedure). The only troublesome case is when (i, τ) is associated with an execution of an **await**-statement. This **await**-statement can be easily located. Suppose it is **await** b **then** R. We now have to check whether $\models b(\tau)$ and $\mathscr{M}(R)(\tau) = \sigma$ holds. The former can be obviously checked by an effective procedure but the latter is in general only recursively enumerable.

Here however, R is of a special form thanks to the clause (iii) of the transformation of S into S^* and the fact that S' is derived from S^*. In Apt et al. [1] we proved that for such programs R the relation $\mathscr{M}(R)(\tau) = \sigma$ is recursive.

Given S', S'', σ, τ, i, m such that S' is derived from S^* and $\langle S', \tau\rangle \xrightarrow{(i,\tau)} \langle S'', \sigma\rangle$, the relation "$(i, \tau)$ is coded by m" is clearly recursive. Namely m codes (i, τ) if $m = \langle\rangle$ and (i, τ) is associated with an evaluation of a boolean expression or $m = \langle \tau(x_1), \ldots, \tau(x_k)\rangle$ and (i, τ) is associated with an execution of an assignment or an **await**-statement.

This shows that the relation V is indeed recursive. □

Lemma 3. *The relation U defined by*
$U(S', S'', \sigma, \tau, h, m) \equiv S'$ is derived from S^*,
$\langle S', \tau\rangle \xrightarrow{h} \langle S'', \sigma\rangle$ and h is coded by m
is recursive.

Proof. We have
$U(S', S'', \sigma, \tau, h, m) \equiv S'$ is derived from S^* and

$$[h=\varepsilon \& S' \equiv S'' \& \sigma=\tau \& m=\langle\rangle]$$
$$\vee [h \neq \varepsilon \& \langle S', \tau\rangle \xrightarrow{(i,\tau)} \langle S_0, \sigma_0\rangle \& (i,\tau) \text{ is coded by } m_1 \&$$
$$U(S_0, S'', \sigma, \sigma_0, h_1, m_2), \text{ where}$$
$$1^\circ.\ h=(i,\tau)^\frown h_1,$$
$$2^\circ.\ \sigma_0 \text{ is the state from the first element of } h_1,$$
$$3^\circ.\ S_0 \text{ is uniquely determined by } S' \text{ and } (i,\tau),$$
$$4^\circ.\ \text{either } m_1=\langle\rangle \text{ and } m_2=m \text{ or } m_1=\langle a_1\rangle \text{ and}$$
$$m_2=\langle a_2, \ldots, a_l\rangle \text{ where } m=\langle a_1, \ldots, a_l\rangle].$$

Given S' and (i,τ) S_0 can be effectively determined. By lemma 2 the relation U is recursive. □

We now show that the assertions pre(R) and post(R) defined in section 4 are recursive.

At first note that the state τ there mentioned can be effectively computed from σ. If $\sigma(z)=\langle\rangle$ then $\tau=\sigma$. Otherwise the values of τ on $x_1, \ldots, x_k$ are determined by the first element of the sequence coded by $\sigma(z)$; $\tau(u)=0$; $\tau(z)=\langle\rangle$ and if $y \in \mathrm{dom}(\tau)-\{x_1, \ldots, x_k, u, z\}$ then $\tau(y)=\sigma(y)$. Thus for some recursive function f, $\tau=f(\sigma)$.

Next, there are only finitely many programs derived from S^*. Thus

$$\models \mathrm{pre}(R)(\sigma) \leftrightarrow \bigvee_{S' \text{ derived from } S^*} \exists h[\models (p \wedge z=\langle\rangle \wedge u=0)(f(\sigma)),$$
$$\langle S^*, f(\sigma)\rangle \xrightarrow{h} \langle S', \sigma\rangle, h \text{ is coded by } \sigma(z) \text{ and}$$
$$\text{the } i\text{-th component of } S' \text{ is } \mathbf{before}(R, S_i^*)]$$

and similarly for post(R).

To show that pre(R) and post(R) are recursive it is now enough due to lemma 3 to put a "bound" effectively depending on σ on all possible histories coded by $\sigma(z)$. Each such history is of the form

$$(j_1, \sigma_1)^\frown h_1^\frown \ldots {}^\frown (j_m, \sigma_m)^\frown h_m,$$

where

$$\sigma(z)=\langle\langle j_1, y_1\rangle, \ldots, \langle j_m, y_m\rangle\rangle$$

and for $i=1, \ldots, m$ $y_i=\langle\sigma_i(x_1), \ldots, \sigma_i(x_k)\rangle$.

All elements of histories h_i-s are associated with evaluations of boolean expressions. Thus the states mentioned in h_i are all equal to σ_i.

Clearly $\sigma_i(z) \leqq \sigma(z)$ and $\sigma_i(u) \leqq \sigma(u)$. Also for $y \in \mathrm{dom}(\sigma_i)-\{x_1, \ldots, x_k, z, u\}$ we have $\sigma_i(y)=\sigma(y)$. Let now τ_i for $i=1, \ldots, m$ be the state defined as follows: $\tau_i(z)=\sigma(z)$, $\tau_i(u)=\sigma(u)$ and $\tau_i(y)=\sigma_i(y)$ if $y \not\equiv z, u$. Then τ_i majorizes σ_i.

Note that the length of each h_i is bounded - each consecutive evaluation of a boolean expression has to correspond with a different substatement of S^*. Thus for $i=1, \ldots, m$ $|h_i| \leqq s$, where s is the number of all substatements of S^*.

This shows that each history h coded by $\sigma(z)$ such that $\langle S^*, f(\sigma)\rangle \xrightarrow{h} \langle S', \sigma\rangle$ is majorized by the following "maximal" history coded by $\sigma(z)$:

$$(j_1, \tau_1)^\frown (n, \tau_1)^{s\frown} \ldots {}^\frown (j_m, \tau_m)^\frown (n, \tau_m)^s.$$

$(n, \tau_i)^s$ stands here for concatenation of s copies of (n, τ_i). Obviously this maximal history can be effectively computed from σ.

This concludes the proof of recursiveness of the assertions pre(R) and post(R).

The asserted programs from clause (a) of lemma 1 and from interference freedom conditions can be proved using recursive assertions only. This is a

consequence of clause (iii) of the transformation of S into S^* and the corresponding completeness result proved in Apt et al. [1].

It follows that if S is a parallel program and p, q are recursive assertions such that $\models \{p\}\, S\{q\}$ then $\{p\}\, S\{q\}$ can be proved in O using recursive assertions only.

If S is a GPL program which is not of the form $S_1 \| \ldots \| S_n$ and p, q are recursive assertions such that $\models \{p\}\, S\{q\}$ then $\{p\}\, S\{q\}$ can also be proved in O using recursive assertions only. The proof follows the proof of the completeness of recursive assertions for while programs presented in Apt et al. [1]. Each while program S in question is transformed there (using clause (iii) of the transformation of S into S^*) into a program S' which can be proved correct in H with respect to the original recursive assertions using recursive assertions only. The latter is demonstrated by showing first that the relation $\mathscr{M}(S')(\sigma) = \tau$ is recursive.

Since we want to apply the same reasoning to GPL programs, we also have to prove that for every GPL program S the relation $\tau \in \mathscr{N}(S')(\sigma)$ is recursive. S' is defined by induction on the structure of S and the only new case is that of parallel programs. We put

$$S' \equiv z := \langle\rangle;\ u := 0;\ S^*.$$

We have to prove that the relation $\tau \in \mathscr{N}(z := \langle\rangle;\ u := 0;\ S^*)(\sigma)$ is recursive. Take for p the assertions **true**. Note that for all states σ, τ defined on $x_1, \ldots, x_k, u, z$

$\tau \in \mathscr{N}(z := \langle\rangle;\ u := 0;\ S^*)(\sigma)$

$\leftrightarrow \tau \in \mathscr{N}(S^*)(\sigma')$ where $\sigma'(z) = \langle\rangle$, $\sigma'(u) = 0$ and $\sigma'(y) = \sigma(y)$ if $y \not\equiv z, u$

$\leftrightarrow \sigma' = f(\tau) \,\&\, \models (\mathrm{post}(S_1^*) \wedge \ldots \wedge \mathrm{post}(S_n^*))(\tau)$

where f is the function defined after lemma 3.

The last equivalence follows from the merging lemma and the observation (proved by induction on the length of h) that if $\langle S^*, \sigma'\rangle \xrightarrow{h} \langle S', \tau_0\rangle$ for some S', h, τ_0 and σ' as above then h is coded by $\tau_0(z)$ and $\sigma' = f(\tau_0)$.

Hence the above relation is indeed recursive.

The rest of the proof is the same as in Apt et al. [1]. This concludes the proof of the completeness of recursive assertions.

The completeness of $\mathscr{BQ}$ follows from the observation that the assertions $\mathrm{pre}(R)$ and $\mathrm{post}(R)$ are in fact $\mathscr{BQ}$ if p is and that all other intermediate assertions are also $\mathscr{BQ}$. The proof requires a tedious but straightforward analysis of the above reasoning and is left to the reader.

6. Auxiliary Variables Used as Program Counters

Owicki suggested (private communication) a simpler proof of the relative completeness of O in which auxiliary variables are used only as program counters and not as histories of a computation. Such a use of auxiliary variables corresponds to Lamport's method of proving correctness of parallel programs (see Lamport [4]). This shows that Lamport's and Owicki's methods, when used for proving partial correctness of GPL programs, are in fact equivalent. It turns out however, that these methods are not equivalent when restricted to recursive assertions. In fact, parallel programs are not needed to show the above.

The use of program counters as auxiliary variables implies that each auxiliary variable used in the proof is bounded by some fixed in advance value. We now

prove that if only auxiliary variables of bounded size are allowed then recursive assertions are not complete for proofs of partial correctness of while programs. This result is a simple generalization of a theorem proved in Apt et al. [1] which states that recursive assertions are not complete for proofs in H of partial correctness of while programs.

Since we shall deal with while programs only we shall use the following special case of the reduction rule used in Owicki and Gries [7].

Rule of Auxiliary Variables

Let $\mathscr{AV}$ be set of variables which do not appear free in p or q and appear in S only in assignments of the form $x:=t$ where $x \in \mathscr{AV}$. Let S' be obtained from S by deleting all assignments to the variables of $\mathscr{AV}$. Then

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

If S and S' satisfy the condition of the above rule we say that S' is a reduction of S.

Denote H supplemented by the above rule by H'. The following lemma will be needed in the sequel.

Lemma 4. *If $\vdash_{H'} \{p\}\, S\, \{q\}$ then there exists a proof of $\{p\}\, S\, \{p\}$ in H' which uses the same (up to substitutions) assertions and in which the rule of auxiliary variables is used only once, namely as the last step in the proof.*

Proof. The proof proceeds by induction on the length of proofs.

Consider the case when the last rule applied in the proof is that of composition. Then S is of the form $S_1; S_2$ and for some assertion r both $\vdash_{H'} \{p\}\, S_1\, \{r\}$ and $\vdash_{H'} \{r\}\, S_2\, \{q\}$.

The proofs of $\{p\}\, S_1\, \{r\}$ and $\{r\}\, S_2\, \{q\}$ are both shorter than the proof of $\{p\}\, S\, \{q\}$. By the induction hypothesis there exist programs S_1' and S_2' such that for $i=1,2$ S_i is a reduction of S_i' and $\vdash_H \{p\}\, S_1'\, \{r\}$ and $\vdash_H \{r\}\, S_2'\, \{q\}$. We might now argue that by the composition rule $\vdash_H \{p\}\, S_1'; S_2'\, \{q\}$ so applying the rule of auxiliary variables we get the desired proof of $\{p\}\, S\, \{q\}$. The problem is that the last application of the rule of auxiliary variables is not legal if some auxiliary variables appearing in $S_1'(S_2')$ are also free in $q(p)$.

To remedy this we first modify S_1' and S_2'. Let $S_1''(S_2'')$ be obtained from $S_1'(S_2')$ by substituting the auxiliary variables occurring both in $S_1'(S_2')$ and and $q(p)$ by some fresh variables. The proofs of $\{p\} S_1'\, \{r\}$ and $\{r\}\, S_2'\, \{q\}$ in H can be transformed into the proofs of $\{p\}\, S_1''\, \{r\}$ and $\{r\}\, S_2''\, \{q\}$ in H by performing the substitutions corresponding to the modification of S_i' into $S_i''(i=1,2)$. By the composition rule we get now $\vdash_H \{p\}\, S_1''; S_2''\, \{q\}$. We can now apply the rule of auxiliary variables and we get the desired proof of $\{p\}\, S\, \{q\}$.

An analogous proof deals with the consequence rule. The other cases are trivial. □

In Apt et al. [1] we proved that there exist two while programs S_1 and S_2 such that $\models \{\mathbf{true}\}\, S_1; S_2\, \{\mathbf{false}\}$ but for no recursive assertion r both $\models \{\mathbf{true}\}\, S_1\, \{r\}$ and $\models \{r\}\, S_2 \{\mathbf{false}\}$. We now show that $\{\mathbf{true}\}\, S_1; S_2 \{\mathbf{false}\}$ cannot be

proved in H' using recursive assertions only and auxiliary variables whose values are all bounded, say by n.

Suppose otherwise. By lemma 4 there exist while programs S_1' and S_2' such that $\{\mathbf{true}\}\, S_1';\ S_2'\, \{\mathbf{false}\}$ can be proved in H using recursive assertions only and $S_1;\ S_2$ is a reduction of $S_1';\ S_2'$. Clearly, for some recursive assertion r both $\vdash_H \{\mathbf{true}\}\ S_1'\{r\}$ and $\vdash_H \{r\}\ S_2'\{\mathbf{false}\}$. By the soundness of $H \models \{\mathbf{true}\}\ S_1'\{r\}$ and $\models \{r\}\ S_2'\{\mathbf{false}\}$. Let $z_1, \ldots, z_k$ be all auxiliary variables appearing free in r and let

$$r_1 \equiv \exists z_1 \leqq n \ldots \exists z_k \leqq n\, r.$$

We have $\models r \rightarrow r_1$, since all auxiliary variables are bounded by n. So $\models \{\mathbf{true}\}\ S_1'\{r_1\}$ and by the soundness of the rule of auxiliary variables $\models \{\mathbf{true}\}\ S_1\{r_1\}$.

On the other hand it is easy to see that $\models \{r_1\}\ S_2'\{\mathbf{false}\}$ implies $\models \{r\}\ S_2\ \{\mathbf{false}\}$ which in turn implies $\models \{r_1\}\ S_2\ \{\mathbf{false}\}$, since none of the variables $z_1, \ldots, z_k$ appears free in S_2.

But r_1 is recursive so we reached a contradiction with the original choice of S_1 and S_2. We thus showed that the use of auxiliary variables as program counters results in an incompleteness of recursive assertions for while programs.

Appendix

Proof of Merging Lemma

The proof proceeds by induction on the length lh of the sequence coded by $\sigma(z)$. If $lh=0$ then all l computations assuring for $j=1, \ldots, l$ (σ, i_j)-reachability of R_{i_j} consist exclusively of evaluations of boolean expressions. A construction of a computation assuring $(\sigma, i_1, \ldots, i_l)$-reachability of $R_{i_1}, \ldots R_{i_l}$ is then a straightforward task.

Assume now that $lh \geqq 0$. Let h be a history coded by $\sigma(z)$. h is nonempty. Let σ_0 be the last state considered in h. σ_0 depends only on σ and $\sigma_0(z)$ codes a sequence of length $lh-1$. For some c $\sigma(z)=\sigma_0(z)^\frown c$.

Fix now j such that $1 \leqq j \leqq l$. Let h_{i_j} be a history related to the (σ, i_j)-reachability of R_{i_j}. h_{i_j} is coded by $\sigma(z)$. We have $h_{i_j}=h_1{}^\frown h_2$ for some h_1 coded by $\sigma_0(z)$. For some $R_1', \ldots, R_n'$

$$\langle S^*, \tau\rangle \xrightarrow{h_1} \langle R_1' \parallel \ldots \parallel R_n', \sigma_0\rangle$$

where τ is the state referred to in the (σ, i_j)-reachability of R_{i_j}. By definition R_{i_j}' is (σ_0, i_j)-reachable.

Thus for each $j=1, \ldots, l$ we found a program R_{i_j}' which is (σ_0, i_j)-reachable. By the induction hypothesis $R_{i_1}', \ldots, R_{i_l}'$ is $(\sigma_0, i_1, \ldots, i_l)$-reachable, say with a state τ and history h.

Exactly one element of each h_2 is associated with an execution of an assignment or **await**-statement. We can assume that this is always the first one and that for each $j=1, \ldots, l$ all other elements of h_2 are of the form (i_j, σ).

Each h_2 is coded by the same number, namely c, and consequently the first element of each h_2 is the same. Call it h'. Delete now this element from all histories

h_2. $R_{i_1}, \ldots, R_{i_l}$ is now $(\sigma, i_1, \ldots, i_l)$-reachable with the state τ and history $h \frown h'$ concatenated with the resulting sequence of all h_2-s. □

Acknowledgement. We thank the referee for suggesting various improvements.

References

1. Apt, K.R., Bergstra, J.A., Meertens, L.G.L.T.: Recursive assertions are not enough - or are they?, Theor. Comput. Sci. **8**, 73–87 (1979)
2. Cook, S.A.: Soundness and completeness of an axiom system for program verification, SIAM J. Comput. **7**, 70–90 (1978)
3. Hennessy, M.C.B., Plotkin, G.D.: Full abstraction for a simple programming language. In: Proc. 8th Symposium on Mathematical Foundations of Computer Science, pp. 108–120, Lecture Notes in Computer Science 74, Berlin Heidelberg New York: Springer 1979
4. Lamport, L.: Proving the correctness of multiprocess programs. IEEE Trans. Software Engrg., **3**, 125–143 (1977)
5. Owicki, S.: Axiomatic proof techniques for parallel programs. Computer Science Dept., Cornell University. PhD thesis 1975
6. Owicki, S.: A consistent and complete deductive system for the verification of parallel programs, In: Proc. 8th Annual Symposium on Theory of Computing, pp. 73–86 (1976)
7. Owicki, S., Gries, D.: An axiomatic proof technique for parallel programs, Acta Informat. **6**, 319–340 (1976)
8. Shoenfield, J.R.: Mathematical logic, Addison-Wesley 1967

Received October 24, 1979/October 1, 1980