Proving the Correctness of Coroutines without History Variables

Edmund M. Clarke (Duke University)

Abstract

We examine the question of whether history variables are necessary in formal proofs of correctness for coroutines. History variables are special variables which are added to a program to facilitate its proof by recording the execution history of the program. Such variables were first used by Clint in his paper "Program Proving: Coroutines." They have also been used by Owicki and Howard (concurrent programs) and by Apt (sequential programs). We argue that recording the entire history of a computation in a single set of variables is inconvenient and leads to extremely complicated proofs. We propose a modification of Clint's axiom system and a strategy for constructing proofs which eliminates the need for history variables in verifying simple coroutines. Examples are given to illustrate this technique of verifying coroutines and our axiom system is shown to be sound and relatively complete with respect to an operational semantics for coroutines. Finally, we discuss extensions of the coroutine concept for which history variables do appear to be needed; we also discuss the question of whether history variables are necessary in verifying concurrent programs.

1. Introduction

This paper examines some of the problems involved in developing an axiomatic proof system for a programming language with coroutines. In particular we investigate the question of whether history variables are necessary in proving partial correctness of coroutines. History variables are special variables which are added to a program to facilitate its proof by recording the sequence of states reached by the program during a computation; after the proof has been completed the history variables may be deleted. The use of such variables in correctness proofs was first suggested by Clint [CL73] in a paper entitled "Program Proving: Coroutines;" subsequently, history variables have been used by Owicki [OW75] and Howard [HO75] in verifying concurrent programs and by Apt [APT77] in verifying sequential programs. Owicki and Howard have conjectured that history variables are necessary for proofs of some concurrent programs.

The obvious power of history variables in program proofs stems from the large amount of information about a program's behavior which can be obtained by examining execution sequences. This power, however, is not available without a sacrifice. Program histories are much more difficult to manipulate in partial correctness assertions than simple program identifiers. Another, less obvious, disadvantage is that it is no longer possible to construct program proofs in a top-down manner in which only the input-output behavior of a statement is used to relate the statement to the remainder of the program. Instead it is necessary to consider the entire history of the program's execution in constructing proofs. Because of these disadvantages we believe that the use of history variables should be avoided whenever possible. We conjecture, in fact, that the success of program verification for actual programs will be inversely related to the frequency of situations in which it is necessary to reason about programs using execution histories.

In this paper we show that history variables are NOT needed in proving the correctness of simple coroutines. We give a modification of Clint's axiom system and a strategy for generating proofs in which the only auxiliary variables needed (in addition to the program identifiers) are simple program counters. Since the program counters have bounded magnitude, they may be encoded by 0,1-valued auxiliary variables. We illustrate our method of proving coroutines with examples and give a proof of soundness and relative completeness for our axiom system.[1] Finally, we discuss some extensions of the coroutine concept which do appear to require the use of history variables. We also discuss the question of whether a proof technique similar to the one presented in this paper can be used to avoid history variables in concurrent programs.

[1] Although this completeness proof has been briefly mentioned in another paper by the author [CK77a], this is the first detailed account of the proof to appear in print.

2. Coroutines

A coroutine will have the form:

coroutine Q_1, Q_2 end

Q_1 is the main routine; execution begins in Q_1 and also terminates in Q_1 (the requirement that execution terminate in Q_1 is not absolutely necessary but simplifies the axiom for coroutines). Otherwise Q_1 and Q_2 behave in identical manners. If an "exit" statement is encountered in Q_1, the next statement to be executed will be the statement following the last "resume" statement in Q_2. Similarly, the execution of a "resume" statement in Q_2 causes execution to be restarted following the last "exit" statement executed in Q_1. A simple example of a coroutine is:

```
coroutine
   while y≠z do
      y:=y+1; x:=x+y; exit;
      end,
   while true do
      y:=y-2; resume;
      y:=y+1; resume;
      end
   end
```

This example illustrates the use of "exit" and "resume" statements within while loops. Note that if x and y are 1 initially and z≥1, then the coroutine will terminate with $y=z^2$.

2.1 Axioms for Coroutines

In this section we give a set of axioms for coroutines and describe a technique for proving correctness of coroutines which is based on the use of auxiliary variables. This technique is different from the technique described by Clint [CL73], in that the auxiliary variables represent program counters (and therefore have bounded magnitude) rather than program histories.

C1. (Coroutines)

$$\frac{\{P'\}\ \text{exit}\ \{R'\} \vdash \{P \wedge b\}\ Q_1\ \{R\} \qquad \{R'\}\ \text{resume}\ \{P'\} \vdash \{P' \wedge b\}\ Q_2\ \{R'\}}{\{P \wedge b\}\ \text{coroutine}\ Q_1, Q_2\ \text{end}\ \{R\}}$$

provided no variable free in b is global to Q_1. (This axiom is a modification of of the one in [CL73]).

C2. (Exit)

$$\frac{\{P'\}\ \text{exit}\ \{R'\}}{\{P' \wedge C\}\ \text{exit}\ \{R' \wedge C\}}$$

provided that C does not contain any free variables that are changed by Q_2. (Here we assume that "exit" occurs in statement Q_1 of "coroutine Q_1, Q_2 end").

C3. (Resume)

$$\frac{\{R'\}\ \text{resume}\ \{P'\}}{\{R' \wedge C\}\ \text{resume}\ \{P' \wedge C\}}$$

provided that C does not contain any free variables that are changed in Q_1. (Here we assume that "resume" occurs in statement Q_2 of "coroutine Q_1, Q_2 end").

C4. (Auxiliary variables)

Let AV be a set of variables such that $x \in AV$ iff x appears in S' only in assignments y:=e with $v \in AV$. If P and Q are assertions which do not contain any free variables from AV and if S is obtained from S' by deleting all assignments to variables in AV, then

$$\frac{\{P\}\ S'\ \{Q\}}{\{P\}\ S\ \{Q\}}$$

(This axiom is essentially the same as the auxiliary variable axiom in [OW76]).

We illustrate the axioms with an example. We show that $\{x=1 \wedge y=1 \wedge z\geq 1\}\ A\ \{x=z^2\}$ where A ≡ "coroutine Q_1, Q_2 end" is the coroutine given in Section 2.1. Our strategy in carrying out the proof will be to introduce auxiliary variables to distinquish the various "exit" and "resume" statements from each other and then use axiom C4 to delete these unnecessary variables as the last step of the proof. Axiom C2 enables us to adapt the general exit assumption {P'} exit {R'} to a specific occurrence of an exit statement in Q_1. A similar comment applies to axiom C3 for the resume statement. We prove:

```
{x=1 ∧ y=1 ∧ z≥1}
   i:=0; j:=0;
   coroutine
      while y≠z do
         y:=y+1; x:=x+y;
         i:=1; exit;
         end,
      while true do
         y:=y-2; j:=1; resume;
         y:=y+1; j:=0; resume;
         end
      end
{x=z^2}
```

Note that two auxiliary variables are needed (one for each routine of the coroutine). The auxiliary variable j of the second routine is assigned a different value prior to each "resume" statement and is not changed by the first routine. Thus the value of j can be used in assertions to distinquish which of the resume statements has been most recently executed. The auxiliary variable i of the first routine has a dual function. This technique of adding auxiliary variables will be formally described in Section 5; however, the general pattern should be clear from the above example. To complete the proof we choose:

$P = \{x=1 \wedge y=1 \wedge z\geq 1 \wedge i=0 \wedge j=0\}$

$b = \{j=0\}$

$R = \{x=z^2\}$

$P' = \{(x=y^2-y+1 \wedge j=0 \wedge y\leq z)$
$\vee (x=y^2+2y+1 \wedge j=1 \wedge y\leq z-1)\}$

$R' = \{(x=y^2+3y+3 \wedge j=1 \wedge y\leq z-2)$
$\vee (x=y^2 \wedge j=0 \wedge y\leq z)\}$

The invariant for the while loop of the first routine is:

$INV_1 = \{(x=y^2+3y+3 \wedge j=1 \wedge y\leq z-2)$
$\vee (x=y^2 \wedge j=0 \wedge y\leq z)\}$

The invariant for the while loop of the second routine is:

$INV_2 = \{x=y^2-y+1 \wedge j=0 \wedge y\leq z\}$

Using axioms C2-C4 together with the axioms for the assignment statement and the while statement, it is possible to prove that:

(a) $\{P'\}$ exit $\{R'\} \vdash \{P \wedge b\}\ Q_1\ \{R\}$

and

(b) $\{R'\}$ resume $\{P'\} \vdash \{P' \wedge b\}\ Q_2\ \{R'\}$

both hold. For example, to prove (b) we assume $\{R'\}$ resume $\{P'\}$ and prove $\{P' \wedge b\}\ Q_2\ \{R'\}$. In order to prove $\{P' \wedge b\}\ Q_2\ \{R'\}$ we show that

(c) $P' \wedge b \rightarrow INV_2$

(d) $\{INV_2\}$

```
while true do
   y:=y-2; j:=1; resume;
   y:=y+1; j:=2; resume;
   end
```

$\{INV_2 \wedge \sim true\}$

(e) $INV_2 \wedge \sim true \rightarrow R'$ are true.

Steps (c) and (e) are easily verified. Step (d) follows from the while axiom and the sequence of assertions below:

(d1) assignment

$\{INV_2 \wedge true\}$ y:=y-2; j=1 $\{R' \wedge j=1\}$

(d2) resume

$\{R' \wedge j=1\}$ resume $\{P' \wedge j=1\}$

(d3) assignment

$\{P' \wedge j=1\}$ y:=y+1; j=0 $\{R' \wedge j=0\}$

(d4) resume

$\{R' \wedge j=0\}$ resume $\{P' \wedge j=0\}$

(d5) arithmetic

$P' \wedge j=0 \rightarrow INV_2$

Once (a) and (b) have been established, the desired conclusion follows immediately by axiom C1.

3. An Operational Semantics for Coroutines

To substantiate our claim that history variables are not necessary for verifying simple coroutines, we prove that the axiom system of Section 2 is sound and complete with respect to an operational semantics for coroutines. In this section we describe the syntax and semantics of a simple programming language for coroutines (LFC). In Sections 4 and 5 the soundness and completeness proofs will be given.

An LFC statement is either an assignment statement "x:=e", a conditional statement "b → A_1, A_2", a while statement "b*A", a compound statement "begin A_1; A_2;...A_n end", or a coroutine statement "coroutine Q_1, Q_2 end". Within a coroutine statement, two additional statement types are possible: the "exit" statement in Q_1 and the "resume" statement in Q_2. Since we are interested in the correctness of LFC programs, we must also specify the logical system in which the correctness assertions are expressed. In this paper the assertion language is a first order language with equality which we denote by AL. To simplify the semantics of LFC programs, we require that the boolean expressions of LFC conditionals be quantifier-free formulas of AL, and that the right hand sides of LFC assignment statements be terms in AL.

An interpretation I for AL consists of a set D (the domain of the interpretation), an assignment of functions on D to the function symbols of AL and an assignment of predicates on D to the predicate symbols of AL. Let ID be the set of identifiers (i.e. variables) of AL, and let I be an interpretation for AL with domain D. A program state is a mapping from ID to D giving the "value" associated with each identifier. The set of all program states will be denoted by S. If t is a term of AL with variables $x_1, x_2 \ldots x_n$ and s is a program state, then t(s) will denote

$$t\ \frac{s(x_1)\ldots s(x_n)}{x_1,\ldots,x_n}$$

i.e. the term obtained from t by simultaneously substituting $s(x_1),\ldots s(x_n)$ for $x_1,\ldots x_n$.

Similarly, we may define P(s) where P is a formula of AL.

It will also be convenient to identify a predicate P with the set $\{s \mid I[P(s)] = true\}$ of program states which make P true. False will correspond to the empty state set, true will correspond to the set S of all program states, and logical operations on predicates may be interpreted as set theoretic operations on subsets of S, i.e. "or" becomes "union", "and" becomes "intersection", "not" becomes "complement", and "implies" becomes "is a subset of". In general there will be many sets of states which are not expressible by formulas of the assertion language AL.

Meanings of LFC statements are specified by a state-transition function COMP(A,s) which associates with statement A and state s, a new state s'. Intuitively s' is the state resulting if A is executed with initial state s. The definition of COMP(A,s) is by cases on A:

(1) A is "x:=e" → s' where s'(y)=s(y)

if $y \neq x$ and $s'(x)=I[e(s)]$.

(2) A is "b → A_1, A_2" → $\begin{cases} COMP(A_1,s) & s\varepsilon b \\ COMP(A_2,s) & \text{otherwise} \end{cases}$

(3) A is "b*A_1" ----→ $\begin{cases} COMP("b*A_1",COMP(A_1,s)) & s\varepsilon b \\ s & \text{otherwise} \end{cases}$

(4) A is "begin A_1; A_2;...A_n end" ----→
COMP("begin A_2...A_n end",COMP(A_1,s))

(5) A is "begin end" ----→ s

(6) COMP("Coroutine Q_1, Q_2 end",s) is defined in terms of two mutually recursive procedures C1 and C2 as follows:
COMP("Coroutine Q_1, Q_2 end",s) = C1(Q_1, Q_2, s)
where C1(R_1, R_2, s) is defined by cases on R_1. (R represents the remainder of statement R_1).

(6a) R_1 = "x:=e; R" ----→ C1(R,R_2,s')
where s'(y)=s(y) if y≠x and s'(x)=I[e(s)].

(6b) R_1 = "b→A_1,A_2; R" → $\begin{cases} C1("A_1;R",R_2,s) & s\varepsilon b \\ C1("A_2;R",R_2,s) & \text{otherwise} \end{cases}$

(6c) R_1 = "b*A_1; R" ----→ $\begin{cases} C1("A_1;b*A_1;R",R_2,s) & s\varepsilon b \\ C1(R,R_2,s) & \text{otherwise} \end{cases}$

(6d) R_1 = "begin A_1;A_2...A_n end; R" ----→
C1("A_1; begin A_2...A_n end; R", R_2, s)

(6e) R_1 = "begin end; R" ----→ C1(R, R_2, s)

(6f) R_1 = "exit; R" ----→ C2(R, R_2, s)

(6g) R_1 = Λ (i.e. R_1 is the empty string) ----→ s.

The definition of C2(R_1, R_2, s) is the dual of the definition of C1 except that C2(R_1, Λ , s) = C1(R_1, Λ , s). Thus execution of the coroutine always terminates in Q_1. Note also that the definition of COMP does not allow for nested coroutines. Clause 6 could be modified to handle this case as well; however, nesting of coroutines is unnecessary to illustrate most of the difficulties involved in using the axioms of Section 2.

Partial correctness formulas will have the form {P} A {Q} where A is an LFC statement and P and Q are formulas of the assertion language AL.

3.1 Definition: {P} A {Q} is true with respect to interpretation I ($\models_I$ {P} A {Q}) iff ∀ s , s'∈S [s∈P ∧ COMP(A,s) = s' => s'∈Q].

In order to prove partial correctness formulas involving LFC statements, five additional axioms and rules of inference are needed:

(H1) assignment

$$\{Q_x^e\}\ x:=e\ \{Q\}$$

(H2) conditional

$$\frac{\{P\wedge b\}\ A_1\ \{Q\},\ \{P\wedge\neg b\}\ A_2\ \{Q\}}{\{P\}\ \ b \to A_1,\ A_2\ \{Q\}}$$

(H3) while

$$\frac{\{P\wedge b\}\ A\ \{P\},\ P\wedge\neg b \to Q}{\{P\}\ b*A\ \{Q\}}$$

(H4a) composition

$$\frac{\{P\}\ A\ \{Q\}}{\{P\}\ \text{begin } A \text{ end}\ \{Q\}}$$

(H4b) composition

$$\frac{\{P\}\ A_1\ \{R\},\ \{R\}\ \text{begin } A_2...A_n \text{ end}\ \{Q\}}{\{P\}\ \text{begin } A_1;\ A_2...A_n \text{ end}\ \{Q\}}$$

(H5) consequence

$$\frac{P \to R_1,\ \{R_1\}\ A\ \{R_2\},\ R_2 \to Q}{\{P\}\ A\ \{Q\}}$$

Proofs of partial correctness formulas are constructed from basic partial correctness axioms H1-H5, the coroutine axioms C1-C4, and a proof system T for the true formulas of the assertion language AL. Formally, a proof will consist of a sequence of partial correctness formulas {P} A {Q} and formulas of AL each of which is either an axiom or follows from previous formulas by a rule of inference. If {P} A {Q} occurs as a line in such a proof, then we write |- {P} A {Q}. In a similar manner we may define Π_1 |- Π_2 where Π_1 and Π_2 are sets of partial correctness formulas.

4. Soundness

A deduction system is sound iff every theorem is actually true. In order to prove the soundness of our deduction system for coroutines, we must show that each axiom is true and that if all of the hypothesis of a rule of inference are true, the conclusion will be true also. For all of the axioms and rules of inference except C1, soundness is either trivial or has been previously demonstrated ([CK77a], [CK77b], [HO74]). Thus, in this section we restrict our attention to the rule of inference C1 for coroutines. We assume that we are given two proofs of the form

$$\{P'\}\ \text{exit}\ \{R'\} \mid\!- \{P\wedge b\}\ Q_1\ \{R\} \qquad (4.1)$$

and

$$\{R'\}\ \text{resume}\ \{P'\} \mid\!- \{P'\wedge b\}\ Q_2\ \{R'\} \qquad (4.2)$$

Without loss of generality we may also assume that there are no redundant lines in the proofs of 4.1 and 4.2 since there is a simple algorithm for eliminating them. We must show that

$$\models \{P\wedge b\}\ \text{coroutine } Q_1, Q_2 \text{ end}\ \{R\}$$

Let L be the set of LFC statements occurring in the proofs of 4.1 and 4.2. In constructing L we distinquish between multiple occurrences of the same statement at different points in "coroutine Q_1, Q_2 end". Thus if Q_1 contains five different "exit" statements, L will contain five different exit statements. We also construct two functions pre and

post[1] which map the statements of L to assertions and satisfy the following conditions:

(1) $Q_1, Q_2 \in L$. $pre(Q_1)=P\wedge b$, $post(Q_1)=R$
$pre(Q_2)=P'\wedge b$, $post(Q_2)=R'$

(2) If A in L is "x:=e", then $pre(A)=post(A)\frac{e}{x}$.

(3) If A in L is "$b\rightarrow A_1, A_2$", then A_1 and A_2 are also in L and
$pre(A)\wedge b \rightarrow pre(A_1)$
$pre(A)\wedge\sim b \rightarrow pre(A_2)$
$post(A_1) \rightarrow post(A)$
$post(A_2) \rightarrow post(A)$.

(4) If A in L is "$b*A_1$", then $A_1 \in L$ and
$pre(A)\wedge b \rightarrow pre(A_1)$
$pre(A)\wedge\sim b \rightarrow post(A)$
$post(A_1) \rightarrow pre(A)$

(5) If A in L is "begin A_1 end", then $A_1 \in L$ and
$pre(A) \rightarrow pre(A_1)$
$post(A_1) \rightarrow post(A)$

(6) If A in L is "begin A_1; A_2;...; A_n end", then $A_1 \in L$, "begin A_2;...A_n end" $\in L$ and
$pre(A) \rightarrow pre(A_1)$
$post(A_1) \rightarrow pre(\text{begin } A_2;...A_n \text{ end})$
$post(\text{begin } A_2...A_n \text{ end}) \rightarrow post(A)$

(7) If A in L is "$exit_i$", then there is a predicate C_i which does not involve any free variables changed by Q_2 such that
$pre(exit_i) = P'\wedge C_i$
$post(exit_i) = R'\wedge C_i$

(8) If A in L is "$resume_i$", then there is a predicate D_i which does not involve any free variables which are changed by Q_1 such that
$pre(resume_i) = R'\wedge D_i$
$post(resume_i) = P'\wedge D_i$

Since the construction of the pre and post functions is relatively straightforward, we will not discuss the details of the construction any further in this paper. The next theorem is the main technical result of this section. From the theorem we are immediately able to deduce the soundness of rule C1.

4.3 Theorem: Let $s\in P\wedge b$. If $C1(A_1, A_2, s')$ ($C2(A_1, A_2, s')$) occurs as the i^{th} step in the computation COMP("coroutine Q_1, Q_2 end", s) then

(1) $A_1=A_1^1; A_2^1;...A_n^1$ where each $A_i^1\in L$

(2) $A_2=A_1^2; A_2^2;...A_m^2$ where each $A_i^2\in L$

(3) $s'\in pre(A_1^1)$ $\quad$ $(s'\in pre(A_1^2))$

(4) $post(A_i^1) \subseteq pre(A_{i+1}^1)$, $1\le i<n$

(5) $post(A_i^2) \subseteq pre(A_{i+1}^2)$, $1\le i<m$

(6) $post(A_n^1) \subseteq R$ $\quad$ $(post(A_m^2) \subseteq R')$

Proof: (By induction on the number of steps in the computation COMP("coroutine Q_1, Q_2 end", s)).

(Basis) The theorem is true initially since COMP("coroutine Q_1, Q_2 end",s)=$C1(Q_1, Q_2, s)$, $Q_1\in L$, $s\in P\wedge b \subseteq pre(Q_1)$, and $post(Q_1) \subseteq R$.

(Induction) We assume that the theorem is true at step i and show that it is also true at step i+1. Assume that step i is $C1(A_1, A_2, s')$. By induction

(1) $A_1=A_1^1; A_2^1;...A_n^1$ where $A_i^1\in L$

(2) $s'\in pre(A_1^1)$, $post(A_n^1) \subseteq P$

(3) $post(A_i^1) \subseteq pre(A_{i+1}^1)$, $1\le i<n$

The $i+1^{th}$ step in the computation will be determined by A_1^1. We will consider the cases in which A_1^1 is an assignment statement, a while statement, and an exit statement. The remaining statements are similar and will be left to the reader.

(a) A_1^1 is "x:=e". In this case the next computation step will be $C1(A_2^1;...A_n^1, A_2, s^*)$ where $s^*(y)=s'(y)$ if $y\neq x$ and $s^*(x)=I[e(s')]$. Since $s'\in pre(A_1^1)$ and $pre(A_1^1) \subseteq post(A_1^1)\frac{e}{x}$, we see that $s'\in post(A_1^1)\frac{e}{x}$ or that $s^*\in post(A_1^1)$. Since $post(A_1^1) \subseteq pre(A_2^1)$, it follows that $s^*\in pre(A_2^1)$. Clearly the other conditions of the theorem are satisfied.

(b) A_1^1 is "b*E". If $s'\in b$ then the next computation step will be $C1("E; b*E: A_2^1;...A_n^1", A_2, s')$. Since $pre(A_1^1)\wedge b \rightarrow pre(E)$ and $s' \in pre(A_1^1)$, it follows that $s' \in pre(E)$. Since $post(E) \rightarrow pre(A_1^1)$, we see that the theorem will also hold for the $i+1^{th}$ computation step. The case in which $s'\notin b$ is similar and will be left to the reader.

A_1^1 is "$exit_i$". In this case the next computation step is $C2("A_2^1;...A_n^1", A_2, s')$. By construction of the pre function we have $s'\in pre(exit) \subseteq P'\wedge C_i$. There are two subcases depending on whether the

[1] Pre and post functions were first used in soundness proofs by S. Owicki [OW76].

second routine (Q_2) of the coroutine has been previously executed.

Case i: Suppose $A_2=Q_2$ and that Q_2 has not been previously executed. In this case $P'\wedge b \subseteq pre(Q_2) = pre(A_2)$. Thus $s'\varepsilon P'$ and $s'\varepsilon b$. It follows that $s'\varepsilon P'\wedge b \subseteq pre(A_2)$.

Case ii: Suppose that "resume$_i$" was the last statement executed when control was previously in Q_2. Assume also that $pre(resume_i)=R'\wedge D_i$ and $post(resume_1)=P'\wedge D_i$. Since D_i does not contain any free variables changed by Q_1, $s'\varepsilon D_i$. Since $s'\varepsilon P'$, we have $s'\varepsilon P'\wedge D_i \subseteq post(resume_i) \subseteq pre(A_1^2)$.

This completes the induction step in the proof of Theorem 4.3. The reader will observe that the omitted cases in the proof including the "resume" statement for C2 are analoguous to the cases considered. Note also that if $s\varepsilon P\wedge b$ and COMP("coroutine Q_1, Q_2 end", s) = s', then by Theorem 4.3 $s'\varepsilon post(A_n^1) \subseteq R$. Thus $\models \{P\wedge b\}$ coroutine Q_1, Q_2 end $\{R\}$. This completes the proof of soundness for the rule of inference C1 for coroutines.

5. Completeness

A deduction system is complete iff every true formula is provable. Unfortunately, if the proof system T for the assertion language is axiomatizable and if a sufficiently rich interpretation (such as number theory) is used for the assertion language, then it is impossible to specify a proof system for partial correctness of LFC programs which is both sound and complete. This follows from the fact that the divergence problem for turing machines can be expressed as a partial correctness problem for LFC programs [CO75]. We can, however, prove a relative completeness theorem similar to the one proposed by Cook for simple Algol programs [CO75]. If the proof system T for the assertion language is complete and if the assertion language satisfies a natural expressibility condition, then every true LFC partial correctness formula will be provable using the axioms and rules of inference described in Sections 2 and 4. Furthermore, these proofs of partial correctness do not involve the use of history variables.

Before describing the notion of expressibility used in the relative completeness theorem, it is necessary to introduce some additional notation.

Since our primary interest is the coroutine statement, we will restrict our attention to LFC programs of the form "coroutine Q_1, Q_2 end". We will represent the computation of such a program with initial state s_0 by

$$<Q_1, Q_2, s_0> <Q_1^1, Q_2^1, s_1>\ldots<Q_1^i, Q_2^i, s_i>\ldots$$

where detailed rules for deriving $<Q_1^{i+1}, Q_2^{i+1}, s_{i+1}>$ from $<Q_1^i, Q_2^i, s_i>$ may be obtained from the semantics for coroutines given in Section 3.

If A is an LFC statement, then SUB(A) is the set of substatements of A. Any statement is a substatement of itself; for composite statements A such as "$b\rightarrow A_1, A_2$" any substatement of A_1 or A_2 is also a substatement of A. Note that different occurrences of the same statement in A are distinguished in SUB(A).

Given a coroutine statement A of the form "coroutine Q_1, Q_2 end" and a predicate P, we define functions PRE and POST which associate sets of states with the statements in SUB(A). These functions are the duals of the pre and post functions used in the soundness proof of Section 4. Intuitively, $PRE(A_1)$ ($POST(A_1)$) is the set of program states in which A can be immediately before (after) the execution of substatement A_1, if the initial state of A satisfies the predicate P. When A_1 is a substatement of Q_1, we may formally define $PRE(A_1)$ and $POST(A_1)$ by

$PRE(A_1)=\{s^* |$ there is a computation of A of the form $<Q_1, Q_2, s> <Q_1^1, Q_2^1, s^1>\ldots<A_1;Q_1^*, Q_2^*, s^*>$ and $s\varepsilon P\}$

$POST(A_1)=\{s^* |$ there is a computation of A of the form $<Q_1, Q_2, s> <Q_1^1, Q_2^1, s^1>\ldots<A_1;Q_1^*, Q_2^*, s''>$ $\ldots<Q_1^*, Q_2^*, s^*>$ and $s\varepsilon P\}$

Analogous definitions may also be given when A_1 is a substatement of Q_2.

5.1 Definition: The assertion language AL is expressive with respect to interpretation I iff for all programs A of the form "coroutine Q_1, Q_2 end" and all predicates P in AL, $PRE(A_1)$ and $POST(A_1)$ are expressible by formulas of AL whenever $A_1\varepsilon SUB(A)$.

There are examples of assertion languages and interpretations which fail to be expressive; however, realistic choices for AL and I do give expressibility. If for example AL is the language of arithmetic and I is an interpretation for AL in which the symbols of number theory receive their usual interpretations, then AL is expressive with respect to I [CO75]. Also if I is a finite interpretation, then the assertion language will be expressive [CK76a]. In the remainder of this paper we will always assume that the expressibility condition is satisfied by the assertion language and interpretation that we are using.

Additional important properties of the PRE and POST functions are listed below; proofs of these properties may be obtained directly from the definitions of the PRE and POST functions and will not be given in this paper.

If $\models\{P\}$ coroutine Q_1, Q_2 end $\{R\}$, then

(1) $P = PRE(Q_1)$, $POST(Q_1) \subseteq R$

(2) $PRE(x:=e) = POST(x:=e)\frac{e}{x}$

(3) $PRE(b\rightarrow A_1,A_2)\wedge b = PRE(A_1)$
$PRE(b\rightarrow A_1,A_2)\wedge\neg b = PRE(A_2)$
$POST(A_1) \subseteq POST(b\rightarrow A_1,A_2)$

$POST(A_2) \subseteq POST(b\rightarrow A_1, A_2)$

(4) $PRE(b*A) \wedge b = PRE(A)$

$PRE(b*A) \wedge \neg b = POST(b*A)$

$POST(A) = PRE(b*A)$

(5) $PRE(\text{begin } A \text{ end}) = PRE(A)$

$POST(\text{begin } A \text{ end}) = POST(A)$

(6) $PRE(\text{begin } A_1; A_2 \ldots A_n \text{ end}) = PRE(A_1)$

$POST(A_1) = PRE(\text{begin } A_2 \ldots A_n \text{ end})$

$POST(\text{begin } A_2 \ldots A_n \text{ end}) = POST(\text{begin } A_1; A_2 \ldots A_n \text{ end})$

(7) $\bigvee_i PRE(exit_i) = PRE(Q_2) \vee \bigvee_j POST(resume_j)$

$\bigvee_i POST(exit_i) = \bigvee_j PRE(resume_j) \vee POST(Q_2)$

The index i in (7) ranges over all distinct "exit" statements in $SUB(Q_1)$. The index j ranges over all distinct "resume" statements in $SUB(Q_2)$.

We are now ready to begin the proof of relative completeness. Assume that {P} coroutine Q_1, Q_2 end {R} is true; we must show that it is provable using the axioms and rules of inference in Sections 2 and 3 and the complete proof system T for the true formulas of the assertion language. Without loss of generality we may assume that auxiliary variables i, j have been added to the coroutine program so that it has the form:

```
i:=0; j:=0;
coroutine
        begin
          .
          .
          i:=i0;
Q1        exit_i0 ;
          .
          .
        end,
        begin
          .
          .
          j:=j0;
Q2        resume_j0 ;
          .
          .
        end
end
```

Let $P' = \bigvee_i PRE(exit_i) = PRE(Q_2) \vee \bigvee_j PRE(resume_j)$

$R' = \bigvee_i POST(exit_i) = \bigvee_j PRE(resume_j) \vee POST(Q_2)$

$b = \{i=0 \wedge j=0\}$

By the expressibility condition P', R', and b are representable by formulas of AL. Note also that all of the following conditions are satisfied:

$P' \wedge (i=i_0) \equiv PRE(exit_{i_0})$

$R' \wedge (i=i_0) \equiv POST(exit_{i_0})$

$P' \wedge b \equiv PRE(Q_2)$

$P \wedge b \equiv PRE(Q_1)$

$POST(Q_1) \rightarrow R$

$POST(Q_2) \rightarrow R'$

Proofs of these formulas may be obtained using the complete proof system T for AL. We need only establish that

$\{P'\}$ exit $\{R'\} \vdash \{PRE(Q_1)\}\ Q_1\ \{POST(Q_1)\}$ (5.2)

and

$\{R'\}$ resume $\{P'\} \vdash \{PRE(Q_2)\}\ Q_2\ \{POST(Q_2)\}$ (5.3)

We will outline a proof that (5.2) holds; (5.3) is similar and will be left to the reader. The proof of 5.2 uses induction on the structure of Q_1. Let A be a substatement of Q_1; we will show that $\{P'\}$ exit $\{R'\} \vdash \{PRE(A)\}\ A\ \{POST(A)\}$. If A is any statement but an "exit" or "resume" statement, this is trivial. For example, suppose that A is "$b\rightarrow A_1, A_2$", then

$\{PRE(A_1)\}\ A_1\ \{POST(A_1)\}$

and

$\{PRE(A_2)\}\ A_2\ \{POST(A_2)\}$

are provable by the induction hypothesis. Thus,

$\{PRE(b\rightarrow A_1, A_2) \wedge b\}\ A_1\ \{POST(b\rightarrow A_1, A_2)\}$

and

$\{PRE(b\rightarrow A_1, A_2) \wedge \neg b\}\ A_2\ \{POST(b\rightarrow A_1, A_2)\}$

may be proved using the rule of consequence. From the rule of inference for the conditional, we conclude that $\{PRE(b\rightarrow A_1, A_2)\}\ b\rightarrow A_1, A_2\ \{POST(b\rightarrow A_1, A_2)\}$ is provable as required.

If A is the statement "$exit_{i_0}$", then we may use the coroutine axiom C2 and the hypothesis $\{P'\}\ exit_{i_0}\ \{R'\}$ to deduce $\{P' \wedge i=i_0\}\ exit_{i_0}\ \{R' \wedge i=i_0\}$. Since $P' \wedge (i=i_0) \equiv PRE(exit_{i_0})$ and $R' \wedge (i=i_0) \equiv POST(exit_{i_0})$, we conclude that $\{PRE(exit_{i_0})\}\ exit_{i_0}\ \{POST(exit_{i_0})\}$ is provable also.

This concludes the outline of the relative completeness proof. Note that history variables are not needed in the construction; in fact, since the variables i and j used in the proof have bounded magnitudes, the entire construction can be carried out with only 0,1-valued auxiliary variable.

6. Open Problems

We have argued that history variables are not needed in proofs of correctness for simple coroutines. One might wonder if history variables are ever needed in correctness proofs. In an earlier paper [CK77a] we proved that it is impossible to obtain a sound and relatively complete Hoare axiom system for a programming language with coroutines, if the coroutines are allowed to contain local recursive procedures. The notion of expressibility used in this incompleteness proof did not allow the use of history variables. If history variables were permitted, would the completeness theorem of Section 5 extend to handle local recursive

procedures also? We conjecture that the answer to this question is YES; if so, an alternative interpretation of the results in [CK76a] would be that history variables are necessary in correctness proofs of the extended coroutine language. We suspect, however, that the difficulty of using history variables in actual correctness proofs, would limit the use of this technique to very simple programs.

History variables have also been used in correctness proofs for concurrent programs. Owicki, for example, describes a proof system for a concurrent programming language in which synchromization is handled by conditional critical regions [OW76]. She also shows that her proof system is sound and relatively complete with respect to an operational semantics for her language. The proof of completeness requires the use of history variables to record the order in which critical regions are entered. Other researchers, including Howard [HW76], have also used history variables in correctness proofs for concurrent programs.

Are history variables necessary for formal verification of concurrent programs? In the case of Owicki's language, any concurrent program can be transformed into an equivalent nondeterministic Algol program in which the nondeterminism is used to simulate the possible interleavings of statements. Since deBakker and Meertens [DE73] have shown that a sound and relatively complete proof system may be given for nondeterministic Algol which does not require the use of history variables, it follows that history variables are not needed in proofs of partial correctness for Owicki's language. This solution is not completely satisfactory, however, since it is not clear that the transformation into nondeterministic Algol preserves such important properties of concurrent programs as absence of starvation. A more interesting open question is whether there is a proof system similar to the one originally described by Owicki which does not require the use of history variables.

In view of these remaining open problems, we believe that the question of whether history variables are really necessary and whether their use significantly complicates correctness proofs is far from settled and deserves additional research.

References

[APT77] Apt, K.R., Bergstra, J.A., and Meertens, L.G.L.T. Recursive assertions are not enough--or are they? Mathematical Centre Report IW 92/77.

[CK77a] Clarke, E.M. Programming language constructs for which it is impossible to obtain good Hoare-like axiom systems. Proceedings of the 4th POPL, 1977.

[CK77b] Clarke, E.M. Program invariants as fixed points. Proceedings of the 18th FOCS, 1977.

[CL73] Clint, M. Program proving: Coroutines. Acta Informatica, 2:50-63, 1973.

[C)75] Cook, S.A. Axiomatic and interpretative semantics for an Algol fragment. Technical Report 79, Department of Computer Science, University of Toronto, 1975 (to be published in SCICOMP).

[DE73] deBakker, J.W. and Meertens, L.G.L.T. On the completeness of the inductive assertion method. Mathematical Centre, December 1973.

[FL67] Floyd, R.W. Assigning meaning to programs. In: Mathematical Aspects of Computer Science Proc. Symposia in Applied Mathematics, J.T. Schwartz, Ed., 19:19-32, Amer. Math. Soc., 1967.

[GO75] Gorelick, G. A complete axiomatic system for proving assertions about recursive and non-recursive programs. Technical Report No. 75, Department of Computer Science, University of Toronto, January 1975.

[HO69] Hoare, C.A.R. An axiomatic approach to computer programming. CACM, 12:322-329, October 1969.

[HO74] Hoare, C.A.R. and Lauer, P.E. Consistent and complementary formal theories of the semantics of programming languages. Acta Informatica, 3:135-154, 1974.

[HW76] Howard, J.H. Proving monitors. COMM ACM, 19(5):273-279, May 1976.

[MA70] Manna, Z. and Pnuefi, A. Formalization of properties of functional programs. JACM, 17(3):555-569, 1970.

[OW76] Owicki, S. A consistent and complete deductive system for the verification of parallel programs. 8th Annual Symposium on Theory of Computing, 1976.

[WA76] Wand, M. A new incompleteness result for Hoare's system. 8th Annual Symposium on Theory of Computing, 1976.