Proving Correctness of Coroutines
Without History Variables

Edmund Melson Clarke*
Harvard University, Center for Research in Computing Technology.
Aiken Computation Laboratory, Cambridge, MA 02138, USA

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 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 [OW76a] and Howard [HO75] in verifying concurrent programs and by Apt [APT77] in verifying sequential programs. We argue that recording the entire history of a computation in a single set of variables can actually complicate a correctness proof and should be avoided if possible. We propose a modification of Clint’s axiom system and a strategy for constructing proofs that eliminates the need for history variables in verifying simple coroutines. Examples (including Clint’s program “Histo”) 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 such variables are necessary in verifying concurrent programs.

1. Introduction

The obvious power of history variables in program proofs stems from the large amount of information about a program’s behavior that 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

* The preparation of this paper was supported by NSF Grant MCS-75-08146. Author’s present address: Center for Research in Computing Technology, Harvard University, Cambridge, MA 02138
correctness assertions than simple program identifiers. For each type $T$ in the original program it is necessary to introduce a new type sequence of $T$ together with operations for concatenating and indexing such sequences. Another, less obvious, disadvantage is that it is no longer possible to construct program proofs in a top-down manner in which programs are regarded as static objects and only the input-output behavior of a statement is used to relate the statement to the remainder of the program. Instead the input-output assertions associated with a given statement in the program may involve variables that reflect the execution history of the entire program. Because of these disadvantages we believe that the use of history variables should be avoided whenever possible. Since history variables seem to be essential for correctness proofs of certain language constructs, it becomes important to identify the constructs where such variables can be avoided.

In this paper we show that history variables are not needed in proving the correctness of simple coroutines. We first give an informal argument by showing how coroutine programs can be transformed to programs using procedure parameters for which history variables are not needed. We next 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 (including Clint’s “Histo” example) and give a proof of soundness and relative completeness for our axiom system. Finally, we discuss some extensions of the coroutine concept that 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.

2. Coroutines

A coroutine will have the form:

```
coroutine $R_1$, $R_2$ end
```

$R_1$ is the main routine; execution begins and terminates in $R_1$ (the requirement that execution terminate in $R_1$ is not absolutely necessary but simplifies the axiom for coroutines). Otherwise $R_1$ and $R_2$ behave in identical manners. If an exit statement is encountered in $R_1$, the next statement to be executed will be the statement following the last resume statement in $R_1$. Similarly, execution of resume statement in $R_2$ causes execution to be restarted following the last exit statement executed in $R_1$. A simple example of a coroutine is:

```
Coroutine

while $y \neq z$ do begin
  $y := y + 1$;  $x := x + y$;  exit
end,
while $y$ do begin
  $y := y - 2$;  resume;
  $y := y + 1$;  resume
end
end
```
Note that if \( x \) and \( y \) are 1 initially and \( z \geq 1 \), then the coroutine will terminate with \( y = z^2 \).

2.1. An Informal Argument that History Variables Are Not Needed

Before discussing our modification of Clint's axiom system we give an informal argument that history variables are not needed for proving partial correctness of simple coroutines. The programs that we consider are statements in the programming language LFC (language for coroutines) which we introduce for expository purposes in this paper. An LFC statement is either an assignment statement \( x := e \), a conditional statement \( \text{if } b \text{ then } A_1 \text{ else } A_2 \), a while statement \( \text{while } b \text{ do } A \), a compound statement \( \text{begin } A_1; A_2; \ldots; A_n \text{ end} \), or a coroutine statement \( \text{coroutine } R_1, R_2 \text{ end} \). Within the coroutine statement two additional statement types are possible: the \text{exit} statement in \( R_1 \) and the \text{resume} statement in \( R_2 \). In Sect. 3 a formal semantics will be given for the language LFC; in this section we will assume that the coroutine statement has the intuitive meaning described in 2.1 and that the other statements of the language have their standard meanings.

We will show that any statement \( H \) in LFC can be transformed into an equivalent statement \( H^* \) in a programming language PL for which there exists a complete proof system that does not require the use of history variables. The transformation of \( H \) into \( H^* \) must have the following properties:

1. It must not introduce history variables into program \( H^* \).

2. It must preserve partial correctness, i.e., for all predicates \( P \) and \( Q \) in the underlying assertion language, the partial correctness formula \( \{ P \} H \{ Q \} \) is true if and only if the formula \( \{ P \} H^* \{ Q \} \) is also true.

Our transformation is based on Wijngaarden's device [W176] and uses procedure calls to simulate the transfer of control that occurs in while loops, conditionals, and \text{exit} (\text{resume}) statements. Without loss of generality we will assume that each statement \( A \) in program \( H \) has a unique label \( L \) and that \( L \) denotes that label of the statement following \( A \) in program \( H \) (if \( A \) has no successor in \( H \), then \( L \) is the label of the null statement). The transformation from \( H \) to \( H^* \) will replace each statement \( L; A \) in \( H \) by a procedure declaration \( \text{proc } L(F); \ldots; \text{end } L \). The name of the procedure will be obtained from the label of the statement it replaces. Each procedure will have a procedure parameter, which is used to determine the next statement to be executed when an exit or resume statement is encountered.

If \( A \) is \text{coroutine } R_1, R_2 \text{ end} \), then replace \( A \) by

\[
\begin{align*}
\text{proc } &L_1^1(F); \ldots; \text{end } L_1^1; \\
\text{proc } &L_1^m(F); \ldots; \text{end } L_1^m; \\
\text{proc } &L_2^1(F); \ldots; \text{end } L_2^1; \\
\text{proc } &L_n^1(F); \ldots; \text{end } L_n^1; \\
&L_1^1 \ldots L_n^m
\end{align*}
\]

where \( L_1^1 \ldots L_n^m \) are the procedure declarations associated with \( R_1 \) and \( L_1^1 \ldots L_n^m \) are the procedure declarations associated with \( R_2 \).
If $A$ is an exit statement or a resume statement, then replace $L:A$ by

\[ \text{proc } L(F); F(L) \text{ end } L; \]

If $A$ is the assignment statement $x := e$; then replace $L:x := e$ by

\[ \text{proc } L(F); \ x := e; \ L(F) \text{ end } L; \]

If $A$ is a while statement, then replace $L: \text{while } b \text{ do } L_1\text{; } A_1$ by

\[ \text{proc } L(F); \ \text{if } b \text{ then } L_1(F) \text{ else } L(F) \text{ end } \]

The cases in which $A$ is a conditional or block statement are similar and will be left to the reader. Thus, the example of Sect. 2 will be transformed to

\[ \text{proc } L_1(F); \ \text{if } y = z \text{ then } L_2(F) \text{ else skip } \text{ end } L_1; \]
\[ \text{proc } L_2(F); \ y := y + 1; \ L_3(F) \text{ end } L_2; \]
\[ \text{proc } L_3(F); \ x := x + y; \ L_4(F) \text{ end } L_3; \]
\[ \text{proc } L_4(F); \ \text{call } F(L_1) \text{ end } L_4; \]
\[ \text{proc } L_1(F); \ \text{if } \text{true} \text{ then } L_5(F) \text{ else skip } \text{ end } L_1; \]
\[ \text{proc } L_5(F); \ y := y - 2; \ L_6(F) \text{ end } L_5; \]
\[ \text{proc } L_6(F); \ F(L_4) \text{ end } L_6; \]
\[ \text{proc } L_4(F); \ y := y + 1; \ L_7 \text{ end } L_4; \]
\[ \text{proc } L_7(F); \ F(L_1) \text{ end } L_7; \]
\[ L_1(L_1) \]

Note that the above construction never results in internal procedure declarations (one procedure declaration nested within another procedure declaration). Clarke [CK77a] has shown that there is a complete proof system for procedures with procedure parameters if internal procedures are not allowed. Clearly, the transformation form $H$ to $H^*$ does not introduce history variables. A formal proof that the transformation preserves partial correctness may be obtained from the semantics of coroutines given in Sect. 3 and the semantics for procedures with procedure parameters given in [CK77].

Although the transformation of simple coroutines into procedures with procedure parameters is straightforward, we do not seriously recommend this technique for proving correctness of coroutines. In constructing proofs involving a programming language construct $C_1$ that can be defined in terms of a more general construct $C_2$, it is usually best to work directly with axioms for the construct $C_1$; axioms for the more general construct $C_2$ must be more complicated simply to handle the additional cases in which $C_2$ may be used. For example, the while statement may be defined in terms of the goto statement; however it is usually easier to prove correctness of while programs directly rather than to translate them into goto programs and prove the resulting goto program correct.

2.2. Axioms for Coroutines

In this section we give a set of axioms for coroutines and describe a technique for proving correctness of coroutines that 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 histories) rather than program histories.

C1. (Coroutine)

\[
\begin{align*}
\{P\} & \text{exit} \{Q\} \vdash \{P \land b\} R_1 \{Q\} \\
\{Q\} & \text{resume} \{P\} \vdash \{P' \land b\} R_2 \{Q'\} \\
\{P \land b\} & \text{coroutine} R_1, R_2 \text{end} \{Q\}
\end{align*}
\]

provided that no variable free in \( b \) is global to \( R_1 \). (This axiom is a modification of the one in [CL73].)

C2. (Exit)

\[
\begin{align*}
\{P\} & \text{exit} \{Q\} \\
\{P' \land C\} & \text{exit} \{Q' \land C\}
\end{align*}
\]

provided that \( C \) does not contain any free variables that are changed by \( R_2 \). (Here we assume that \text{exit} occurs only in statement \( R_1 \) of \text{coroutine} \( R_1, R_2 \text{end} \).

C3. (Resume)

\[
\begin{align*}
\{Q\} & \text{resume} \{P'\} \\
\{Q' \land C\} & \text{resume} \{P' \land C\}
\end{align*}
\]

provided that \( C \) does not contain any free variables that are changed in \( R_1 \). (Here we assume that \text{resume} occurs only in statement \( R_2 \) of \text{coroutine} \( R_1, R_2 \text{end} \).

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 \( y \in AV \). If assertions \( P \) and \( Q \) do not contain any free variables from \( AV \) and if \( S \) is obtained from \( S' \) by deleting all assignments to variables in \( AV \), then

\[
\begin{align*}
\{P\} & S'(Q) \\
\{P\} & S(Q)
\end{align*}
\]

(This axiom is the same as that in [OW76].)

We illustrate the axioms with an example. We show that \( \{x = 1 \land y = 1 \land z \geq 1\} A\{x = z^2\} \) where \( A \equiv \text{coroutine} R_1, R_2 \text{end} \) is the coroutine given in Sect. 2.1. Our strategy in carrying out the proof will be to introduce auxiliary variables to distinguish the various \text{exit} and \text{resume} statements from each other and then to 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 \( \{P'\} \text{exit} \{Q'\} \) to a specific occurrence of an \text{exit} statement in \( R_1 \). A similar comment applies to axiom C3 for the \text{resume} statement. We prove:
\{x = 1 \land y = 1 \land z \geq 1\}
\ i = 0; \ j = 0;

**coroutine**
\[
\text{while } y \neq z \text{ do begin}
\quad y := y + 1; \ x := x + y;
\quad i := 1; \text{ exit}
\endend.
\]
\[
\text{while true do begin}
\quad y := y - 2; \ j := 1; \text{ resume;}
\quad y := y + 2; \ j := 0; \text{ resume}
\endend
\]
\{x = z^2\}

Note that two auxiliary variables are needed (one for each routine of the coroutine). 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 \(j\) can be used in assertions to distinguish which of the `resume` statements has been most recently executed. Auxiliary variable \(i\) of the first routine has a dual function. This technique of adding auxiliary variables will be formally described in Sect. 5; however, the general pattern should be clear from the above example.

To complete the proof we choose:
\[
\begin{align*}
P &= \{x = 1 \land y = 1 \land z \geq 1 \land i = 0 \land j = 0\} \\
b &= \{j = 0\} \\
Q &= \{x = z^2\} \\
P' &= \{(x = y^2 - y + 1 \land j = 0 \land y \leq z) \lor (x = y^2 + 2y + 1 \land j = 1 \land y = y \leq z - 1)\} \\
Q' &= \{(x = y^2 + 3y + 3 \land j = 1 \land y \leq z - 2) \lor (x = y^2 \land j = 0 \land y = z)\}.
\end{align*}
\]

The invariant for the **while** loop of the first routine is:
\[
\text{INV}_1 = \{(x = y^2 + 3y + 3 \land j = 1 \land y \leq z - 2) \lor (x = y^2 \land j = 0 \land y = z)\}
\]

The invariant for the **while** loop of the second routine is:
\[
\text{INV}_2 = \{x = y^2 - y + 1 \land j = 0 \land y \leq z\}.
\]

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

(a) \(\{P\} \text{ exit } \{Q'\} \vdash \{P \land b\} R_1 \{Q\}\)

and

(b) \(\{Q'\} \text{ resume } \{P'\} \vdash \{P' \land b\} R_2 \{Q'\}\)

both hold. For example, to prove (b) we assume \(\{Q'\} \text{ resume } \{P'\}\) and prove \(\{P' \land b\} R_2 \{Q'\}\). In order to prove \(\{P' \land b\} R_2 \{Q'\}\) we show that

(c) \(P' \land b \rightarrow \text{INV}_2\)
(d) \{\text{INV}_2\}

\textbf{while} true \textbf{do}\begin{align*}
y & := y - 2; \quad j := 1; \quad \text{resume:} \\
y & := y + 1; \quad j := 0; \quad \text{resume:}
\end{align*}
\textbf{end}
\{\text{INV}_2 \land \sim \text{true}\}

(e) \text{INV}_2 \land \sim \text{true} \rightarrow Q' \text{ are true.}

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

(d1) \text{assignment}
\{\text{INV}_2 \land \text{true}\} \quad y := y - 2; \quad j := 1 \{Q' \land j = 1\}

(d2) \text{resume}
\{Q' \land j = 1\} \text{ resume } \{P' \land j = 1\}

(d3) \text{assignment}
\{P' \land j = 1\} \quad y := y + 1; \quad j := 0 \{Q' \land j = 0\}

(d4) \text{resume}
\{Q' \land j = 0\} \text{ resume } \{P' \land j = 0\}

(d5) \text{arithmetic}
\quad P' \land j = 0 \rightarrow \text{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 (without using history variables) that the axiom system of Sect. 2 is sound and complete with respect to an operational semantics for coroutines. In this section we give a formal semantics for the language LFC introduced in Sect. 2.2. In Sects. 4 and 5 the soundness and completeness proofs will be given.

Since we are interested in the correctness of LFC programs, we must specify the logical system in which the correctness assertions are expressed. In this paper the \textit{assertion language} is a first order language with equality, which we denote \textit{by} AL. To simplify the semantics of LFC programs, we require that the Boolean expressions of LFC conditionals be \textit{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

$$I[s(x_1)/x_1, \ldots, s(x_n)/x_n]$$

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 $[s][P(s)] = true$ of program states that 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 $\text{COMP}(A, s)$, 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 $\text{COMP}(A, s)$ is by cases on $A$. We abbreviate if $b$ then $A_1$ else $A_2$ by “$b \rightarrow A_1, A_2$” and while $b$ do $A_1$ by “$b * A_1$”.

1. $A$ is “$x := e$” $\rightarrow s'$ where $s'(y) = s(y)$ if $y \neq x$ and $s'(x) = I[e(s)]$.

2. $A$ is “$b \rightarrow A_1, A_2$” $\rightarrow$$\begin{cases} \text{COMP}(A_1, s) & s \in b \\ \text{COMP}(A_2, s) & \text{otherwise} \end{cases}$

3. $A$ is “$b * A_1$” $\rightarrow$$\begin{cases} \text{COMP}("b * A_1", \text{COMP}(A_1, s)) & s \in b \\ s & \text{otherwise} \end{cases}$

4. $A$ is “begin $A_1; A_2; \ldots A_n$ end” $\rightarrow \text{COMP("begin $A_2; \ldots A_n$ end"', \text{COMP}(A_1, s))}$

5. $A$ is “begin end” $\rightarrow s$

6. $\text{COMP("Coroutine $R_1, R_2$ end", s)}$ is defined in terms of two mutually recursive procedures C1 and C2 as follows:

   $\text{COMP("Coroutine $R_1, R_2$ end", s)} = \text{C1}(R_1, R_2, s)$ where $\text{C1}(R_1, R_2, s)$ is defined by cases on $R_1$, ($R$ represents the remainder of statement $R_1$)

   (6a) $R_1 = "x := e; R" \rightarrow \text{C1}(R, R_2, s')$ where $s'(y) = s(y)$ if $y \neq x$ and $s'(x) = I[e(s)]$.

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

   (6c) $R_1 = "b * A_1; R" \rightarrow$$\begin{cases} \text{C1("A_1; b * A_1; R", R_2, s)} & s \in b \\ \text{C1}(R, R_2, s) & \text{otherwise} \end{cases}$
(6d) \( R_1 = \textbf{begin} A_1 ; A_2 \ldots A_n \textbf{end}; R \rightarrow \text{CI}(\sim A_1; \textbf{begin} A_2 \ldots A_n \textbf{end}; R, R_2, s) \)

(6e) \( R_1 = \textbf{begin} \textbf{end}; R \rightarrow \text{CI}(R, R_2, s) \)

(6f) \( R_1 = \textbf{exit}; R \rightarrow \text{C2}(R, R_2, s) \)

(6g) \( R_1 = \varepsilon \) (i.e., \( R_1 \) is the empty string) \( \rightarrow \varepsilon \).

The definition of \( \text{C2}(R_1, R_2, s) \) is the dual of the definition of \( \text{CI} \) except that \( \text{C2}(R_1, \varepsilon, s) = \text{C1}(R_1, \varepsilon, s) \). Thus execution of the coroutine always terminates in \( R_1 \). Note also that the definition of \( \text{COMP} \) 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 Sect. 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 \( I(\Rightarrow \{P\} A\{Q\}) \) iff \( \forall s, s' \in S, s \in P \land \text{COMP}(A, s) = s' \Rightarrow s' \in Q \).

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

(H1) assignment

\[
\left\{\begin{array}{c}
Q \leftarrow e
\end{array}\right\} x := e\{Q\}
\]

(H2) conditional

\[
\frac{\{P \land b\} A_1\{Q\}, \{P \land \sim b\} A_2\{Q\}}{\{P\} b \rightarrow A_1, A_2\{Q\}}
\]

(H3) while

\[
\frac{\{P \land b\} A\{P\}, P \land \sim b \rightarrow Q}{\{P\} b^* A\{Q\}}
\]

(H4a) composition

\[
\frac{\{P\} A\{Q\}}{\{P\} \textbf{begin} A \textbf{end}\{Q\}}
\]

(H4b) composition

\[
\frac{\{P\} A_1\{R\}, \{R\} \textbf{begin} A_2 \ldots A_n \textbf{end}\{Q\}}{\{P\} \textbf{begin} A_1; A_2 \ldots A_n \textbf{end}\{Q\}}
\]

(H5) consequence

\[
\frac{P \rightarrow R_1, \{R_1\} A\{R_2\}, R_2 \rightarrow Q}{\{P\} A\{Q\}}
\]
Proofs of partial correctness formulas are constructed from basic partial correctness axioms H1–H15, 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 \( \vdash \{P\} A\{Q\} \). In a similar manner we may define \( \Pi_1 \vdash \Pi_2 \) where \( \Pi_1 \) and \( \Pi_2 \) 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 the axioms and rules of inference except C1, soundness is either trivial or has been previously demonstrated ([CK77a], [CK77b], [HO74b]). 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

\[
\{P\} \text{ exit } \{Q\} \vdash \{P \land b\} R_1\{Q\} \tag{4.1}
\]

and

\[
\{Q\} \text{ resume } \{P\} \vdash \{P \land b\} R_2\{Q\}. \tag{4.2}
\]

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

\[\vdash \{P \land b\} \text{ coroutine } R_1, R_2 \text{ end } \{Q\}.\]

Let \( L \) be the set of LFC statements occurring in the proofs of (4.1) and (4.2). In constructing \( L \) we distinguish between multiple occurrences of the same statement at different points in \textit{coroutine } \( R_1, R_2 \text{ end}. \) Thus if \( R_1 \) contains five different \textit{exit} statements, \( L \) will contain five different \textit{exit} statements. We also construct two functions \textit{pre} and \textit{post} \(^1\) which map the statements of \( L \) to assertions and satisfy the following conditions:

1. \( R_1, R_2 \in L, \) \( \text{pre}(R_1) = P \land b, \) \( \text{post}(R_1) = Q \)
   \( \text{pre}(R_2) = P' \land b, \) \( \text{post}(R_2) = Q' \)

2. If \( A \) in \( L \) is "\( x_1 = e \)", then \( \text{pre}(A) = \text{post}(A) \frac{e}{x_1} \)

3. If \( A \) in \( L \) is "\( b \rightarrow A_1, A_2 \)", then \( A_1 \) and \( A_2 \) are also in \( L \) and
   \( \text{pre}(A) \land b \rightarrow \text{pre}(A_1) \)
   \( \text{pre}(A) \land \lnot b \rightarrow \text{pre}(A_2) \)
   \( \text{post}(A_1) \rightarrow \text{post}(A) \)
   \( \text{post}(A_2) \rightarrow \text{post}(A) \)

\(^1\) \text{Pre and post functions were first used in soundness proofs by S. Owicki} [OW76]
(4) If $A$ in $L$ is 

\[ \text{“} b \# A_i \text{”} \], then $A_i \in L$ and 

\[ \begin{align*}
\text{pre}(A) \land b & \rightarrow \text{pre}(A_i) \\
\text{pre}(A) \land \sim b & \rightarrow \text{post}(A) \\
\text{post}(A_i) & \rightarrow \text{pre}(A).
\end{align*} \]

(5) If $A$ in $L$ is 

\[ \text{“} \text{begin } A_1 \text{ end”} \], then $A_1 \in L$ and 

\[ \begin{align*}
\text{pre}(A) & \rightarrow \text{pre}(A_1) \\
\text{post}(A) & \rightarrow \text{post}(A).
\end{align*} \]

(6) If $A$ in $L$ is 

\[ \text{“} \text{begin } A_1, A_2, \ldots, A_n \text{ end”} \], then $A_1 \in L$, 

\[ \text{“} \text{begin } A_2, \ldots, A_n \text{ end”} \] 

\[ \in L \] 

and 

\[ \begin{align*}
\text{pre}(A) & \rightarrow \text{pre}(A_1) \\
\text{post}(A) & \rightarrow \text{pre}(\text{begin } A_2, \ldots, A_n \text{ end}) \\
\text{post}(\text{begin } A_1, \ldots, A_n \text{ end}) & \rightarrow \text{post}(A).
\end{align*} \]

(7) If $A$ in $L$ is 

\[ \text{“} \text{exit}_i \text{”} \], then there is a predicate $C_i$ that does not involve any free variables changed by $R_i$ such that 

\[ \begin{align*}
\text{pre}(\text{exit}_i) & = P' \land C_i \\
\text{post}(\text{exit}_i) & = Q' \land C_i.
\end{align*} \]

(8) If $A$ in $L$ is 

\[ \text{“} \text{resume}_i \text{”} \], then there is a predicate $D_i$ that does not involve any free variables that are changed by $R_i$ such that 

\[ \begin{align*}
\text{pre}(\text{resume}_i) & = Q' \land D_i \\
\text{post}(\text{resume}_i) & = P' \land D_i.
\end{align*} \]

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 $s \in P \land b$. If $C_1(A_1, A_2, s') (C_2(A_i, A_2, s'))$ occurs as the $i$-th step in the computation $\text{COMP}(\text{“coroutine } R_1, R_2 \text{ end”}, s)$ then

\[ \begin{align*}
(1) & \quad A_1 = A_1^1; A_1^2; \ldots, A_1^n \quad \text{where each } A_1^i \in L \\
(2) & \quad A_2 = A_2^1; A_2^2; \ldots, A_2^m \quad \text{where each } A_2^i \in L \\
(3) & \quad s' \in \text{pre}(A_1^1) \quad (s' \in \text{pre}(A_2^1)) \\
(4) & \quad \text{post}(A_1^i) \subseteq \text{pre}(A_{i+1}^1), \quad 1 \leq i < n \\
(5) & \quad \text{post}(A_2^i) \subseteq \text{pre}(A_{i+1}^2), \quad 1 \leq i < m \\
(6) & \quad \text{post}(A_n^1) \subseteq Q' \quad (\text{post}(A_n^2) \subseteq Q')
\end{align*} \]

Proof. (By induction on the number of steps in the computation $\text{COMP}(\text{“coroutine } R_1, R_2 \text{ end”}, s).$)
(Basis) The theorem is true initially since \( \text{COMP}("\text{coroutine } R_1, R_2, \text{end"}, s) = C1(R_1, R_2, s), R_1 \in L, s \subseteq P \wedge b \subseteq \text{pre}(R_1), \) and \( \text{post}(R_1) \subseteq Q). \)

(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. \( A_i = A_1; A_2; \ldots; A_i \) where \( A_j \in L \)
2. \( s' \in \text{pre}(A_1), \text{ post}(A_i) \subseteq Q \)
3. \( \text{post}(A_j) \subseteq \text{pre}(A_{j+1}), 1 \leq j < n. \)

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

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

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

\( A_1 \) is "\( \text{exit} \)." In this case the next computation step is \( C2(" A_1; \ldots; A_i, A_2, s' \)". By construction of the pre function we have \( s' \in \text{pre}(\text{exit}) \subseteq P \wedge C_i \). There are two subcases depending on whether the second routine \( R_2 \) of the coroutine has been previously executed.

Case i: Suppose \( A_2 = R_2 \) and that \( R_2 \) has not been previously executed. In this case \( P \wedge b \subseteq \text{pre}(R_2) = \text{pre}(A_2) \). Thus \( s' \in P \) and \( s' \in b \). It follows that \( s' \in P \wedge b \subseteq \text{pre}(A_2) \).

Case ii: Suppose that "\( \text{resume} \)" was the last statement executed when control was previously in \( R_2 \). Assume also that \( \text{pre}(\text{resume}) = P \wedge D_i \) and \( \text{post}(\text{resume}) = P \wedge D_i \). Since \( D_i \) does not contain any free variables changed by \( R_1, s' \in D_i \). Since \( s' \in P \), we have \( s' \in P \wedge D_i \subseteq \text{post}(\text{resume}) \subseteq \text{pre}(A_2) \).

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 "\( \text{resume} \)" statement for \( C2 \) are analogous to the cases considered. Note also that if \( s \in P \wedge b \) and \( \text{COMP}(" \text{coroutine } R_1, R_2, \text{end"}, s) = s' \), then by Theorem 4.3 \( s' \in \text{post}(A_2) \subseteq Q \). Thus \( = \{ P \wedge b \} \text{ coroutine } R_1, R_2 \text{ end } \{ Q \} \). This completes the proof of soundness for the rule of inference \( C1 \) for coroutines.

5. Completeness

We 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 Sects. 2 and 3. Furthermore, these proofs of partial correctness do not involve the use of history variables.

Since our primary interest is the coroutine statement, we will restrict our attention to LFC programs of the form coroutine $R_1, R_2$ end. We will represent the computation of such a program with initial state $s_0$ by

$$\langle R_1, R_2, s_0 \rangle \langle R_1^1, R_2^1, s_1 \rangle \ldots \langle R_1^i, R_2^i, s_i \rangle \ldots$$

where detailed rules for deriving $\langle R_1^{i+1}, R_2^{i+1}, s_{i+1} \rangle$ from $\langle R_1^i, R_2^i, s_i \rangle$ may be obtained from the semantics for coroutines given in Sect. 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 $R_1, R_2$ 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 Sect. 4. Intuitively, PRE($A_1$) (POST($A_1$)) is the set of program states which can occur 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 $R_1$, we may formally define PRE($A_1$) and POST($A_1$) by

$$\text{PRE}(A_1) = \{s_0^* | \text{there is a computation of } A \text{ of the form} \langle R_1, R_2, s_0 \rangle \langle R_1^1, R_2^1, s_1 \rangle \ldots \langle A_1; R_1^*, R_2^*, s^* \rangle \text{ and } s \in P \}.$$  

$$\text{POST}(A_1) = \{s_0^* | \text{there is a computation of } A \text{ of the form} \langle R_1, R_2, s_0 \rangle \langle R_1^1, R_2^1, s_1 \rangle \ldots \langle A_1; R_1^*, R_2^*, s^* \rangle \ldots \langle R_1^*, R_2^*, s^* \rangle \text{ and } s \in P \}.$$  

Analogous definitions may also be given when $A_1$ is a substatement of $R_2$.

5.1. Definition. The assertion language $A_L$ is expressible with respect to interpretation I iff for all programs $A$ of the form coroutine $R_1, R_2$ end and all predicates $P$ in $A_L$, PRE($A_1$) and POST($A_1$) are expressible by formulas of $A_L$ whenever $A_1 \in \text{SUB}(A)$.

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$R_1, R_2$ end, then
(1) \[ P = \text{PRE}(R_1), \quad \text{POST}(R_1) \subseteq Q \]

(2) \[ \text{PRE}(x := c) = \text{POST}(x := c) \frac{c}{x} \]

(3) \[ \begin{align*}
\text{PRE}(b \rightarrow A_1, A_2) \land h &= \text{PRE}(A_1) \\
\text{PRE}(b \rightarrow A_1, A_2) \land \neg h &= \text{PRE}(A_2) \\
\text{POST}(A_1) \subseteq &\text{POST}(b \rightarrow A_1, A_2) \\
\text{POST}(A_2) \subseteq &\text{POST}(b \rightarrow A_1, A_2) 
\end{align*} \]

(4) \[ \begin{align*}
\text{PRE}(b \ast A) \land h &= \text{PRE}(A) \\
\text{PRE}(b \ast A) \land \neg h &= \text{POST}(b \ast A) \\
\text{POST}(A) &= \text{PRE}(b \ast A) 
\end{align*} \]

(5) \[ \text{PRE}(\text{begin } A \text{ end}) = \text{PRE}(A) \]
[\text{POST}(\text{begin } A \text{ end}) = \text{POST}(A)]

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

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

(7) \[ \begin{align*}
\bigvee_i &\text{PRE}(\text{exit}_i) = \text{PRE}(R_2) \lor \bigvee_j \text{POST}(\text{resume}_j) \\
\bigvee_i &\text{POST}(\text{exit}_i) = \bigvee_j \text{PRE}(\text{resume}_j) \lor \text{POST}(R_2). 
\end{align*} \]

The index \(i\) in (7) ranges over all distinct \text{exit} statements in \text{SUB}(R_1). The index \(j\) ranges over all distinct \text{resume} statements in \text{SUB}(R_2).

We are now ready to begin the proof of relative completeness. Assume that \(\{P\} \text{ coroutine } R_1, R_2 \text{ end } \{Q\}\) is true; we must show that it is provable using the axioms and rules of inference in Sects. 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:

\[
\begin{align*}
i &:= 0; \quad j := 0; \\
\text{coroutine} &
\begin{align*}
\text{begin} \\
\ldots \\
R_1 &
\begin{align*}
i &:= i_0; \\
\text{exit}_{i_0}; \\
\end{align*} \\
\text{end}, \\
\text{begin} \\
\ldots \\
R_2 &
\begin{align*}
\text{begin} \\
\ldots \\
\text{begin} \\
\ldots \\
\end{align*} \\
\text{end} \\
\end{align*}
\end{align*}
\]
We will also assume that \( P \) has been modified to reflect the fact that \( i \) and \( j \) are 0 initially. Let

\[
P' = \bigvee_i \text{PRE}(\text{exit}_i) \lor \text{PRE}(R_2) \lor \bigvee_j \text{POST}(\text{resume}_j)
\]

\[
Q' = \bigvee_i \text{POST}(\text{exit}_i) \lor \bigvee_j \text{PRE}(\text{resume}_j) \lor \text{POST}(R_2)
\]

\[
b = \{j = 0\}.
\]

By the expressibility \( P' \), \( Q' \), and \( b \) are representable by formulas of AL. Note also that all of the following conditions are satisfied:

\[
P' \land (i = i_0) \Rightarrow \text{PRE}(\text{exit}_{i_0})
\]

\[
Q' \land (i = i_0) \Rightarrow \text{POST}(\text{exit}_{i_0})
\]

\[
P' \land b \Rightarrow \text{PRE}(R_2)
\]

\[
P \land b \Rightarrow \text{PRE}(R_1)
\]

\[
\text{POST}(R_1) \rightarrow Q
\]

\[
\text{POST}(R_2) \rightarrow Q'.
\]

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

\[
\{P'\} \text{exit}_{i_0} \{Q'\} \Rightarrow \{\text{PRE}(R_1)\} R_1\{\text{POST}(R_1)\}
\]

and

\[
\{Q'\} \text{resume}_{i_0} \{P'\} \Rightarrow \{\text{PRE}(R_2)\} R_2\{\text{POST}(R_2)\}.
\]

(5.2)

(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 \( R_1 \). Let \( A \) be a substatement of \( R_1 \); we will show that \( \{P'\} \text{exit}_{i_0} \{Q'\} \Rightarrow \{\text{PRE}(A)\} A\{\text{POST}(A)\} \).

If \( A \) is any statement but an \( \text{exit} \) or \( \text{resume} \) statement, this is trivial. For example, suppose that \( A \) is \( "b \rightarrow A_1, A_2" \), then

\[
\{\text{PRE}(A_1)\} A_1\{\text{POST}(A_1)\}
\]

and

\[
\{\text{PRE}(A_2)\} A_2\{\text{POST}(A_2)\}
\]

are provable by the induction hypothesis. Thus,

\[
\{\text{PRE}(b \rightarrow A_1, A_2) \land b\} A_1\{\text{POST}(b \rightarrow A_1, A_2)\}
\]

and

\[
\{\text{PRE}(b \rightarrow A_1, A_2) \land \lnot b\} A_2\{\text{POST}(b \rightarrow A_1, A_2)\}
\]

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

If \( A \) is the statement \( \text{exit}_{i_0} \), then we may use the coroutine axiom C2 and the hypothesis \( \{P'\} \text{exit}_{i_0} \{Q'\} \) to deduce \( \{P' \land i = i_0\} \text{exit}_{i_0} \{Q' \land i = i_0\} \). Since
$P \land (i = i_0) \equiv \text{PRE}(\text{exit}_0)$ and $Q \land (i = i_0) \equiv \text{POST}(\text{exit}_0)$, we conclude that $\{\text{PRE}(\text{exit}_0)\} \text{ exit}_0 \{\text{POST}(\text{exit}_0)\}$ 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 variables.

6. The "Histo" Example

Frequently, it is desirable to establish the correctness of the first routine $R1$ of a coroutine without knowing what the second routine $R2$ is. Examples (including the program "Histo" which we discuss below) illustrating this point are given in [CL73]. One might argue that, given a particular $R2$, it would be simple to prove coroutine $R1$, $R2$ end correct without the use of history variables, but to first prove $R1$ correct, in isolation, and then to fit in $R2$ without regard to the internal workings of $R1$ might require the use of history variables to describe the function of $R2$. Although it is undoubtedly more difficult to prove the correctness of $R1$ when the code for $R2$ is not available, we show in this section that history variables are not needed in this more general case and that proofs constructed without the use of history variables are as natural as proofs using history variables.

To illustrate how history variables can be avoided when code for one routine of the coroutine is not available, we consider the example "Histo" discussed in [CL73]. In this example the routine $R2$ generates values of the global variable "obs". The values assigned to "obs" are guaranteed to be in the interval $a \leq \text{obs} < b$, but no other assumptions are made concerning the way in which the values are generated. Routine $R1$ constructs a histogram with $N$ intervals for the observations produced by $R2$.

count := 0; obs := a;

**coroutine**

begin
  $i$ := 0;
  while $i < N$ do begin $i$ := $i + 1$; $H[i]$ := 0 end;
  while count $<$ limit do
    begin
      exit;
      $i$ := (obs $-$ a)* $N$/(b $-$ a) + 1;
      $H[i]$ := $H[i]$ + 1;
      count := count + 1
    end
  end,
  $R2$
end.

Note that the "Histo" program is an example of the use of coroutines to model producer-consumer processes. This is one of the most frequent applications of coroutines.
The proof of correctness for "Histo" given by Clint [CL73] uses a history variable \( s \) to record the sequence of observations produced by \( R2 \). Thus the first routine \( R1 \) of the coroutine is modified to obtain

\[
\begin{align*}
&\text{begin} \\
&\quad i := 0; \\
&\quad \text{while } i < N \text{ do begin } i := i + 1; \ H[i] := 0 \text{ end;} \\
&\quad \text{while } \text{count} < \text{limit do} \\
&\quad \quad \text{begin} \\
&\quad\quad \quad \text{exit;} \\
&\quad\quad \quad s := s|\text{obs}; \\
&\quad\quad \quad i := (\text{obs} - a)^* N/(b-a) + 1; \\
&\quad\quad \quad H[i] := H[i] + 1; \\
&\quad\quad \quad \text{count} := \text{count} + 1 \\
&\quad \quad \text{end} \\
&\quad \text{end} \\
\end{align*}
\]

where "\(|\)" is the concatenation operator for sequences of integers. Note, however, that each value of "\( \text{obs} \)" generated by routine \( R2 \) is a function of the current value of "\( \text{count} \)". Thus, we can avoid the use of history variables simply by introducing a function \( f \) such that \( a \leq f(\text{count}) < b \) for \( 0 \leq \text{count} \leq \text{limit} \), i.e.,

\[
\begin{align*}
&\text{count} := 0; \quad \text{obs} := a; \\
&\textbf{coroutine} \\
&\quad \text{begin} \\
&\quad \quad i := 0; \\
&\quad \quad \text{while } i < N \text{ do begin } i := i + 1; \ H[i] := 0 \text{ end;} \\
&\quad \quad \text{while } \text{count} < \text{limit do} \\
&\quad\quad \quad \text{begin} \\
&\quad\quad\quad \text{exit;} \\
&\quad\quad\quad i := (\text{obs} - a)^* N/(b-a) + 1; \\
&\quad\quad\quad H[i] := H[i] + 1; \\
&\quad\quad\quad \text{count} := \text{count} + 1 \\
&\quad\quad \text{end} \\
&\quad \quad \text{end} \\
&\quad \text{end} \\
&\quad \text{while } \text{true} \text{ do} \\
&\quad \quad \text{begin } \text{obs} := f(\text{count}); \ \text{resume} \text{ end} \\
&\quad \text{end} \\
&\textend
\]

Although function \( f \) will serve much the same purpose in the correctness proof as the auxiliary variable \( s \), it is no longer necessary to introduce the new data type \textit{sequence of real} or the new operation of concatenation on this data type. The use of such functions also makes explicit how the values generated by \( R2 \) depend on variables global to the coroutine.

Let \( X_j = a + j^* (b-a)/N \) for \( 0 \leq j \leq N \) be the interval end points for the histogram. We prove that
\[ \{ \text{count} = 0 \land \text{obs} = a \} \]

\[ \forall j[0 \leq j < N \rightarrow H[j] = \text{number(} \text{limit, } X_j, X_{j+1} \text{)} \land \text{number(} \text{limit, } a, b \text{)} = \text{limit} \} \]

where \( \text{number}(k, X_i, X_m) \) is the function defined by

\[
\text{number}(k, X_i, X_m) = \begin{cases} 
0 & \text{if } k = 0 \\
1 + \text{number}(k-1, X_i, X_m) & \text{if } k > 0 \text{ and } X_i \leq f(k) < X_m \\
\text{number}(k-1, X_i, X_m) & \text{otherwise}
\end{cases}
\]

Let

1. \( P = \{ \text{count} = 0 \land a = \text{obs} \} \)
2. \( Q = \forall j[0 \leq j < N \rightarrow H[j] = \text{number(} \text{limit, } X_j, X_{j+1} \text{)} \land \text{number(} \text{limit, } a, b \text{)} = \text{limit} \} \)
3. \( P' = Q' = \{ a \leq \text{obs} < b \} \)
4. \( b = \text{true} \)

Then it is not difficult to show that

\[ \{ P' \} \text{ exit } \{ Q' \} \leftarrow \{ P \land b \} \text{ R1} \{ Q \} \]  \hspace{1cm} (1)

and

\[ \{ Q' \} \text{ resume } \{ P' \} \leftarrow \{ P' \land b \} \text{ R2} \{ Q' \} \]  \hspace{1cm} (2)

The invariant \( I1 \) for the while loop in \( Q1 \) that contains the exit statement is

\[ I1 = \forall j[0 \leq j < N \rightarrow H[j] = \text{number(} \text{count, } X_j, X_{j+1} \text{)} \land \text{number(} \text{count, } a, b \text{)} = \text{count} \land \text{count} \leq \text{limit} \land a \leq \text{obs} < b \} \]

Note that by axiom \( C2 \) we may deduce

\[ \{ a \leq \text{obs} < b \} \text{ exit } \{ a \leq \text{obs} < b \} \leftarrow \{ I1 \} \text{ exit } \{ I1 \} . \]

The invariant \( I2 \) of the loop in the routine \( Q2 \) is simpler: \( I2 = \{ a \leq \text{obs} < b \} \).

Filling in the details of the proofs of (1) and (2) is straightforward and will be left to the reader. From (1) and (2) we may deduce \( \{ P \} \text{ coroutine R1, R2 end} \{ Q \} \) by rule \( C1 \).

7. 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 pro-
gramming 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 Sect. 5 extend to handle local recursive 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.

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 synchronization 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 de Bakker and Meertens [DE73] have shown that a sound and relatively complete proof system may be given for nondeterministic ALGOL that 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 that 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.

Acknowledgement. The author is grateful to Robert Constable and Susan Owicki for their help with various parts of this paper.

References

[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


[OW76a] Owicki, S.: A consistent and complete deductive system for the verification of parallel programs. 8th Annual Symposium on Theory of Computing, 1976


Received April 13, 1978 / Revised October 5, 1979