Abstract. The paper discusses local correctness criteria and local correctness proofs of semicoroutines subject to certain simplifying assumptions. A nontrivial worked example is given.

1. Introduction.

An important goal in programming is to construct programs that are easy to understand. A good way to achieve understanding of a program is to prove that it is correct. Since a program easily proved correct is probably easy to understand, one looks for program structuring mechanisms which admit simple and powerful proof rules. A simple correctness proof is not necessarily easy to find, unless certain key assertions about the program variables are given. Therefore such assertions should be provided with the program text as comments, and in such quantity that the construction of a correctness proof is trivial.

In the present paper we shall discuss correctness criteria and correctness proofs of coroutines. More specifically we consider so called semicoroutines as defined by Wang and Dahl [1] and expressed in a slightly modified Simula 67. A class of semicoroutines is thus defined by a class declaration, say C.

```plaintext
class C; (class body);
```

A semicoroutine of the class C is a dynamic instance of the class body, and is named by a reference variable, say X.

```plaintext
X := new C
```

In general any number of semicoroutines of the same class (or different ones) may coexist. We assume for simplicity that each semicoroutine is named by one and only one reference variable throughout its life.

Consider a process consisting of the execution of a master program M
and a single semicoroutine \( X \). It can be viewed as two separate processes operating in parallel, but such that only one is executing at a time, while the other is waiting.

Control is transferred from \( M \) to \( X \) at the time when \( X \) is generated, i.e., when \( M \) executes the generator "new C", and whenever \( M \) executes "call(X)". Control returns to \( M \) whenever \( X \) executes a "detach" statement, and at the time of termination of \( X \).

\[
\begin{align*}
M: & \quad \underline{X} :- \underline{newC} & \quad call(X)_1 & \quad \ldots & \quad call(X)_v & \quad T \\
& \quad P_0 & \quad Q_0 & \quad P_1 & \quad Q_1 & \quad P_v & \quad Q_v \\
X: & \quad \text{body begin} & \quad \text{detach}_1 & \quad \text{detach}_2 & \quad \text{body end}
\end{align*}
\]

The figure gives a pictorial representation of the sequence of events in time. There is a mapping which maps each event call(X) onto some occurrence of call(X) in the program text of \( M \), and similarly for the other events of \( M \) and \( X \).

Understanding the processes \( M \) and \( X \) requires a prior knowledge (except in special cases) of the interface between them. The interface consists of assertions \( P_0, P_1, P_2, \ldots \) and \( Q_0, Q_1, Q_2, \ldots \) characterizing the state vector at the times of transition of control from \( M \) to \( X \) and from \( X \) to \( M \) respectively (see the figure). Assuming that \( M \) calls \( X \) \( v \) times, after which \( X \) terminates, we can formulate a proof rule for the process as a whole with a formalism similar to that of Hoare.

\[
P_0(X :- \underline{newC}) Q_0, \forall t \in [1, v]: P_t[call(X)_t]Q_t \vdash S[M]T,
\]

\[
\forall t \in [1, v]: Q_{t-1}[\text{detach}_t]P_t \vdash P_0[\text{body of } C]Q_v
\]

\[
S[M,C]T
\]

The rule as it stands leaves little hope of providing a correctness proof of the semicoroutine separately from \( M \), or any clue as to what the criterion of its correctness should be. A major difficulty is the fact that the assertions \( P_t \) and \( Q_t \) in general are predicates on the whole state vector. This reflects the fact that Simula 67 permits the entire state vector to be accessible to both