CORRECTNESS PROOFS OF CSP PROGRAMS

N. SOUNDRARAJAN

Department of Computer and Information Science, The Ohio State University, Columbus, OH 43210, U.S.A.

Abstract. In a research report we have proposed an axiomatic semantics for the language of communicating sequential processes (CSP) of Hoare (1978). In this paper, we use the axiomatic semantics to prove the correctness of a number of CSP programs.

1. Introduction

The language of communicating sequential processes (CSP) proposed by Hoare [2] is one of the most elegant languages for parallel programming. In [3] we have proposed an axiomatic semantics for CSP; in this paper we use the semantics of [3] to prove the correctness of a number of CSP programs.

The paper is organized as follows: in Section 2 we summarize the axiomatic semantics of [3]. In Section 3, we prove the correctness of a program for distributed partitioning of sets. In Section 4, the correctness of a program for the distributed computation of the g.c.d of \( n \) numbers is proved. These two examples are taken from Apt et al. [1] and the reader may wish to compare the correctness proofs of these two examples given in this paper with the proofs in Apt et al. [1].

The remaining programs we deal with are from Hoare [2]. In Section 5 we consider a program that simulates a bounded buffer. In Section 6 we deal with a process which behaves (as we show) as an integer semaphore. The final section contains some concluding remarks on our approach to the semantics of CSP.

2. Axiomatic semantics of CSP

Consider a CSP program \([P_1|| \cdots ||P_n]\), \(P_1, \ldots, P_n\) being the communicating processes; we assume that the \(P_i\)'s are strictly sequential; thus parallel composition exists only at the outermost level. The communication sequences exchanged between the processes will play a rather vital role in the semantics; \(h_i\) will denote the communication sequence associated with the process \(P_i\). Thus \(h_i\) is a sequence of elements of the form \((i, j, u)\) and \((j, i, v)\) where the former element corresponds to the number \(u\) being sent by \(P_i\) to \(P_j\), while the latter element corresponds to the number \(v\) received by \(P_i\) from \(P_j\). If there are loops in \(P_n\), \(h_i\) may also include
elements of the kind \((i, T, \tau)\), \(\tau\) being just a constant symbol and \(T\) a subset of \(\{1, \ldots, i-1, i+1, \ldots, n\}\). Such an element, if it occurs in \(h_i\), denotes that a loop in \(P_i\) terminated because each of the processes whose index appears in \(T\) had terminated. (Recall the CSP convention that a loop in \(P_i\) terminates if all the guards fail; a guard may fail either because the boolean portion is false or the process addressed in the I/O portion of the guard has terminated.)

Next consider the axioms and rules of inference corresponding to the various constructs which may appear in \(P_i\). The pre- and post-conditions in \(P_i\) will be predicates over the local state of \(P_i\) (recall that there are no shared variables in \(P_i\)), and the sequence \(h_i\).

A1. Assignment
\[
\{p^*\}x := e\{p\}.
\]

A2. Skip
\[
\{p\} \text{skip} \{p\}.
\]

A3. Output
\[
\{p^h_{h_i \rightarrow \langle i, i, y \rangle}\}P_i!y\{p\}.
\]

The effect of the output statement \(P_i!y\) is to concatenate the element \((i, j, y)\) to the right end of \(h_i\). The symbol \(\rightarrow\) denotes concatenation of an element to the right end of a sequence. (Similarly \(\leftarrow\) will denote concatenation to the left end of a sequence.)

A4. Input
\[
\{\forall k \cdot p_{h_i \rightarrow \langle i, i, k \rangle}^k\}P_i?x\{p\}.
\]

The effect of the input statement is to concatenate an element of the kind \((j, i, k)\) to \(h_i\) and to replace the value of \(x\) by \(k\). The universal quantifier over \(k\) corresponds to the fact that, in \(P_i\) we have no knowledge of the number that \(P_i\) will send to \(P_i\).

R1. Sequential composition
\[
\frac{\{p\}S_1\{p'\}, \{p'\}S_2\{q\}}{\{p\}S_1; S_2\{q\}}.
\]

R2. Guarded selection
\[
\frac{\{p \land B(g_i)\}C(g_i); S_1\{q\}, l = 1, \ldots, m}{\{p\} \left[ \square (l = 1, \ldots, m \land g_i \rightarrow S_i) \right] \{q\}}.
\]
There are $m$ guards; each guard $g$ is of the form $b; P_i ? x$ or $b; P_i ! y$ or $b$ where $b$ is a boolean expression. In each case $B(g_i)$ is just $b$. $C(g_i)$ is $P_i ? x$, $P_i ! y$ or $\text{skip}$ if $g_i$ is respectively $b; P_i ? x$, $b; P_i ! y$ or $b$.

R3. Guarded repetition

\[
\left[ p \land \left( \bigwedge_{l \in \text{PB}} \neg b_l \right) \right] \Rightarrow q_{h_1 \rightarrow l, (i, j) \exists l < m. \left[ B(g_l) \land C(g_l) \neq \text{skip} \land D(g_l) = \text{false} \right]}
\]

\[
\frac{\{ p \land B(g_i) \} C(g_i) ; S_i \{ p \}, l = 1, \ldots, m}{\{ p \} \ast \{ \square(l = 1, \ldots, m) g_l \rightarrow S_i \{ q \} \}}.
\]

PB is the set of indices of those guards that are purely boolean (i.e., of the form $b_i$ and hence $C(g_i) = \text{skip}$); $D(g_i)$ is $j$ if $C(g_i)$ is $P_i ? x$ or $P_i ! y$ (and, say, 0 if $C(g_i) = \text{skip}$). Thus the purpose of the first line of R3 is to ensure that the element $(i, T, \tau)$ is concatenated to $h_i$ when the loop terminates, $T$ being the set of indices of those processes with which $P_i$ was willing to continue communicating, but which had terminated forcing the loop in $P_i$ to terminate.

R4. Rule for parallel composition

\[
\frac{\{ p_i \land h_i = \varepsilon \} P_i \{ q_i \}, i = 1, \ldots, n}{\{ \bigwedge_{i=1}^n p_i \} P_1 \parallel \cdots \parallel P_n \{ \bigwedge_{i=1}^n q_i \land \text{Compat}(h_1, \ldots, h_n) \}}
\]

The clause $\text{Compat}(h_1, \ldots, h_n)$ simply expresses the fact that the sequences $h_1, \ldots, h_n$ are consistent or compatible with each other. (Note that $\varepsilon$ is the empty sequence).

More formally, $\text{Compat}(h_1, \ldots, h_n)$ may be defined recursively as follows:

$\text{Compat}(h_1, \ldots, h_n)$ is

(i) true if $h_1 = \cdots = h_n = \varepsilon$;

(ii) if $\exists i, j$ such that $h_i, h_j \neq \varepsilon$ and first($h_i$) = first($h_j$) where first($h_i$) is the leftmost element of $h$, then $\text{Compat}(h_1, \ldots, h_{i-1}, \text{rest}(h_i), h_{i+1}, \ldots, h_{j-1}, \text{rest}(h_j), h_{j+1}, \ldots, h_n)$ where rest($h$) is the sequence got by deleting the leftmost element of $h$;

(iii) If $\exists i$ such that first($h_i$) = $(i, T, \tau)$ and for all $j$ belonging to $T$, $h_j = \varepsilon$, then $\text{Compat}(h_1, \ldots, h_{i-1}, \text{rest}(h_i), h_{i+1}, \ldots, h_n)$.

(iv) if none of the clauses above is applicable then false.

(Note: If clause (ii) is applicable and there are, say, 2 pairs $(i, j)$, $(i', j')$ such that first($h_i$) = first($h_i'$) and first($h_j$) = first($h_j'$), we may use either pair, since it is easy to see that the final value of $\text{Compat}$ will be the same in both cases. A formal proof of this result (by induction on the lengths of the sequences) is left to the reader).
Apart from these, we also have the usual 'logical' rules:

R5. Consequence

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

R6. Conjunction

$$\frac{\{p_1\} S\{q_1\}, \{p_1\} S\{q_2\}}{\{p_1\} S\{q_1 \land q_2\}}$$

R7. Disjunction

$$\frac{\{p_1\} S\{q\}, \{p_2\} S\{q\}}{\{p_1 \lor p_2\} S\{q\}}$$

3. A set partitioning program

$S_0, T_0$ are disjoint sets, $S_0 \neq \emptyset$. The purpose of the program is to partition the sets in such a way that, when it finishes, we have $[S \cup T = S_0 \cup T_0] \land [\max(S) < \min(T)] \land [\|S\| = \|S_0\|] \land [\|T\| = \|T_0\|]$, $|R|$ being the cardinality of $R$. $S, T$ have initial values $S_0, T_0$.

The following functions on sequences will be useful in the proofs of correctness:

- $\|h\|$ = number of elements in the sequence $h$;
- $\text{Elem}(h, i) = i$th element of $h$;
- $\text{Elem}(h, i) = \text{Elem}(h, \|h\| - i + 1) = i$th element from the right end of $h$;
- $\text{Val}(h, i) = \text{value of the number being communicated in the } i$th element of $h$;
- $\text{Val}(h, i) = \text{Val}(h, \|h\| - i + 1)$;
- $\text{Dir}(h, i) = \text{direction}$ of the $i$th element of $h$. Thus if $\text{Elem}(h, i)$ is $(k, 1, u)$, $\text{Dir}(h, i) = (k, 1)$; and if $\text{Elem}(h, i) = (k, T, r)$, then $\text{Dir}(h, i) = (k, r)$;
- $\text{Dir}(h, i) = \text{Dir}(h, \|h\| - i + 1)$.

The program for set partitioning annotated with the assertions which hold at the intermediate points is as follows:

The program is $P_1||P_2$ where

$P_1 := \{h_i = e, S = S_0 \land S_0 \cap T_0 = \emptyset\}$

$$\text{mx} := \max(S); P_2!mx, S := S - \{\text{mx}\}; P_2?x;$$

$$S := S \cup \{x\}; \text{mx} := \max(S);$$

$L_1$

$$[\text{mx} > x \Rightarrow P_2!\text{mx}, S := S - \{\text{mx}\}; P_2?x, S := S \cup \{x\}; \text{mx} := \max(S)]$$

$L_1 \land \text{mx} = x$
\[ P_2 ::= \{ R_2 \} \]
\[ [P_1 ? y \rightarrow T := T \cup \{ y \}; \ mn := \min(T); \ P_1 ! mn \ ; \ T := T - \{ mn \}] \]
\[ \{ R_3 \} \]

where

\[ R_1 \equiv [T_1(h_1, S) \land mx = \max(S) \land \|h_1\| \geq 2 \land x = \Valb(h_1, 1) \land S_0 \cap T_0 = \emptyset] \]

and

\[ T_1(h_1, S) \equiv [[h_1 = \varepsilon \land S = S_0] \]
\[ \lor [\text{Even}(\|h_1\|) \land v \in S \land u > \max(S - \{ v \})] \]
\[ \land \text{Dirb}(h_1, 1) = (2, 1) \land \text{Dirb}(h_1, 2) = (1, 2) \]
\[ \land [T_1(Lr(Lr(h_1)), (S - \{ v \}) \cup \{ u \}) \]
\[ \lor T_1(Lr(Lr(h_1)), S \cup \{ u \})] \]

where \( v \) denotes \( \Valb(h_1, 1) \), \( u \) denotes \( \Valb(h_1, 2) \); \( Lr(h) \) is the sequence got by deleting the rightmost element from \( h \). \( T_1 \) is defined in an inductive fashion. Thus a given pair \((h_1, S)\) satisfies \( T_1 \) if \( h_1 = \varepsilon \land S = S_0 \), or if the last two values in \( h_1 \) satisfy the clauses \( v \in S \land u > \max(S - \{ v \}) \) and either the pair \((Lr(Lr(h_1)), (S - \{ v \}) \cup \{ u \})\) or the pair \((Lr(Lr(h_1)), S \cup \{ u \})\) satisfies \( T_1 \).

Inductive definitions of this kind will occur throughout this paper. The reader should have no difficulty in seeing that \( R_1 \) is indeed a loop invariant for \( P_1 \), and that \( R_1 \land mx \leq x \) is the proper post-condition.

Next we define \( R_2, R_3 \):

\[ R_2(h_2, T) \equiv [h_2 = \varepsilon \land T = T_0] \]
\[ \lor [\text{Even}(\|h_2\|) \land v < \min(T) \land u \in T \cup \{ v \}] \]
\[ \land \text{Dirb}(h_2, 1) = (2, 1) \land \text{Dirb}(h_1, 2) = (1, 2) \]
\[ \land [R_2(Lr(Lr(h_2)), (T \cup \{ v \}) - \{ u \}) \]
\[ \lor R_2(Lr(Lr(h_2)), T \cup \{ v \})] \],

\( u, v \) denoting \( \Valb(h_2, 2) \) and \( \Valb(h_2, 1) \) respectively;

\[ R_3 \equiv [\text{Elemb}(h_2, 1) = (2, \{ 1 \}, \tau) \land R_2(Lr(h_2), T)]. \]

Then by the rule for parallel composition, we get

\[ \{ S = S_0 \land T = T_0 \land S_0 \cap T_0 = \emptyset \} P_1 \| P_2 \{ R_1(h_1, S) \land max(S) \leq x \]
\[ \land R_3(h_2, T) \land \text{Compat}(h_1, h_2) \}. \]

Using \( R_1, R_3 \) and \( \text{Compat} \), it is easy to see that \( h_1 \) and \( Lr(h_2) \) consist of alternative elements of the form \((1, 2, k)\) and \((2, 1, l)\), the corresponding elements of \( h_1 \) and
Lr(h₂) being identical. Thus the post-condition of \( P_1 \| P_2 \) implies the following:

\[
\max(S) \leq x = \text{Valb}(h_1, 1) = \text{Valb}(\text{Lr}(h_2), 1) < \min(T).
\]

Thus, \( \max(S) < \min(T) \).

To see that the post-condition also implies

\[
[S \cup T = S_0 \cup T_0] \land [|S| = |S_0|] \land [|T| = |T_0|]
\]

we use induction on the length of the sequences. Since \( h_1 = \text{Lr}(h_2) \), we use \( h_1 \) to denote \( \text{Lr}(h_2) \) also. Thus, the post-condition implies

\[ R_1(h_1, S) \land R_2(h_1, T) \land [S_0 \cap T_0 = \emptyset]. \]

If \( h_1 = \epsilon \), then \( R_1, R_2 \) imply \( S = S_0 \) and \( T = T_0 \). Thus this case is trivial. If not, let

\[
u = \text{Valb}(h_1, 2), v = \text{Valb}(h_1, 1); \text{ also let}
\]

\[
S' = (S - \{v\}) \cup \{u\}, \quad S'' = S \cup \{u\}, \quad T' = (T \cup \{v\}) - \{u\}, \quad T'' = T \cup \{v\}.
\]

Then it is easy to see, using the clauses \( v \in S, u \in T \cup \{v\} \) of \( R_1 \) and \( R_2 \), that

\[
S \cup T = S' \cup T' = S' \cup T'' = S'' \cup T' = S'' \cup T''
\]

and

\[
|S| \leq |S'|, \quad |T| \leq |T'|, \quad |S| \leq |S''|, \quad |T| \leq |T''|.
\]

Also one of the following four predicates is true:

\[
R_1(\text{Lr}(\text{Lr}(h_1))), S') \land R_2(\text{Lr}(\text{Lr}(h_1)), T'),
\]

\[
R_1(\text{Lr}(\text{Lr}(h_1))), S') \land R_2(\text{Lr}(\text{Lr}(h_1)), T''),
\]

\[
R_1(\text{Lr}(\text{Lr}(h_1))), S'' \land R_2(\text{Lr}(\text{Lr}(h_1)), T'),
\]

\[
R_1(\text{Lr}(\text{Lr}(h_1))), S'' \land R_2(\text{Lr}(\text{Lr}(h_1)), T'').
\]

In all four cases, it is easy to verify, using the inductive hypothesis, that

\[
S \cup T = S_0 \cup T_0 \land |S| \leq |S_0| \land |T| \leq |T_0|,
\]

which, in conjunction with \( S_0 \cap T_0 = \emptyset \), gives the desired result.

4. Distributed computation of the gcd of \( n \) numbers

The following program is meant to compute the gcd of \( n \) numbers \( \sigma_1, \ldots, \sigma_n \). The program does not terminate properly; instead all the processes reach a deadlock at which point the gcd is available (as the value of \( x_1, \ldots, x_n \)).
The program is $P := P_1 \cdot \cdots \cdot P_n$, where $P_i$ is

\[
\{h_i = \varepsilon\}
\]

\[
x_i := \sigma_i; \ rsi_i := \text{true}; \ rsr_i := \text{true};
\]

\[
\{R_i\}
\]

\[
[rsl_i; P_{i-1}!x_i \rightarrow rsl_i := \text{false}]
\]

\[
\square \ rsr_i; P_{i+1}!x_i \rightarrow rsr_i := \text{false}
\]

\[
\square \ P_{i-1}?y_i \rightarrow [y_i \equiv x_i \rightarrow \text{skip}]
\]

\[
\square y_i < x_i \rightarrow [y_i | x_i \rightarrow x_i := y_i]
\]

\[
\square y_i \nmid x_i \rightarrow x_i := x_i \mod y_i;
\]

\[
rsl_i := \text{true}; \ rsr_i := \text{true}\]

\[
\square P_{i+1}?y_i \rightarrow [y_i \equiv x_i \rightarrow \text{skip}]
\]

\[
\square y_i < x_i \rightarrow [y_i | x_i \rightarrow x_i := y_i]
\]

\[
\square y_i \nmid x_i \rightarrow x_i := x_i \mod y_i;
\]

\[
rsl_i := \text{true}; \ rsr_i := \text{true}\]

(Note: the $i \pm 1$ above is modulo $n$. ‘$y | x$’ denotes ‘$y$ divides $x$’, and ‘$y \nmid x$’ denotes ‘$y$ does not divide $x$’). And

\[
R_i = [h_i \ \text{seq}\{(i, i \pm 1), (i \pm 1, i)\}] \land [x_i = f_i(\sigma_i, h_i)] \land T_i(h_i)
\]

\[
[\neg rsl_i \Rightarrow x_i = \text{Valb}(h_{i/(i,i-1)}, 1)] \land [\neg rsr_i \Rightarrow x_i = \text{Valb}(h_{i/(i,i+1)}, 1)].
\]

The first clause says that $h_i$ is a sequence of elements of the form $(i, i \pm 1, l)$ and $(i \pm 1, i')$; $h_{i/(i,i \pm 1)}$ is got from $h_i$ by removing all elements except those of the form $(i, i \pm 1, l)$. The predicate $T_i$ is defined as follows:

\[
T_i(h_i) = [h_i = \varepsilon]
\]

\[
\lor [\text{Dirb}(h_i, 1) = (i \pm 1, i) \land T_i(Lr(h_i))]
\]

\[
\lor [\text{Dirb}(h_i, 1) = (i, i \pm 1)]
\]

\[
\land \text{Valb}(h_i, 1) = f_i(\sigma_i, Lr(h_i)) \land T_i(Lr(h_i))
\]

where

\[
f_i(z, h_i) = \begin{cases} 
\text{if } h_i = \varepsilon & \text{then } z \\
\text{else if } \text{Dirb}(h_i, 1) = (i, i \pm 1) & \text{then } f_i(z, Lr(h_i)) \\
\text{else if } \text{Dirb}(h_i, 1) = (i \pm 1, i) & \text{then } g(f_i(z, Lr(h_i)), \text{Valb}(h_i, 1))
\end{cases}
\]
where
\[ g(z, m) = \text{if } n \geq z \text{ then } z \text{ else if } m \mid z \text{ then } m \text{ else } z \mod m. \]

Again the reader should have no difficulty in verifying the invariance of \( R_i \). At the
time of deadlock, we have
\[
\bigwedge_{i=1}^n \left[ R_i \land \neg rs_i \land \neg rs_{i+1} \right] \land \text{Compat}(h_1, \ldots, h_n). \tag{1}
\]

We need to show that at this time we shall have
\[
[x_1 = \cdots = x_n = \gcd(\sigma_1, \ldots, \sigma_n)].
\]

First we shall show \( x_i \geq x_{i+1} \) for all \( i = 1, \ldots, n \). This will show that \( x_1 = \cdots = x_n \).

Now, \( R_i \) and \( \neg rs_i \) imply \( x_i = \text{Valb}(h_i/i_i+1, 1) \). Also \( \text{Compat}(h_1, \ldots, h_n) \) implies
\( h_{i/i_i+1,i+1} = h_{i+1/i_i,i+1} \). Moreover \( R_i+1 \) implies \( x_{i+1} = f_{i+1}(\sigma_{i+1, h_{i+1}}) \), which from the
definition of \( f_{i+1} \) gives, \( x_{i+1} \leq \text{Valb}(h_{i+1/i_i,i+1}, 1) = x_i \). Thus (1) implies \( x_1 = \cdots = x_n \).

Next we shall show that (1) implies
\[
\gcd(x_1, \ldots, x_n) = \gcd(\sigma_1, \ldots, \sigma_n)
\]
which will complete the proof.

To prove the above, we need only prove
\[
\gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n)) = \gcd(\sigma_1, \ldots, \sigma_n).
\]
The proof of this will be by induction on the length of the sequences. If \( h_1 = \cdots = h_n = \varepsilon \), the result follows directly from the definitions of \( f_1, \ldots, f_n \). If not, from the
definitions of \( \text{Compat} \), it follows that there exist \( i, k \) such that
\[
[\text{Elemb}(h_i, 1) = \text{Elemb}(h_{i+1}, 1) = (i, i+1, k)]
\]
or
\[
[\text{Elemb}(h_i, 1) = \text{Elemb}(h_{i-1}, 1) = (i-1, i, k)]
\]

Then from \( R_i \) and \( R_{i+1} \) (or \( R_{i-1} \)) and the definitions of \( f_1, \ldots, f_n \) it easily follows that
\[
\gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n)) = \gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n))
\]
or
\[
\gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n)) = \gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n)).
\]

In either case, by the inductive hypothesis, we have
\[
\gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n)) = \gcd(\sigma_1, \ldots, \sigma_n).
\]
This completes the proof.
5. A bounded buffer

The following process (a modified version of the one in [2]) simulates a bounded buffer (of length 10):

\[ P_2::\{h_2 = \varepsilon\} \]
\[ \texttt{in} := 0; \texttt{out} := 0; \]
\[ *\{R_2\} \]
\[ [\texttt{in} < \texttt{out} + 10; P_1?\texttt{buf (in mod 10)} \rightarrow \texttt{in} := \texttt{in} + 1] \]
\[ \texttt{out} < \texttt{in}; P_3!\texttt{buf (out mod 10)} \rightarrow \texttt{out} := \texttt{out} + 1]\]

\[ \{R'_3\} \]

\( P_1 \) is the producer and \( P_3 \) the consumer process.

\[ R_2 = [h_2 \text{ seq} \{(2, 3), (1, 2)\}] \land [\texttt{out} \leq \texttt{in} \leq \texttt{out} + 10] \land [\texttt{out} \geq 0] \land [\texttt{in} \geq 0] \]
\[ \land [[\|h_2/(2,3)\| = \texttt{out}] \land [[\|h_2/(1,2)\| = \texttt{in}]] \]
\[ \land [\forall i \cdot 0 \leq i < \texttt{out} \Rightarrow [\text{Val}(h_2/(2,3), i + 1) = \text{Val}(h_2/(1,2), i + 1)]] \]
\[ \land [\forall i \cdot \texttt{out} \leq i < \texttt{in} \Rightarrow [\text{Val}(h_2/(1,2), i + 1) = \text{buf}(i \text{ mod } 10)]]] \]

\[ R'_2 = [R_2(Lr(h_2))] \]
\[ \land [[\text{Elemb}(h_2, 1) = (2, \Phi, \tau) \land \text{out} \geq \text{in} \geq \text{out} + 10] \]
\[ \lor [\text{Elemb}(h_2, 1) = (2, \{1\}, \tau) \land \text{out} \geq \text{in} \land \text{in} < \text{out} + 10] \]
\[ \lor [\text{Elemb}(h_2, 1) = (2, \{3\}, \tau) \land \text{in} \geq \text{out} + 10 \land \text{out} \leq \text{in}] \]
\[ \lor [\text{Elemb}(h_2, 1) = (2, \{1, 3\}, \tau) \land \text{out} < \text{in} \land \text{out} < \text{out} + 10] \].

\( R'_2 \) may then be reduced (using \( R_2 \)) to

\[ R'_2 = [R_2(Lr(h_2))] \]
\[ \land [[\text{Elemb}(h_2, 1) = (2, \{1\}, \tau) \land \text{out} = \text{in}] \]
\[ \lor [\text{Elemb}(h_2, 1) = (2, \{3\}, \tau) \land \text{in} = \text{out} + 10] \]
\[ \lor [\text{Elemb}(h_2, 1) = (2, \{1, 3\}, \tau) \land \text{out} < \text{in} < \text{out} + 10] \].

Again, it is easy to verify \( R_2, R'_2, R'_2 \). \( R'_2 \) shows that the numbers sent to \( P_3 \) form an initial subsequence of the numbers received from \( P_1 \). This, in fact, is all we can prove (considering \( P_2 \) in isolation) since if the consumer were to terminate prematurely, the buffer would also do the same, and not all the numbers will reach the consumer.
6. An integer semaphore

The following process simulates an integer semaphore serving 100 processes:

\[ P_0::\{h_0 = e\} \]

\[ \text{Val} := 1; \]

\[ *\{R_0\} \]

\[ [P_1?\text{V()} \rightarrow \text{Val} := \text{Val} + 1] \]

\[ \square \text{Val} > 0; P_1?\text{P()} \rightarrow \text{Val} := \text{Val} - 1 \]

\[ \square P_2?\text{V()} \rightarrow \text{Val} := \text{Val} + 1 \]

\[ \square \text{Val} > 0; P_2?\text{P()} \rightarrow \text{Val} := \text{Val} - 1 \]

\[ \vdots \]

\[ \square P_{100}?\text{V()} \rightarrow \text{Val} := \text{Val} + 1 \]

\[ \square \text{Val} > 0; P_{100}?\text{P()} \rightarrow \text{Val} := \text{Val} - 1 \]

\[ \{R'_0\} \]

\( P_0 \) may input two different kinds of objects: \( P() \) and \( \text{V()} \); the input guard \( P_1?\text{V()} \) can only be matched by an output \( P_0!\text{V()} \) in the process \( P_1 \); similarly the guard \( P_1?\text{P()} \) can only be matched by an output \( P_0!\text{P()} \) in \( P_1 \). This requires some minor changes to be made in the rules, and we do not set these down explicitly.

\[ R_0 = [\text{Val} \geq 0 \land h_0 \text{ seq}\{(i, 0, P()), (j, 0, \text{V()} )\} | i, j \in \{1, \ldots, 100\}] \land \forall k \leq ||h_0|| \cdot [ f_p(\text{Init}(h_0, k)) \leq f_x(\text{Init}(h_0, k)) + 1] \]

where \( \text{Init}(h, k) \) is the initial subsequence of \( h \) containing the first \( k \) elements of \( h \).

\[ f_p(h) = \begin{cases} 0 & \text{if } h = e \text{ then} \\ \text{else if } \text{Val}(h, 1) = P() \text{ then } f_p(\text{Lr}(h)) + 1 & \text{else } f_p(\text{Lr}(h)). \end{cases} \]

Thus \( f_p(h) \) is the number of \( P() \)'s in \( h \). \( f_v(h) \) is similar.

\[ R'_0 = R_0(\text{Lr}(h_0)) \land [\text{Elemb}(h_0, 1) = (0, \{1, \ldots, 100\}, \tau)]. \]

It may seem surprising that \( R'_0 \) does not ensure that if a particular element of \( h_0 \) is, say, \( (i, 0, P()) \) then its next element must necessarily be \( (i, 0, \text{V()} ) \). This in fact cannot be ensured by \( P_0 \) as it stands; it is something that must be guaranteed by the user processes (and \( \text{Compat}(h_0, h_1, \ldots, h_{100}) \)). \( R'_0 \) does guarantee that, for every initial subsequence of \( h_0 \), the number of \( P() \)'s exceeds the number of \( \text{V()} \)'s, at most by 1; and that \( P_0 \) will terminate only when all the user processes have terminated. That completes the discussion of our final example.
7. Concluding remarks

We have considered a large number of CSP programs, and in each case proved, using the axiomatic semantics of Section 2, useful and interesting properties of the programs. The proofs are, in the author's opinion, relatively simple, although in some cases we have omitted some of the tedious details. Perhaps the single most important factor contributing to the simplicity of the proofs is that, using the semantics of Section 2, one can deal with the processes independently, and then take the conjunction of the individual post-conditions as the post-condition for the entire program. This may be contrasted with the system of Apt et al. [1], where one must show that the proofs of the individual processes 'cooperate' before arriving at a post-condition for the entire program.

Moreover, no auxiliary variables are needed in our system, in contrast to the system of [1]. Our communication sequences are not auxiliary variables, since we do not introduce any assignment statements corresponding to them. This also contributes to the simplicity of the proofs, since the introduction of auxiliary variables often requires considerable ingenuity.

Finally, we would like to point out the absence of shared variables in CSP is extremely important; without this feature, proofs of correctness of even simple programs would be very difficult; perhaps, it is also this same feature which makes CSP one of the most elegant languages for parallel programming.

Acknowledgment

The author would like to thank Professor O.J. Dahl for several discussions, during which some of the proofs sketched in this paper took shape. The work reported in this paper was performed while the author was supported by a post-doctorate fellowship of the Royal Norwegian Council for Scientific and Industrial Research (NTNF).

References