Theoretical Computer Science 24 (1983) 131–141
North-Holland

CORRECTNESS PROOFS OF CSP PROGRAMS

N. SOUNDARARAJAN

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 gcd 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_i, h_i may also include

0304-3975/83/$03.00 © Elsevier Science Publishers B.V. (North-Holland)

elements of the kind (i, T, τ), τ 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*

$$\{p_e^x\}x := e\{p\}.$$

A2. *Skip*

$$\{p\}\ \textbf{skip}\ \{p\}.$$

A3. *Output*

$$\{p_{h_i \vdash (i,j,y)}^{h_i}\}P_j!y\{p\}.$$

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

A4. *Input*

$$\{\forall k \cdot p_{k, h_i \vdash (j,i,k)}^{x, h_i}\}P_j?x\{p\}.$$

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_j 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*

$$\frac{\{p \wedge B(g_l)\}C(g_l); S_l\{q\}, l = 1, \ldots, m}{\{p\}[\Box(l = 1, \ldots, m)g_l \to S_l]\{q\}}.$$

There are m guards; each guard g is of the form $b;P_j?x$ or $b;P_j!y$ or b where b is a boolean expression. In each case $B(g_l)$ is just b. $C(g_l)$ is $P_j?x$, $P_j!y$ or **skip** if g_l is respectively $b;P_j?x$, $b;P_j!y$ or b.

R3. *Guarded repetition*

$$\left[p \wedge \left[\bigwedge_{l \in \mathrm{PB}} \neg b_l\right]\right] \Rightarrow q^{h_i \vdash (i, \{j \mid \exists l \leq m.[B(g_l) \wedge C(g_l) \neq \mathbf{skip} \wedge D(g_l) = j]\}, \tau)}_{h_i}$$

$$\frac{\{p \wedge B(g_l)\}C(g_l); S_l\{p\},\ l = 1, \ldots, m}{\{p\} * [\Box(l = 1, \ldots, m) g_l \rightarrow S_l]\{q\}}.$$

PB is the set of indices of those guards that are purely boolean (i.e., of the form b_i, and hence $C(g_l) = \mathbf{skip}$); $D(g_l)$ is j if $C(g_l)$ is $P_j?x$ or $P_j!y$ (and, say, 0 if $C(g_l) = \mathbf{skip}$). Thus the purpose of the first line of R3 is to ensure that the element (i, T, τ) 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*

$$\frac{\{p_i \wedge h_i = \varepsilon\}P_i\{q_i\},\ i = 1, \ldots, n}{\{\bigwedge_{i=1}^{n} p_i\}P_1 /\!/ \cdots /\!/ P_n\{\bigwedge_{i=1}^{n} q_i \wedge \mathrm{Compat}(h_1, \ldots, h_n)\}}.$$

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

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

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

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

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

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

(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 $\mathrm{first}(h_i) = \mathrm{first}(h_j)$ and $\mathrm{first}(h_{i'}) = \mathrm{first}(h_{j'})$, we may use either pair, since it is easy to see that the final value of 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*

$$\frac{\{p_1\}S\{q_1\}, \{p_1\}S\{q_2\}}{\{p_1\}S\{q_1 \wedge q_2\}}.$$

R7. *Disjunction*

$$\frac{\{p_1\}S\{q\}, \{p_2\}S\{q\}}{\{p_1 \vee p_2\}S\{q\}}.$$

3. A set partitioning program

S_0, T_0 are disjoint sets, $S_0 \neq \Phi$. The purpose of the program is to partition the sets in such a way that, when it finishes, we have $[S \cup T = S_0 \cup T_0] \wedge [\max(S) < \min(T)] \wedge [|S| = |S_0|] \wedge [|T| = |T_0|]$, $|R|$ 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;
- Elem(h, i) = ith element of h;
- Elemb(h, i) = Elem$(h, \|h\| - i + 1)$ = ith element from the right end of h;
- Val(h, i) = value of the number being communicated in the ith element of h;
- Valb(h, i) = Val$(h, \|h\| - i + 1)$;
- Dir(h, i) = 'direction' of the ith element of h. Thus if Elem(h, i) is $(k, 1, u)$, Dir$(h, i) = (k, 1)$; and if Elem$(h, i) = (k, T, \tau)$, then Dir$(h, i) = (k, \tau)$;
- Dirb(h, i) = 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

$P_1 :: \{h_1 = \epsilon \wedge S = S_0 \wedge S_0 \cap T_0 = \Phi\}$

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

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

$*\{R_1\}$

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

$\{R_1 \wedge mx \leq x\}$

$$P_2 :: *\{R_2\}$$

$$[P_1?y \to T := T \cup \{y\};\ mn := \min(T);\ P_1!mn;\ T := T - \{mn\}]$$

$$\{R_3\}$$

where

$$R_1 \equiv [T_1(h_1, S) \wedge mx = \max(S) \wedge \|h_1\| \geq 2 \wedge x = \mathrm{Valb}(h_1, 1) \wedge S_0 \cap T_0 = \Phi]$$

and

$$\begin{aligned} T_1(h_1, S) \equiv [[h_1 = \varepsilon \wedge S = S_0] \\ \vee [\mathrm{Even}(\|h_1\|) \wedge v \in S \wedge u > \max(S - \{v\}) \\ \wedge \mathrm{Dirb}(h_1, 1) = (2, 1) \wedge \mathrm{Dirb}(h_1, 2) = (1, 2) \\ \wedge [T_1(\mathrm{Lr}(\mathrm{Lr}(h_1)), (S - \{v\}) \cup \{u\}) \\ \vee T_1(\mathrm{Lr}(\mathrm{Lr}(h_1)), S \cup \{u\})]]] \end{aligned}$$

where v denotes $\mathrm{Valb}(h_1, 1)$, u denotes $\mathrm{Valb}(h_1, 2)$; $\mathrm{Lr}(h)$ 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 $h_1 = \varepsilon \wedge S = S_0$, or if the last two values in h_1 satisfy the clauses $v \in S \wedge u > \max(S - \{v\})$ and either the pair $(\mathrm{Lr}(\mathrm{Lr}(h_1)), (S - \{v\}) \cup \{u\})$ or the pair $(\mathrm{Lr}(\mathrm{Lr}(h_1)), S \cup \{u\})$ 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 $R_1 \wedge mx \leq x$ is the proper post-condition.

Next we define R_2, R_3:

$$\begin{aligned} R_2(h_2, T) \equiv [h_2 = \varepsilon \wedge T = T_0] \\ \vee [\mathrm{Even}(\|h_2\|) \wedge v < \min(T) \wedge u \in T \cup \{v\} \\ \wedge \mathrm{Dirb}(h_2, 1) = (2, 1) \wedge \mathrm{Dirb}(h_1, 2) = (1, 2) \\ \wedge [R_2(\mathrm{Lr}(\mathrm{Lr}(h_2)), (T \cup \{v\}) - \{u\}) \\ \vee R_2(\mathrm{Lr}(\mathrm{Lr}(h_2)), T \cup \{v\})]], \end{aligned}$$

u, v denoting $\mathrm{Valb}(h_2, 2)$ and $\mathrm{Valb}(h_2, 1)$ respectively;

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

Then by the rule for parallel composition, we get

$$\begin{aligned} \{S = S_0 \wedge T = T_0 \wedge S_0 \cap T_0 = \Phi\} P_1 \| P_2 \{R_1(h_1, S) \wedge \max(S) \leq x \\ \wedge R_3(h_2, T) \wedge \mathrm{Compat}(h_1, h_2)\}. \end{aligned}$$

Using R_1, R_3 and Compat, it is easy to see that h_1 and $\mathrm{Lr}(h_2)$ consist of alternative elements of the form $(1, 2, k)$ and $(2, 1, l)$, the corresponding elements of h_1 and

$\mathrm{Lr}(h_2)$ being identical. Thus the post-condition of $P_1 \| P_2$ implies the following:

$$\max(S) \leqslant x = \mathrm{Valb}(h_1, 1) = \mathrm{Valb}(\mathrm{Lr}(h_2), 1) < \min(T).$$

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

To see that the post-condition also implies

$$[S \cup T = S_0 \cup T_0] \wedge [|S| = |S_0|] \wedge [|T| = |T_0|]$$

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

$$R_1(h_1, S) \wedge R_2(h_1, T) \wedge [S_0 \cap T_0 = \Phi].$$

If $h_1 = \varepsilon$, then R_1, R_2 imply $S = S_0$ and $T = T_0$. Thus this case is trivial. If not, let $u = \mathrm{Valb}(h_1, 2)$, $v = \mathrm{Valb}(h_1, 1)$; 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

$$|S| \leqslant |S'|, \quad |T| \leqslant |T'|, \quad |S| \leqslant |S''|, \quad |T| \leqslant |T''|.$$

Also one of the following four predicates is true:

$$R_1(\mathrm{Lr}(\mathrm{Lr}(h_1)), S') \wedge R_2(\mathrm{Lr}(\mathrm{Lr}(h_1)), T'),$$

$$R_1(\mathrm{Lr}(\mathrm{Lr}(h_1)), S') \wedge R_2(\mathrm{Lr}(\mathrm{Lr}(h_1)), T''),$$

$$R_1(\mathrm{Lr}(\mathrm{Lr}(h_1)), S'') \wedge R_2(\mathrm{Lr}(\mathrm{Lr}(h_1)), T'),$$

$$R_1(\mathrm{Lr}(\mathrm{Lr}(h_1)), S'') \wedge R_2(\mathrm{Lr}(\mathrm{Lr}(h_1)), T'').$$

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

$$S \cup T = S_0 \cup T_0 \wedge |S| \leqslant |S_0| \wedge |T| \leqslant |T_0|,$$

which, in conjunction with $S_0 \cap T_0 = \Phi$, 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 $P :: P_1 \| \cdots \| P_n$, where P_i is

$\{h_i = \varepsilon\}$

$x_i := \sigma_i$; $\mathrm{rsl}_i :=$ **true**; $\mathrm{rsr}_i :=$ **true**;

$*\{R_i\}$

$[\mathrm{rsl}_i; P_{i-1}!x_i \to \mathrm{rsl}_i :=$ **false**

$\square\ \mathrm{rsr}_i; P_{i+1}!x_i \to \mathrm{rsr}_i :=$ **false**

$\square\ P_{i-1}?y_i \to [y_i \geqslant x_i \to$ **skip**

$\square\ y_i < x_i \to [y_i | x_i \to x_i := y_i$

$\square\ y_i \nmid x_i \to x_i := x_i \bmod y_i];$

$\mathrm{rsl}_i :=$ **true**; $\mathrm{rsr}_i :=$ **true**]

$\square\ P_{i+1}?y_i \to [y_i \geqslant x_i \to$ **skip**

$\square\ y_i < x_i \to [y_i | x_i \to x_i := y_i$

$\square\ y_i \nmid x_i \to x_i := x_i \bmod y_i];$

$\mathrm{rsl}_i :=$ **true**; $\mathrm{rsr}_i :=$ **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

$$R_i \equiv [h_i \textbf{ seq}\{(i, i \pm 1), (i \pm 1, i)\}] \wedge [x_i = f_i(\sigma_i, h_i)] \wedge T_i(h_i)$$
$$[\neg \mathrm{rsl}_i \Rightarrow x_i = \mathrm{Valb}(h_{i/(i,i-1)}, 1)] \wedge [\neg \mathrm{rsr}_i \Rightarrow x_i = \mathrm{Valb}(h_{i/(i,i+1)}, 1)].$$

The first clause says that h_i is a sequence of elements of the form $(i, i \pm 1, l)$ and $(i \pm 1, i, l')$; $h_{i/(i,i\pm1)}$ 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:

$$T_i(h_i) = [h_i = \varepsilon]$$
$$\vee\, [\mathrm{Dirb}(h_i, 1) = (i \pm 1, i) \wedge T_i(\mathrm{Lr}(h_i))]$$
$$\vee\, [\mathrm{Dirb}(h_i, 1) = (i, i \pm 1)$$
$$\wedge\, \mathrm{Valb}(h_i, 1) = f_i(\sigma_i, \mathrm{Lr}(h_i)) \wedge T_i(\mathrm{Lr}(h_i))]$$

where

$$f_i(z, h_i) = \textbf{if } h_i = \varepsilon \textbf{ then } z$$
$$\textbf{else if } \mathrm{Dirb}(h_i, 1) = (i, i \pm 1) \textbf{ then } f_i(z, \mathrm{Lr}(h_i))$$
$$\textbf{else if } \mathrm{Dirb}(h_i, 1) = (i \pm 1, i)$$
$$\textbf{then } g(f_i(z, \mathrm{Lr}(h_i)), \mathrm{Valb}(h_i, 1))$$

where

$$g(z, m) = \textbf{if } m \geq z \textbf{ then } z \textbf{ else if } m | z \textbf{ then } m \textbf{ else } z \bmod m.$$

Again the reader should have no difficulty in verifying the invariance of R_i. At the time of deadlock, we have

$$\bigwedge_{i=1}^{n} [R_i \wedge \neg \mathrm{rsl}_i \wedge \neg \mathrm{rsr}_i] \wedge \mathrm{Compat}(h_1, \ldots, h_n). \tag{1}$$

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 $\neg \mathrm{rsr}_i$ imply $x_i = \mathrm{Valb}(h_{i/(i,i+1)}, 1)$. Also $\mathrm{Compat}(h_1, \ldots, h_n)$ implies $h_{i/(i,i+1)} = h_{i+1/(i,i+1)}$. 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, $x_{i+1} \leq \mathrm{Valb}(h_{i+1/(i,i+1)}, 1) = x_i$. 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 Compat, it follows that there exist i, k such that

$$[\mathrm{Elemb}(h_i, 1) = \mathrm{Elemb}(h_{i+1}, 1) = (i, i+1, k)]$$

or

$$[\mathrm{Elemb}(h_i, 1) = \mathrm{Elemb}(h_{i-1}, 1) = (i-1, i, k)].$$

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

$$\gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n))$$

$$= \gcd(f_1(\sigma_1, h_1), \ldots, f_{i-1}(\sigma_{i-1}, \mathrm{Lr}(h_{i-1})), f_i(\sigma_i, \mathrm{Lr}(h_i)), \ldots, f_n(\sigma_n, h_n))$$

or

$$\gcd(f_1(\sigma_1, h_1), \ldots, f_n(\sigma_n, h_n))$$

$$= \gcd(f_1(\sigma_1, h_1), \ldots, f_i(\sigma_i, \mathrm{Lr}(h_i)), f_{i+1}(\sigma_{i+1}, \mathrm{Lr}(h_{i+1})), \ldots, f_n(\sigma_n, h_n)).$$

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):

$P_2::\{h_2 = \varepsilon\}$

in := 0; out := 0;

$*\{R_2\}$

[in < out + 10; P_1?buf(in mod 10) → in := in + 1

□ out < in; P_3!buf(out mod 10) → out := out + 1]

$\{R_2'\}$

P_1 is the producer and P_3 the consumer process.

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

$$R_2' \equiv [R_2(\text{Lr}(h_2))$$
$$\wedge [[\text{Elemb}(h_2, 1) = (2, \Phi, \tau) \wedge \text{out} \geq \text{in} \geq \text{out} + 10]$$
$$\vee [\text{Elemb}(h_2, 1) = (2, \{1\}, \tau) \wedge \text{out} \geq \text{in} \wedge \text{in} < \text{out} + 10]$$
$$\vee [\text{Elemb}(h_2, 1) = (2, \{3\}, \tau) \wedge \text{in} \geq \text{out} + 10 \wedge \text{out} \leq \text{in}]$$
$$\vee [\text{Elemb}(h_2, 1) = [2, \{1, 3\}, \tau) \wedge \text{out} < \text{in} < \text{out} + 10]]].$$

R_2' may then be reduced (using R_2) to

$$R_2' \equiv [R_2(\text{Lr}(h_2))$$
$$\wedge [[\text{Elemb}(h_2, 1) = (2, \{1\}, \tau) \wedge \text{out} = \text{in}]$$
$$\vee [\text{Elemb}(h_2, 1) = (2, \{3\}, \tau) \wedge \text{in} = \text{out} + 10]$$
$$\vee [\text{Elemb}(h_2, 1) = (2, \{1, 3\}, \tau) \wedge \text{out} < \text{in} < \text{out} + 10]]].$$

Again, it is easy to verify 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:

$P_0::\{h_0 = \varepsilon\}$

Val := 1;

$*\{R_0\}$

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

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

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

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

$\vdots$

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

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

$\{R_0'\}$

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

$$R_0 \equiv [\text{Val} \geq 0 \wedge h_0\ \mathbf{seq}\{(i, 0, P(\,)), (j, 0, V(\,)) \mid i, j \in \{1, \ldots, 100\}\}$$

$$\wedge \forall k \leq \|h_0\| \cdot [f_p(\text{Init}(h_0, k)) \leq f_v(\text{Init}(h_0, k)) + 1]]$$

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

$$f_p(h) = \textbf{if } h = \varepsilon \textbf{ then } 0$$

$$\textbf{else if } \text{Val}(h, 1) = P(\,) \textbf{ then } f_p(\text{Lr}(h)) + 1$$

$$\textbf{else } f_p(\text{Lr}(h)).$$

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

$$R_0' \equiv R_0(\text{Lr}(h_0)) \wedge [\text{Elemb}(h_0, 1) = (0, \{1, \ldots, 100\}, \tau)].$$

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 $(i, 0, V(\,))$. 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 $V(\,)$'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

[1] K.R. Apt, N. Francez and W.P. de Roever. A proof system for communicating sequential processes, *ACM TOPLAS* **2** (1980) 359–385.

[2] C.A.R. Hoare, Communicating sequential processes, *Comm. ACM* **21** (1978) 666–677.

[3] N. Soundararajan, Axiomatic semantics for communicating sequential processes, Research Report, Institute for Informatics, University of Oslo.