Theoretical Computer Science 33 (1984) 279-304
North-Holland

DENOTATIONAL SEMANTICS OF CSP

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

Communicated by M. Nivat
Received September 1983
Revised May 1984

Abstract. In this paper we propose a new denotational semantics for CSP. The domains used in the semantics are very simple, compared to those used in other approaches to the semantics of CSP. Moreover, our denotations are more abstract than those of the other approaches.

1. Introduction

Francez et al. [3] have proposed a denotational semantics for CSP. In this paper we propose an alternative semantics. The main advantage of our approach is the simplicity of the domains and of the denotations.

Consider a CSP program $[P_1 \| \ldots \| P_n]$. In order to define the semantics of this program, we need to define the semantics of the individual processes $P_1, \ldots, P_n$, and specify how the semantics of the individual processes may be combined to obtain the semantics of the entire program. Consider the ith process P_i; the state of P_i at any time consists of two components: s_i the 'local' state of P_i consisting of the (local) variables of P_i (recall that there are no shared variables in CSP); and h_i, the sequence of all communications that P_i has so far participated in. Thus the semantics of the statements that may appear in P_i will be functions of the kind $f: S_i \times H_i \to S_i \times H_i$, S_i being the set of possible local states of P_i, and H_i the set of its possible communication sequences. Nondeterminism will, however, require us to modify the range of the functions to allow for several possible results, and hence the functions will, in fact, be of the kind $f: S_i \times H_i \to P(S_i \times H_i)$, $P(S)$ being the powerset of S. (Further considerations will require us to restrict the range of f somewhat, so that it will not be the entire powerset of $S_i \times H_i$. The result will be somewhat like the 'powerdomains' of Plotkin [5], except that Plotkin constructs much more general domains including 'recursive powerdomains'. Since we do not need such complex domains, we shall construct our domains from scratch rather than using Plotkin's powerdomain constructor.)

Let us consider a particular statement that may appear in P_i: the output command "$P_j!k$", k being a constant integer (for simplicity, **integer** will be the only type in the language; and there will be no declarations of variables). Then, the denotation of

"$P_j!k$" will be the function

$$f(\langle s_i, h_i\rangle) = \{\langle s_i, h_i\,\hat{}\,(i, j, k)\rangle\}$$

where "$\hat{}$" denotes concatenation of the element (i, j, k) to the right end of h_i. Thus the denotation of $P_j!k$ is a function that concatenates the element (i, j, k) to the right end of h_i to indicate that (when this statement finishes) P_i has participated in a new communication in which the number k was sent by P_i to P_j.

In order to allow for composition of functions (which will, of course, correspond to sequential composition of statements), the domain of the functions must also be $P(S_i \times H_i)$; the value of f over this domain will (usually) be defined in the obvious 'distributive' fashion:

$$f(X) = \bigcup_{\langle s_i, h_i\rangle \in X} f(\{\langle s_i, h_i\rangle\})$$

for any $X \in P(S_i \times H_i)$. Note the braces around $\langle s_i, h_i\rangle$; the reason is that we are taking the domain of f to be $P(S_i \times H_i)$ rather than $S_i \times H_i$.

Consider another example: the input command "$P_j?u$", u being a variable of P_i. Then, $f(\{\langle s_i, h_i\rangle\}) = \{\langle s_i[u \leftarrow k], h_i\,\hat{}\,(j, i, k)\rangle \mid k \in N\}$, where N is the set of all integers. Thus the effect of this statement is to replace the value of u by (some) k, and to cancatenate to h_i the element (j, i, k). Note that when considering P_i in isolation, we have to allow for all possible values for k, since we have no way of knowing, in P_i, what number P_j will actually send to P_i. That will be known only if we consider all the processes; correspondingly, in the semantics, the value of k will be fixed only when we combine the semantics of the individual processes to obtain the semantics of the entire program. In fact, the operation that we shall define for combining the semantics of the individual processes will do little more than to ensure that the numbers received by P_i from P_j (as recorded in P_i's semantics) are identical to the numbers sent by P_j to P_i (as recorded in P_j's semantics).

The paper is organized as follows: in Section 2 we specify the domains to be used in the semantics, and consider some properties of functions (and functionals) on these domains. In Section 3 we define the semantics of the individual processes, and in Section 4 we specify the operation that will combine the individual semantics to obtain the semantics of the entire program. The final section compares our approach with other approaches.

2. The domains and functions

As explained in the introduction, S_i will be the domain of all possible local states of P_i; S_i will be a 'flat' domain, with a bottom element "$\perp$", the partial order on S_i being

$$s_i \sqsubseteq s_i' \text{ iff } s_i = s_i' \text{ or } s_i = \perp.$$

H_i, the domain of communication sequences of P_i will be rather more complex. Note that we must allow not just finite sequences but also infinite ones, since P_i may conceivably communicate forever with one or more of its partners (the so-called 'infinite chattering'). Thus H_i is the set of all finite and infinite sequences of individual communications that P_i may participate in; thus,

$$H_i = C_i^* \cup C_i^\infty,$$

C_i being the set of individual communications that P_i may participate in, C_i^* the set of all finite sequences of elements of C_i, and C_i^∞ the set of all infinite sequences of elements of C_i. We shall call C_i the set of all possible communication elements of P_i.

Next, we need to specify C_i; we have already seen some of the elements of C_i: (i, j, k), $k \in N$, corresponding to an output statement in P_i that sends the number k to P_j; (j, i, k), $k \in N$, corresponding to an input statement in P_i that receives the number k from P_j. There are three other kinds of communications that P_i may participate in corresponding respectively to (a) output guards, (b) input guards, and (c) distributed termination of loops in P_i. First, consider an output guard $b; P_j!k$, b being a boolean expression (in the local variables of P_i), k being a constant. If b evaluates to **true** in the current (local) state of P_i, and this guard is chosen, then P_i will send the number k to P_j in executing this guard. This communication should, however, not be represented by the element (i, j, k) since that is likely to cause problems when the semantics of the individual processes are combined to obtain the semantics of the entire program.

What we need to do is record, in the communication element corresponding to to the output guard, not only the value k communicated by P_i, and the process P_j to which it was communicated, but also what other options were available to P_i at this point. Thus this communication element will be of the form (i, j, k, T), T being the set of 'other options' that P_i had at this point:

$$T \subseteq \{(j', i) \mid 1 \leq j' \leq n, j' \neq i\} \cup \{(i, j') \mid 1 \leq j' \leq n, j' \neq i, j' \neq j\}$$
$$\cup \{(i, T') \mid T' \subseteq \{1, \ldots, i-1, i+1, \ldots, n\}\} \cup \{\iota\}.$$

An element (j', i) in T indicates that P_i could (instead of outputting to P_j) have input from $P_{j'}$; such an element would be included in T if one (or more) of the other guards were of the form $b'; P_{j'}?x$, and the boolean b' had the value **true** in the current state of P_i. Similarly, an element (i, j') in T indicates that P_i could have output to $P_{j'}$ instead of to P_j, since it had a guard $b'; P_{j'}!e$ with b' evaluating to **true**. The element "ι" in T indicates that P_i could have continued 'locally', since it had a purely boolean guard evaluating to **true**. (Note that we allow I/O guards and purely boolean guards to be freely mixed.) Finally an element (i, T') in T indicates that the output guard we are considering occurred in a loop in P_i, and this loop could have terminated at this point (rather than outputting to P_j, and going onto another iteration) if every process whose index appears in T' had already terminated. Note that if "ι" is an element of T, then an element of the kind (i, T') cannot simultaneously be an element of T, since $\iota \in T$ implies that there is a purely boolean guard that evaluates to **true**, and hence the loop cannot possibly have terminated at this point, irrespective of which other processes had already terminated.

Note also that only one element of the kind (i, T') can belong to the T of (i, j, k, T), since T' is the set of indices of *all* processes that must terminate for P_i's loop to terminate. Also j must necessarily be an element of such a T', since P_i is clearly willing to output to P_j. Note finally that "ι" is just a constant symbol used to indicate the 'local option' that is available to P_i.

Next consider an input guard $b\,; P_j?x$. Clearly the communication element corresponding to this will be quite similar to the one for output guards, and will be of the form (j, i, k, T), T being the set of 'other options' open to P_i:

$$T \subseteq \{(i, j') \mid j' \neq i\} \cup \{(j', i) \mid j' \neq i, j' \neq j\}$$
$$\cup \{(i, T') \mid T' \subseteq \{1, \ldots, i-1, i+1, \ldots, n\}\} \cup \{\iota\}.$$

Finally, consider the communication element corresponding to the termination of a loop on account of the distributed termination convention. Such an element will be of the form (i, T, τ, T') where T is the set of indices of all processes whose termination caused the loop in P_i to terminate, "τ" (like "ι") is a constant symbol to indicate the nature of this element, and T' is the set of other options open to P_i at this point:

$$T' \subseteq \{(i, j) \mid j \neq i\} \cup \{(j, i) \mid j \neq i\}.$$

Thus,

$$C_i = \{(i, j, k) \mid j \neq i, k \in N\} \cup \{(j, i, k) \mid j \neq i, k \in N\}$$
$$\cup \{(i, j, k, T) \mid j \neq i, T \subseteq O_j, k \in N\}$$
$$\cup \{(j, i, k, T) \mid j \neq i, T \subseteq O'_j, k \in N\}$$
$$\cup \{(i, T, \tau, T') \mid T \subseteq \{1, \ldots, i-1, i+1, \ldots, n\}, T' \subseteq O''\},$$

where

$$O_j = \{(i, j') \mid j' \neq i, j' \neq j\} \cup \{(j', i) \mid j' \neq i\} \cup \{\iota\}$$
$$\cup \{(i, T'') \mid T'' \subseteq \{1, \ldots, i-1, i+1, \ldots, n\}\}.$$

O'_j is defined similarly.

$$O'' = \{(i, j) \mid j \neq i\} \cup \{(j, i) \mid j \neq i\}.$$

We can simplify C_i somewhat by replacing O_j, O'_j and O'' by

$$O = \{(i, j') \mid j' \neq i\} \cup \{(j', i) \mid j' \neq i\} \cup \{\iota\}$$
$$\cup \{(i, T'') \mid T'' \subseteq \{1, \ldots, i-1, i+1, \ldots, n\}\}.$$

This will allow certain elements that operationally speaking are meaningless (for instance, "ι" being an element of T' in an element (i, T, τ, T') is meaningless, since the loop cannot possibly terminate if P_i has a 'local' option). However, the function definitions will make sure that such impossible elements are not introduced into the h_i.

We can simplify C_i further by writing (i, j, k) as (i, j, k, Φ), Φ being the empty set (of 'other options'), and (j, i, k) as (j, i, k, Φ). Thus,

$$
\begin{aligned}
C_i = &\{(i, j, k, T) \mid j \neq i, k \in N, T \subseteq O\} \\
&\cup \{(j, i, k, T) \mid j \neq i, k \in N, T \subseteq O\} \\
&\cup \{(i, T, \tau, T') \mid T \subseteq \{1, \ldots, i-1, i+1, \ldots, n\}, T' \subseteq O\}.
\end{aligned}
$$

Consider an example: a loop in the process P_1 of a program $[P_1 \| P_2 \| P_3 \| P_4]$:

$$
\begin{aligned}
&*[b_1 ; P_2!100 \rightarrow \cdots \\
&\square\ b_2 ; P_3?x \rightarrow \cdots \\
&\square\ b_3 ; P_4!200 \rightarrow \cdots \\
&\square\ b_4 ; P_2?y \rightarrow \cdots].
\end{aligned}
$$

Suppose during a particular iteration, all the booleans evaluate to **true**; and that we are considering the case when the first guard is chosen. The corresponding communication element concatenated to h_1, the communication sequence of P_1, would be $(1, 2, 100, T)$ where

$$T = \{(3, 1), (1, 4), (2, 1), (1, \{2, 3, 4\})\}$$

to say that P_1 sent the number 100 to P_2, and the other options available to P_1 at this point were (a) to receive a number from P_3, (b) to send a number to P_4, (c) to receive a number from P_2, and (d) to terminate its loop, this option, however, requiring that P_2, P_3 and P_4 had already terminated.

H_i, as already remarked, is the set of all finite and infinite sequences of elements of C_i:

$$H_i = C_i^* \cup C_i^\infty.$$

The order on H_i is the initial subsequence order:

$$h_i \sqsubseteq h_i' \text{ iff } h_i \text{ is an initial subsequence of } h_i'.$$

The empty sequence ε is the least element of H_i.

Next, we consider the domain $S_i \times H_i$. In fact, it is rather incorrect to use the cartesian product notation "$\times$" for this domain since the order we shall use on this domain will be

$$\langle s_i, h_i \rangle \sqsubseteq \langle s_i', h_i' \rangle \text{ iff } [s_i = s_i' \text{ and } h_i = h_i'] \text{ or } [s_i = \perp \text{ and } h_i \sqsubseteq h_i'],$$

which is not the usual order on cartesian product domains. Moreover, the domain $S_i \times H_i$ will not include all elements of the kind $\langle s_i, h_i \rangle$; instead,

$$S_i \times H_i = \{\langle s_i, h_i \rangle \mid s_i \in S_i, h_i \in C_i^*\} \cup \{\langle \perp, h_i \rangle \mid h_i \in C_i^\infty\}.$$

Thus a non-bottom element of S_i cannot be paired with an infinite communication sequence. The reason for this is that the 'current' state of P_i can have an infinitely long communication sequence component only as a result of a nonterminating loop in P_i, and the corresponding 'local' state component of P_i must necessarily be $\perp$. ($\langle \perp, h_i \rangle$, where h_i is a finite sequence is, of course, a perfectly reasonable state of P_i.)

Despite the above remarks, in this paper we shall use the notation $S_i \times H_i$ for the above domain, since it is 'almost' the cartesian product of S_i and H_i. In summary,

$$S_i \times H_i = \{\langle s_i, h_i \rangle \mid s_i \in S_i, h_i \in C_i^*\} \cup \{\langle \perp, h_i \rangle \mid h_i \in C_i^\infty\}$$

and

$$\langle s_i, h_i \rangle \sqsubseteq \langle s_i', h_i' \rangle \text{ iff } [s_i = s_i', h_i = h_i'] \text{ or } [s_i = \perp, h_i \sqsubseteq h_i'].$$

Definition. An infinite sequence $\{x_1, x_2, \ldots\}$ of elements of $S_i \times H_i$ is a *chain* if $x_j \sqsubseteq x_{j+1}$ for all j.

Theorem 2.1. *Every chain $\{x_1, x_2, \ldots\}$ of elements of $S_i \times H_i$ has a unique least upper bound.*

Proof. Either there exists a k such that $x_k = x_{k+1} = \cdots$, in which case x_k is $\text{lub}\{x_1, x_2, \ldots\}$; or, for all k, $x_k = \langle \perp, h_k \rangle$, and there exists a k' such that $k' > k$ and $x_k \neq x_k'$; in this case, $\text{lub}\{x_1, x_2, \ldots\} = \langle \perp, \text{lt}\{h_1, h_2, \ldots\} \rangle$, the second component of the lub being an infinite sequence.

Next consider the domain $P(S_i \times H_i)$. The order on this domain will be the usual Egli–Milner order:

$$X \sqsubseteq X' \text{ iff } [\forall x \in X. \exists x' \in X'. x \sqsubseteq x' \wedge \forall x' \in X'. \exists x \in X. x \sqsubseteq x'].$$

We shall impose the following restrictions on the elements of $P(S_i \times H_i)$:

(a) $X \in P(S_i \times H_i)$ implies X is *convex*, i.e.,

$$[x \sqsubseteq y \sqsubseteq z \wedge x \in X \wedge z \in X] \Rightarrow y \in X.$$

This restriction results in slightly unnatural denotations for some *individual processes*; however, it simplifies the theory considerably (for instance, lub's of chains of elements of $P(S_i \times H_i)$ become unique). Moreover, the denotations of *complete CSP* programs are unaffected by the imposition of the convexity requirement.

(b) If $X \in P(S_i \times H_i)$, X is an infinite set, and there exist $x_1, x_2, \ldots, x_1', x_2', \ldots$, such that, for all j, $x_j \sqsubset x_{j+1} \wedge x_j \sqsubseteq x_j' \wedge x_j' \in X$, then $\text{lub}\{x_1, x_2, \ldots\} \in X$.

Essentially this requirement states that if (a loop in) a process can communicate an arbitrarily large number of times, then it can also communicate forever.

A subset of $S_i \times H_i$ that satisfies restriction (b) will be called *complete*. Thus every element of $P(S_i \times H_i)$ is convex and complete. □

Lemma 2.2. *If X and Y are complete (subsets of $S_i \times H_i$), then so is $X \cup Y$.*

Proof. Suppose $z = \text{lub}\{z_1, z_2, \ldots\}$, $z_j \sqsubset z_{j+1}$ for all j, and there exist $z'_1, z'_2, \ldots$ such that

$$z_j \sqsubseteq z'_j \wedge z'_j \in X \cup Y \quad \text{for all } j;$$

we have to show that $z \in X \cup Y$. If all but finitely many z'_j belong to X (or Y), then by the closure of X (or Y) z will belong to X (or Y) and hence to $X \cup Y$. If not, we can replace any z'_k that does not belong to X by a z'_j $(j > k)$ that does belong to X, and hence z will belong to X (by completeness of X), hence to $X \cup Y$. Thus $X \cup Y$ is complete. □

Definition. The *convex closure operator* is defined as follows: For any $X \subseteq S_i \times H_i$,

$$C_1[X] = \{y \mid \exists x, z \in X. x \sqsubseteq y \sqsubseteq z\}.$$

It is easy to see that C_1 is a closure operator, i.e., $C_1[C_1[X]] = C_1[X]$.

Lemma 2.3. *If $Y \subseteq S_i \times H_i$ and Y is complete, then so is $C_1[Y]$.*

Proof. The proof is straightforward and is left to the reader. □

Thus if Y is a complete subset of $S_i \times H_i$, then $C_1[Y]$ is an element of $P(S_i \times H_i)$.

Definition. The *completeness closure operator* C_2 is defined as follows: For any $X \subseteq S_i \times H_i$,

$$C_2[X] = X \cup \{x \mid \exists x_1, x_2, \ldots, x'_1, x'_2, \ldots$$
$$.[[\forall j. x_j \sqsubset x_{j+1} \wedge x_j \sqsubseteq x'_j \wedge x'_j \in X] \wedge x = \text{lub}\{x_1, x_2, \ldots\}]\}.$$

Note. Similar operators have also been defined by Boasson and Nivat [1] but in a different context.

Lemma 2.4. $[X$ *is a convex subset of* $S_i \times H_i] \Rightarrow [C_2[X] \in P(S_i \times H_i)]$.

Proof. Suppose

$$x = \text{lub}\{x_1, x_2, \ldots\}, \quad x_1 \sqsubset x_2 \sqsubset x_3 \sqsubset \cdots$$

and there exist $x'_1, x'_2, \ldots$ such that $[x_j \sqsubseteq x'_j \wedge x'_j \in C_2[X]]$ for all j; then we need to show $x \in C_2[X]$.

If $x'_j \in X$ for all j, we are done. If not, we shall show that we can find $x''_1, x''_2, \ldots \in X$ which will serve the same purpose as $x'_1, x'_2, \ldots$ (i.e., $x_j \sqsubseteq x''_j$). Suppose, in particular, $x'_j \in C_2[X] - X$, "$-$" being the set subtraction operator. Then, by definition of $C_2[X]$, $x'_j = \text{lub}\{y_1, y_2, \ldots\}$, and there exist $y'_1, y'_2, \ldots$ such that $\forall k.[y_k \sqsubset y_{k+1} \wedge y_k \sqsubseteq y'_k \wedge y'_k \in X]$. Hence x'_j is infinite (i.e., is of the form $\langle \bot, h' \rangle$ where h' is an infinite sequence). Also x_j is finite (i.e., its sequence component is finite) and is of the form $\langle \bot, h_j \rangle$ since $x_j \sqsubset x_{j+1}$. This, along with the fact that $x_j \sqsubseteq x'_j$ implies that there exists an l such that $x_j \sqsubseteq y_l$, and so we can take x''_j to be y'_l (since $y_l \sqsubseteq y'_l$ and $x_j \sqsubseteq y_l$).

This process can be repeated for any x'_j that does not belong to X. Hence

$$x = \text{lub}\{x_1, x_2, \ldots\}, \qquad x_j \sqsubset x_{j+1} \wedge x_j \sqsubseteq x''_j \wedge x''_j \in X \quad \text{for all } j;$$

and so, by the definition of $C_2[X]$, $x \in C_2[X]$. Thus $C_2[X]$ is complete. (The proof also shows that $C_2[C_2[X]] = C_2[X]$.)

We can easily show that if X is convex, so is $C_2[X]$. Thus if X is a convex subset of $S_i \times H_i$, then $C_2[X]$ is an element of $P(S_i \times H_i)$. □

Notation. We shall often write $C_2[X]$ as $X \cup X^*$ where X^* denotes the set

$$\{x \mid \exists x_1, x_2, \ldots, x'_1, x'_2, \ldots .[[\forall j. x_j \sqsubset x_{j+1} \wedge x'_j \in X \wedge x_j \sqsubseteq x'_j] \wedge x = \text{lub}\{x_1, x_2, \ldots\}]\}.$$

The reason for this notation is the following lemma.

Lemma 2.5. *$C_2[Y] \sqsubseteq C_2[Z]$ if the following condition is satisfied:*

$$[\forall y \in Y. \exists z \in C_2[Z]. y \sqsubseteq z] \wedge [\forall z \in Z. \exists y \in C_2[Y]. y \sqsubseteq z].$$

Proof. Suppose the condition is satisfied. Suppose also that $y \in Y^*$. Then $y = \text{lub}\{y_1, y_2, \ldots\}$, and there exist $y'_1, y'_2, \ldots,$ such that $\forall j.[y_j \sqsubset y_{j+1} \wedge y_j \sqsubseteq y'_j \wedge y'_j \in Y]$. Hence there exist $z'_1, z'_2, \ldots$ such that $\forall j. y'_j \sqsubseteq z'_j \wedge z'_j \in Z$. Hence, by completeness of $C_2[Z]$, $y \in C_2[Z]$. Similarly, we can show that if $z \in Z^*$, there exists a $y \in C_2[Y]$ such that $y \sqsubseteq z$. □

Lemma 2.6. *C_1 and C_2 commute, i.e., $C_1[C_2[X]] = C_2[C_1[X]]$ for all $X \subseteq S_i \times H_i$.*

Proof. Straightforward. □

The following lemma proves that C_2 distributes over (finite) union.

Lemma 2.7. *$C_2[X_1 \cup X_2] = C_2[X_1] \cup C_2[X_2]$, $\forall X_1, X_2 \subseteq S_i \times H_i$.*

Proof. Straightforward. □

Definition. An infinite sequence $\{X_1, X_2, \ldots\}$ of elements of $P(S_i \times H_i)$ is a *chain* if $X_j \sqsubseteq X_{j+1}$ for all j.

Theorem 2.8. *Every chain $\{X_1, X_2, \ldots\}$ of elements of $P(S_i \times H_i)$ has a unique lub*

Proof. Define $X = C_2[Y]$, where

$$Y = \{x \mid \exists x_1, x_2, \ldots .[[\forall j.x_j \sqsubseteq x_{j+1} \wedge x_j \in X_j] \wedge x = \text{lub}\{x_1, x_2, \ldots\}]\}.$$

We need to show that X has the following properties:

(a) $X \in P(S_i \times H_i)$, i.e., X is convex and complete.

(b) $X_j \sqsubseteq X$ for all j.

(c) If $Z \in P(S_i \times H_i)$ and $X_j \sqsubseteq Z$ for all j, then $X \sqsubseteq Z$.

Ad (a): It is easy to show that Y is convex; hence by Lemma 2.2, X is convex and complete.

Ad (b): We have to show, for all j,

$$x_j \in X_j \Rightarrow \exists x \in X.x_j \sqsubseteq x$$

$$x \in Y \Rightarrow \exists x_j \in X_j.x_j \sqsubseteq x.$$

Both results are trivial, and we omit the details. (We have used Lemma 2.5 to write $x \in Y$ rather than $x \in X$ in the left-hand side of the second implication above.)

Ad (c): First we need to show

$$\forall x \in Y.\exists z \in Z.z \sqsubseteq z.$$

If there exists a j such that $\forall k \geq j.x \in X_k$, the result is immediate. If not,

$$x = \text{lub}\{x_1, x_2, \ldots\}.$$

From the sequence $\{x_1, x_2, \ldots\}$, (repeatedly) remove all elements x_j that satisfy $x_j = x_{j-1}$. Then we can find $x'_1, x'_2, \ldots \in Z$ such that $\forall j.x_j \sqsubseteq x'_j$. Hence $x \in Z$, since Z is complete.

Next we need to show

$$\forall z \in Z.\exists x \in X.x \sqsubseteq z.$$

Since $z \in Z$, and $X_j \sqsubseteq Z$, there exist (for all j) $z_j \in X_j$ such that $z_j \sqsubseteq z$. Replace each z_j by the least (under the order on $S_i \times H_i$) $x_j \in X_j$ that satisfies $x_j \sqsubseteq z$. Then we can easily show that $\{x_1, x_2, \ldots\}$ is a chain; its lub x will belong to X and will satisfy $x \sqsubseteq z$.

Thus X is indeed the l.u.b. of $\{X_1, X_2, \ldots\}$. □

Note. It is easy to show that $\text{lub}\{X, X, \ldots\} = X$ for all $X \in P(S_i \times H_i)$.

Lemma 2.9. *If $\{X_1, X_2, \ldots\}$ is a chain, and there exists a chain $\{x_{j_1}, x_{j_2}, \ldots\}$ of elements of $S_i \times H_i$, $x_{j_k} \in X_{j_k}$ for all k, and $j_k < j_{k+1}$ for all k, then* $\text{lub}\{x_{j_1}, x_{j_2}, \ldots\} \in \text{lub}\{X_1, X_2, \ldots\}$.

Proof. $X_1 \sqsubseteq X_2 \sqsubseteq \cdots \sqsubseteq X_{j_1}$, hence we can find $x_1, \ldots, x_{j_1-1}$ such that $x_l \in X_l$ for all $l \leq j_1 - 1$, and $x_1 \sqsubseteq x_2 \sqsubseteq \cdots \sqsubseteq x_{j_1}$. Now, $X_{j_1} \sqsubseteq X_{j_1+1}$; hence, there exist $x_{j_1+1} \in X_{j_1+1}$ such that $x_{j_1} \sqsubseteq x_{j_1+1}$. Take the least $x_{j_1+1} \in X_{j_1+1}$ that satisfies $x_{j_1} \sqsubseteq x_{j_1+1}$. This x_{j_1+1} will necessarily satisfy $x_{j_1+1} \sqsubseteq x_{j_2}$ since $X_{j_1+1} \sqsubseteq X_{j_2}$. This process can be repeated so that we get a chain $\{x_1, x_2, \ldots\}$ such that $x_l \in X_l$, and x_{j_k} is the original x_{j_k} we started with, for all k. Clearly

$$\text{lub}\{x_{j_1}, x_{j_2}, \ldots\} = \text{lub}\{x_1, x_2, \ldots\} \in \text{lub}\{X_1, X_2, \ldots\}. \qquad \square$$

Lemma 2.10. *If $X_1, X_2, \ldots, Y_1, Y_2, \ldots$ are elements of $P(S_i \times H_i)$ such that $X_j \sqsubseteq X_{j+1}$, $Y_j \sqsubseteq Y_{j+1}$ for all j, then*

$$\text{lub}\{C_1[X_1 \cup Y_1], C_1[X_2 \cup Y_2], \ldots\} = C_1[\text{lub}\{X_1, X_2, \ldots\} \cup \text{lub}\{Y_1, Y_2, \ldots\}].$$

Proof. Note first that

$$C_1[X_j \cup Y_j] \sqsubseteq C_1[X_{j+1} \cup Y_{j+1}] \quad \text{for all } j,$$

and hence $\text{lub}\{C_1[X_1 \cup Y_1], C_1[X_2 \cup Y_2], \ldots\}$ does exist and is convex. It is easy to see that

$$C_1[\text{lub}\{X_1, X_2, \ldots\} \cup \text{lub}\{Y_1, Y_2, \ldots\}] \subseteq \text{lub}\{C_1[X_1 \cup Y_1], C_1[X_2 \cup Y_2], \ldots\}.$$

Next suppose $z = \text{lub}\{z_1, z_2, \ldots\}$, $z_j \sqsubseteq z_{j+1}$, $z_j \in X_j \cup Y_j$ for all j. We need to show that $z \in C_1[\text{lub}\{X_1, X_2, \ldots\} \cup \text{lub}\{Y_1, Y_2, \ldots\}]$. If $z_j \in X_j$ (or Y_j) for all but finitely many j, the result is immediate. If not, it follows from Lemma 2.9 that $z \in \text{lub}\{X_1, X_2, \ldots\}$. $\square$

That completes the discussion of the domains. Next consider the functions. These will have the functionality f: $P(S_i \times H_i) \to P(S_i \times H_i)$.

Definition. f is a *monotonic function* if $X \sqsubseteq Y \Rightarrow f(X) \sqsubseteq f(Y)$.

All our functions will be monotonic. In fact, all our functions will also satisfy some other conditions.

Definition. A monotonic function f is a *semantic function* if it satisfies the following conditions:

(a) $X = \Phi \;\Rightarrow\; f(X) = \Phi$,

(b) $f(C_1[X_1 \cup X_2]) = C_1[f(X_1) \cup f(X_2)] \quad \forall X_1, X_2 \in P(S_i \times H_i)$,

(c) $\forall x \in X. \exists y \in f(X). x^2 \sqsubseteq y^2$ and

$\forall y \in f(X). \exists x \in X. x^2 \sqsubseteq y^2$ for all $X \in P(S_i \times H_i)$,

where x^2 is the sequence component of x, i.e., x^2 is h if x is $\langle s, h\rangle$. (Similarly we shall use x^1 to denote the 'state' component s of x.)

Note. Condition (b) implies that $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$; and (c) implies (a).

All our functions will be semantic functions. Most (but not all) of our functions will, in fact, be 'simple' semantic functions.

Definition. A semantic function f is a *simple* semantic function if it satisfies the following additional conditions:

(d) $f(\{\langle \bot, h\rangle\}) = \{\langle \bot, h\rangle\}$,

(e) $f(\{\langle a_i, h\rangle\}) = \{\langle a_i, h\rangle\}$,

where a_i is the 'abort' state into which P_i goes if it attempts to execute a selection statement that has no guard whose boolean portion evaluates to **true**,

(f) $f(X) = \bigcup_{x \in X} f(\{x\})$ for all $X \in P(S_i \times H_i)$.

Definition. If f, g are functions on $P(S_i \times H_i)$, then the functions $f \circ g, f \cup g$ are defined as follows:

$$f \circ g(X) = f(g(X)), \qquad f \cup g(X) = C_1[f(X) \cup g(X)].$$

(1) We can easily prove that $f \circ g(X), f \cup g(X) \in P(S_i \times H_i)$ for all $X \in P(S_i \times H_i)$, and

(2) $f \circ g, f \cup g$ are monotonic if f and g are.

We can also prove the following result.

Theorem 2.11. *$f \circ g, f \cup g$ are semantic functions if f and g are.*

Proof. The only slightly nontrivial proofs are those of the facts that

$$f \cup g(C_1[X_1 \cup X_2]) = C_1[f \cup g(X_1) \cup f \cup g(X_2)]$$

and

$$[\forall x \in X. \exists y \in f \cup g(X). x^2 \sqsubseteq y^2] \wedge [\forall y \in f \cup g(X). \exists x \in X. x^2 \sqsubseteq y^2].$$

Consider the first result:

$$\begin{aligned} f \cup g(C_1[X_1 \cup X_2]) &= C_1[f(C_1[X_1 \cup X_2]) \cup g(C_1[X_1 \cup X_2])] \\ &= C_1[C_1[f(X_1) \cup f(X_2)] \cup C_1[g(X_1) \cup g(X_2)]] \\ &= C_1[f(X_1) \cup f(X_2) \cup g(X_1) \cup g(X_2)] \\ &= C_1[C_1[f(X_1) \cup g(X_1)] \cup C_1[f(X_2) \cup g(X_2)]] \\ &= C_1[f \cup g(X_1) \cup f \cup g(X_2)]. \end{aligned}$$

Next suppose $x \in X$. Then there exists a $y \in g(X)$ such that $x^2 \sqsubseteq y^2$. By definition of $f \cup g$, $y \in g(X) \Rightarrow y \in f \cup g(X)$. Hence $\forall x \in X . \exists y \in f \cup g(X) . x^2 \sqsubseteq y^2$. Similarly we can show $\forall y \in f \cup g(X) . \exists x \in X . x^2 \sqsubseteq y^2$. □

Definition. If f, g are monotonic functions, then $f \sqsubseteq g$ if

$$f(X) \sqsubseteq g(X) \quad \forall X \in P(S_i \times H_i).$$

Definition. An infinite sequence $\{f_1, f_2, \ldots\}$ of monotonic functions is a *chain* if $f_j \sqsubseteq f_{j+1}$ $\forall j$.

Theorem 2.12. *Every chain $\{f_1, f_2, \ldots\}$ of semantic functions has a unique lub f which is also a semantic function.*

Proof. Define $f(X) = \text{lub}\{f_1(X), f_2(X), \ldots\}$ $\forall X \in P(S_i \times H_i)$. It is easy to show that f is a monotonic function and that f is the least upper bound of $\{f_1, f_2, \ldots\}$. We also need to verify that f is a semantic function:

It is clear that $f(\Phi) = \Phi$.

Next,

$$\begin{aligned} f(C_1[X \cup Y]) &= \text{lub}\{f_1(C_1[X \cup Y]), f_2(C_1[X \cup Y]), \ldots\} \\ &= \text{lub}\{C_1[f_1(X) \cup f_1(Y)], C_1[f_2(X) \cup f_2(Y)], \ldots\} \\ &= C_1[\text{lub}\{f_1(X), f_2(X), \ldots\} \\ &\qquad \cup \text{lub}\{f_1(Y), f_2(Y), \ldots\}] \quad \text{(by Lemma 2.10)} \\ &= C_1[f(X) \cup f(Y)]. \end{aligned}$$

Next, suppose $x \in X$. We must show that there exists a $y \in f(X)$ such that $x^2 \sqsubseteq y^2$. Since f_1 is semantic, $\exists y_1 \in f_1(X) . x^2 \sqsubseteq y_1^2$; and since $f_1 \sqsubseteq f_2$, $\exists y_2 \in f_2(X) . y_1 \sqsubseteq y_2$; similarly $\exists y_3 \in f_3(X) . y_2 \sqsubseteq y_3$ and so on; $y = \text{lub}\{y_1, y_2, \ldots\}$ will belong to $f(X)$, and will satisfy $x^2 \sqsubseteq y^2$.

The converse is harder. Suppose $y \in f(X)$. If $y = \text{lub}\{y_1, y_2, \ldots\}$, $y_j \in f_j(X)$ for all j, then the result is immediate (since $y_1 \in f_1(X)$ and f_1 is a semantic function together imply that there exists a $x \in X$ such that $x^2 \sqsubseteq y_1^2 \sqsubseteq y^2$). If on the other hand, $y = \text{lub}\{y'_1, y'_2, \ldots\}$, $y'_j \sqsubseteq y'_{j+1}$, and there exist $y_1, y_2, \ldots$ such that $y'_j \sqsubseteq y_j$, and each y_j is of the form $\text{lub}\{y_{j1}, y_{j2}, \ldots\}$, $y_{jk} \sqsubseteq y_{j(k+1)}$, $y_{jk} \in f_k(X)$ for all k, then the following argument shows that the result is true: $y_{jk} \in f_k(X)$ implies there exists a $x_{jk} \in X$ such that $x_{jk}^2 \sqsubseteq y_{jk}^2$. Thus we have $x_{jk}^2 \sqsubseteq y_{jk}^2 \sqsubseteq y_j^2$ and $y_j'^2 \sqsubseteq y_j^2$. Hence, either there exists some j and k such that $x_{jk}^2 \sqsubseteq y_j'^2 \sqsubseteq y^2$; or for each j and k, $y_j'^2 \sqsubseteq x_{jk}^2$. In this case, by the completeness of X, and the fact that $y_j'^1 = \bot$ (hence $y'_j \sqsubseteq x_{jk}$), we can conclude that $y \in X$.

Thus f is indeed a semantic function. □

Next, consider functionals. All our functionals will map semantic functions to semantic functions.

Definition. A functional t is *monotonic* if $f \sqsubseteq g \Rightarrow t[f] \sqsubseteq t[g]$.

Definition. A monotonic functional t is *continuous* if for all chains $\{f_1, f_2, \ldots\}$,

$$t[\text{lub}\{f_1, f_2, \ldots\}] = \text{lub}\{t[f_1], t[f_2], \ldots\}.$$

(Note that the right-hand side above exists since $t[f_j] \sqsubseteq t[f_{j+1}]$, t being a monotonic functional.)

Theorem 2.13. *If t is a continuous functional, then* lub$\{\Omega, t[\Omega], t^2[\Omega], \ldots\}$ *is the least fixed point of t, and is a semantic function, where Ω is the following semantic function:*

$$\Omega(X) = \{x \mid x^1 = \bot \wedge \exists y \in X. x^2 = y^2\}.$$

Proof. The proof that lub$\{\Omega, t[\Omega], \ldots\}$ is the l.f.p. of t proceeds along standard lines; that it is a semantic function follows from Theorem 2.12, and the fact that Ω is a semantic function, and t maps semantic functions to semantic functions. □

Results

(1) *The functional t defined by $t[f] = f \circ g$, g being a given semantic function, is monotonic.*

Proof. Trivial. (The fact that t maps semantic functions to semantic functions follows from Theorem 2.11.) □

(2) *$t[f] = f \cup g$, g being a given semantic function, is a monotonic functional and maps semantic functions to semantic functions.*

Proof. Trivial. □

(3) *$t[f] = f \circ g$, f being a given semantic function, is a continuous functional.*

Proof. Suppose $\{f_1, f_2, \ldots\}$ is a chain of monotonic functions. Then,

$$t[\text{lub}\{f_1, f_2, \ldots\}](X) =$$
$$= \text{lub}\{f_1, f_2, \ldots\} \circ g(X)$$
$$= \text{lub}\{f_1, f_2, \ldots\}(g(X))$$
$$= \text{lub}\{f_1(g(X)), f_2(g(X)), \ldots\} \quad \text{(by construction of lub}\{f_1, f_2, \ldots\}\text{, Theorem 2.12)}$$
$$= \text{lub}\{f_1 \circ g(X), f_2 \circ g(X), \ldots\}$$
$$= \text{lub}\{t[f_1], t[f_2], \ldots\}(X).$$

Hence t is a continuous functional. □

(4) *$t[f] = f \cup g$ is a continuous functional.*

Proof. $t[\text{lub}\{f_1, f_2, \ldots\}](X) = C_1[\text{lub}\{f_1, f_2, \ldots\}(X) \cup g(X)]$. Hence, we need to show

$$C_1[\text{lub}\{f_1, f_2, \ldots\}(X) \cup g(X)] = \text{lub}\{f_1 \cup g(X), f_2 \cup g(X), \ldots\}.$$

Now,

$$\begin{aligned}
&\text{lub}\{f_1 \cup g(X), f_2 \cup g(X), \ldots\} = \\
&= \text{lub}\{C_1[f_1(X) \cup g(X)], C_1[f_2(X) \cup g(X)], \ldots\} \\
&= C_1[\text{lub}\{f_1(X), f_2(X), \ldots\} \cup \text{lub}\{g(X), g(X), \ldots\}] \quad \text{(by Lemma 2.10)} \\
&= C_1[\text{lub}\{f_1, f_2, \ldots\}(X) \cup g(X)].
\end{aligned}$$

Thus $t[f] = f \cup g$ is a continuous functional. □

(5) *The functional t defined below is continuous and maps semantic functions to semantic functions, g and g' being given semantic functions:*

$$t[f](X) = C_1[X^1 \cup g(C_2[X^2]) \cup f(g'(C_2[X^3]))]$$

where

$$\begin{aligned}
X^1 &= \{x \mid x \in X \wedge [x^1 = \bot \vee x^1 = a_i]\}, \\
X^2 &= \{x \mid x \in X \wedge x^1 \neq \bot \wedge x^1 \neq a_i \wedge b(x^1)\}, \\
X^3 &= \{x \mid x \in X \wedge x^1 \neq \bot \wedge x^1 \neq a_i \wedge b'(x^1)\},
\end{aligned}$$

where b, b' are boolean expressions that are always well defined if $x^1 \neq \bot \wedge x^1 \neq a_i$, and are such that $X^1 \cup X^2 \cup X^3 = X$.

Proof. We can easily see that X^1, X^2 and X^3 are convex; moreover, X^1 is also complete, hence we do not need to write $C_2[X^1]$ in the definition of t.

First consider the monotonicity of $t[f]$, f being a semantic function. Suppose $X \sqsubseteq Y$. We can easily show that there exist convex Y' and Y'' such that

$$Y^2 = X^2 \cup Y' \quad \text{and} \quad Y^3 = X^3 \cup Y'',$$

and

$$\forall y' \in Y'. \exists x \in X^1. [x \sqsubseteq y' \wedge x^1 = \bot], \qquad \forall y'' \in Y''. \exists x \in X^1. [x \sqsubseteq y'' \wedge x^1 = \bot].$$

We must first show that $\forall u \in t[f](X). \exists v \in t[f](Y). u \sqsubseteq v$. Suppose $u \in t[f](X)$. We can assume

$$u \in X^1 \cup g(C_2[X^2]) \cup f(g'(C_2[X^3])).$$

If $u \in X^1$, there exist $v \in Y$ such that $u \sqsubseteq v$. If $v \in Y^1$, we are done. If $v \in Y^2$, the fact that g is a semantic function implies that there exist $w \in g(C_2[Y^2])$ such that $v^1 \sqsubseteq w^2$, and hence $u \sqsubseteq w$. A similar argument works (f, g' being semantic functions) if $v \in Y$. If $u \in g(C_2[X^2])$, then the facts that g is a semantic function and $X^2 \subseteq Y^2$ (hence $C_2[X^2] \subseteq C_2[Y^2]$) imply that $u \in g(C_2[Y^2])$. A similar argument holds if $u \in f(g'(C_2[X^3]))$.

Thus we have shown $\forall u \in t[f](X).\exists v \in t[f](Y).u \sqsubseteq v$. Next we have to show $\forall v \in t[f](Y).\exists u \in t[f](X).u \sqsubseteq v$. Suppose $v \in t[f](Y)$. We can assume $v \in Y^1 \cup g(C_2[Y^2]) \cup f(g'(C_2[Y^3]))$. If $v \in Y^1$, it is easy to see that there exists a $u \in X^1$ such that $u \sqsubseteq v$. If $v \in g(C_2[Y^2])$: now $Y^2 = X^2 \cup Y'$. Hence,

$$g(C_2[Y^2]) = C_1[g(C_2[X^2]) \cup g(C_2[Y'])].$$

Hence there exists a $v' \in g(C_2[X^2]) \cup g(C_2[Y'])$ such that $v' \sqsubseteq v$. If $v' \in g(C_2[X^2])$, we are done; if $v' \in g(C_2[Y'])$, there exists a $v'' \in C_2[Y']$ such that $v''^2 \sqsubseteq v'^2$, g being a semantic function; hence there exists a $w \in Y'$ such that $w \sqsubseteq v''$; and there exists a $u \in X^1$ such that $u^1 = \bot \wedge u \sqsubseteq w$. Hence $u \sqsubseteq v$ and $u \in t[f](X)$. A similar argument holds if $v \in f(g'(C_2[Y^3]))$.

Next consider the montonicity of t. Suppose $f \sqsubseteq f'$. Then

$$t[f](X) = C_1[X^1 \cup g(C_2[X^2]) \cup f(g'(C_2[X^3]))],$$

$$t[f'](X) = C_1[X^1 \cup g(C_2[X^2]) \cup f'(g'(C_2[X^3]))].$$

It is easy to show that $t[f](X) \sqsubseteq t[f'](X)$. Thus t is monotonic. Now consider the continuity of t: suppose $\{f_1, f_2, \ldots\}$ is a chain. We need to show that $t[f](X) = \text{lub}\{t[f_1](X), t[f_2](X), \ldots\}$, f being the lub of $\{f_1, f_2, \ldots\}$. Now $\text{lub}\{t[f_1](X), t[f_2](X), \ldots\} = \text{lub}\{Y_1, Y_2, \ldots\}$, where

$$Y_i = C_1[X^1 \cup g(C_2[X^2]) \cup f_i(g'(C_2[X^3]))].$$

Hence, by Lemma 2.10,

$$\begin{aligned}
&\text{lub}\{t[f_1](X), t[f_2](X), \ldots\} = \\
&\quad = C_1[\text{lub}\{X^1, X^1, \ldots\} \cup \text{lub}\{g(C_2[X^2]), g(C_2[X^2]), \ldots\} \\
&\qquad \cup \text{lub}\{f_1(g'(C_2[X^3])), f_2(g'(C_2[X^3])), \ldots\}] \\
&\quad = C_1[X^1 \cup g(C_2[X^2]) \cup f(g'(C_2[X^3]))] \\
&\quad = t[f](X).
\end{aligned}$$

We still need to show that $t[f]$ is a semantic function if f is a semantic function. It is easy to see that $t[f](\Phi) = \Phi$. Next we have to show that

$$t[f](C_1[X \cup Y]) = C_1[t[f](X) \cup t[f](Y)].$$

Let Z denote $C_1[X \cup Y]$. Then the required result easily follows from the fact that f, g and g' are semantic functions and the following easily proved result:

$$Z^2 = X^2 \cup Y^2, \quad Z^3 = X^3 \cup Y^3 \quad \text{and} \quad Z^1 = C_1[X^1 \cup Y^1] \cup Z',$$

such that $\forall z' \in Z'.\exists z \in Z^2 \cup Z^3.z' \sqsubseteq z$.

Finally, we have to show that

$$\forall x \in X. \exists y \in t[f](X). x^2 \sqsubseteq y^2 \quad \text{and} \quad \forall y \in t[f](X). \exists x \in X. x^2 \sqsubseteq y^2.$$

The first result is trivial (using, in case $x \in X^2$ or X^3, the facts that f, g and g' are semantic functions).

The second result is also straightforward (using the results that if $z \in C_2[X^2] - X^2$, then $z \in X^1$, and similarly if $z \in C_2[X^3] - X^3$, then $z \in X^1$).

Thus $t[f]$ is a continuous functional and maps semantic functions to semantic functions. □

We have almost completed the discussion of the domains, functions and functionals needed for defining the denotations of individual CSP processes. We shall conclude by proving that a particular function (needed for defining the semantics of the selection statement) is a semantic function.

Lemma 2.14. *The function f defined as follows is a semantic function*:

$$f(X) = C_1[X^0 \cup f_1(C_2[X^1]) \cup \cdots \cup f_m(C_2[X^m]) \cup C_2[X^{m+1}]]$$

where

$$X^0 = \{x \mid x \in X \wedge [x^1 = \bot \vee x^1 = a_i]\},$$

$$X^{m+1} = \{z \mid z^1 = a_i \wedge \exists x \in X. x^1 \neq \bot \wedge x^1 \neq a_i \wedge x^2 = z^2 \wedge \neg b_1(x^1) \wedge \cdots \wedge \neg b_m(x^1)\},$$

where $b_1, \ldots, b_m$ are boolean expressions that are well defined if $x^1 \neq \bot$ and $x^1 \neq a_i$. (Essentially, b_j is the boolean portion of the j-th guard of the selection statement.)

$$X^j = \{z \mid z \in X \wedge z^1 \neq \bot \wedge z^1 \neq a_i \wedge b_j(z^1)\}, \quad 1 \leq j \leq m.$$

Proof. First consider the monotonicity of f. Suppose $X \sqsubseteq Y$. Then we can easily show for all j, $1 \leq j \leq m$,

$$Y^j = X^j \cup X'', \quad \text{where } X'' \text{ is such that } \forall y \in X''. \exists x \in X^0. x \sqsubseteq y.$$

Suppose now $x \in f(X)$. If $x \in X^0$, and $x^1 \neq \bot$, then $x \in Y^0$, hence $x \in f(Y)$; if $x^1 = \bot$, then there exists a $y \in Y$ such that $x \sqsubseteq y$. If $y \in Y^0$, we are done, since y will belong to $f(Y)$. If y is such that $y^1 \neq \bot \wedge y^1 \neq a_i \wedge \neg b_1(y^1) \wedge \cdots \wedge \neg b_m(y^1)$, then again we are done. If $y^1 \neq \bot \wedge y^1 \neq a_i \wedge b_j(y^1)$, we can find $z \in f_j(C_2[Y^j])$ which will satisfy $y^2 \sqsubseteq z^2$, hence $x \sqsubseteq z$. Thus we have shown

$$\forall x \in X^0. \exists y \in f(Y). x \sqsubseteq y.$$

If x is such that there exists a $u \in X$ such that

$$u^1 \neq \bot \wedge \neg b_1(u^1) \wedge \cdots \wedge \neg b_m(u^1) \wedge x^1 = a_i \wedge x^2 = u^2,$$

then u will be an element of Y and hence of $f(Y)$. A similar argument holds if $x \in f_j(C_2[X^j])$, making use of the fact that $X^j \subseteq Y^j$, and hence $f_j(C_2[X^j]) \subseteq f_j(C_2[Y^j])$.

Conversely, suppose $y \in f(Y)$. Then we can go through similar arguments to show that $\exists x \in f(X).x \sqsubseteq y$. (*Note*: the fact that

$$f_j(C_2[Y^j]) = C_1[f_j(C_2[X^j]) \cup f_j(C_2[X''^j])]$$

which follows from $Y^j = X^j \cup X''^j$ and f_j is a semantic function, is used in proving the converse.)

Next, we need to prove that f is a semantic function. It is easy to see that $f(\Phi) = \Phi$.

Consider $f(C_1[X \cup Y])$. It is straightforward to see that $[X \cup Y]^j = X^j \cup Y^j$ for all $j, 0 \leq j \leq m+1$. Hence, using the fact that f_j is a semantic function we can show that $f(C_1[X \cup Y]) = C_1[f(X) \cup f(Y)]$.

Finally, we have to show that

$$[\forall x \in X. \exists y \in f(X). x^2 \sqsubseteq y^2] \wedge [\forall y \in f(X). \exists x \in X. x^2 \sqsubseteq y^2].$$

This again is straightforward, using (for the second half of the result) the fact that $C_2[X^j] - X^j \subseteq X''$.

Thus f is indeed a semantic function. □

3. Semantics of individual processes

We are now ready to define the denotations corresponding to various statements that may appear in a CSP process, say P_i. As explained earlier, the denotations corresponding to a statement σ_i that appears in P_i will have the functionality

$$M[\sigma_i]: P(S_i \times H_i) \to P(S_i \times H_i).$$

All our functions will be semantic functions; the functions corresponding to the basic statements (**skip, assignment, input** and **output**) will be simple semantic functions, i.e., will also satisfy the conditions

$$f(X) = \bigcup_{x \in X} f(\{x\}), \qquad f(\{\langle \bot, h_i \rangle\}) = \{\langle \bot, h_i \rangle\}, \qquad f(\{\langle a_i, h_i \rangle\}) = \{\langle a_i, h_i \rangle\}.$$

For such functions we only need to define $f(\{\langle s_i, h_i \rangle\})$ for all $\langle s_i, h_i \rangle \in S_i \times H_i$, $s_i \neq \bot$, $s_i \neq a_i$, in order to specify f fully; in specifying such functions we shall often write $f(s_i, h_i)$ instead of the proper notation $f(\{\langle s_i, h_i \rangle\})$.

Now consider the various statements that may appear in P_i:

(1) **Skip**: $M[\textbf{skip}]$ is a simple semantic function:

$$M[\textbf{skip}](s_i, h_i) = \{\langle s_i, h_i \rangle\}$$

(2) **Assignment**: $M[x := e]$ is a simple semantic function:

$$M[x := e](s_i, h_i) = \{\langle s_i[x \leftarrow e(s_i)], h_i \rangle\},$$

where $e(s_i)$ is the value of the expression e in the state s_i; $s_i[x \leftarrow e(s_i)]$ is the state obtained by replacing the value of x by the value of e in the state s_i and leaving the other identifiers unchanged. (We assume that evaluation of expressions does not cause any problems, i.e., $e(s_i)$ is well defined if $s_i \neq \perp$ and $s_i \neq a_i$.)

(3) **Output**: $M[P_j!e]$ is a simple semantic function:

$$M[P_j!e](s_i, h_i) = \{\langle s_i, h_i\hat{}(i, j, e(s_i), \Phi)\rangle\}.$$

Recall that the fourth component of a communication element is the set of 'other options' open to P_i at this point, and in this case is the empty set, since P_i has no other options at the moment. "$\hat{}$" is the concatenation operation.

(4) **Input**: $M[P_j?x]$ is a simple semantic function:

$$M[P_j?x](s_i, h_i) = \{\langle s_i[x \leftarrow k], h_i\hat{}(j, i, k, \Phi)\rangle \mid k \in N\}.$$

(5) **Sequential composition**: $M[S_1; S_2](X) = M[S_2](M[S_1](X))$.

(6) **Selection**: Consider a selection statement; we allow mixed guards, i.e., some of the guards of the statement may be purely boolean while others are I/O guards. Suppose the statement is

$$[b_1; c_1 \rightarrow S_1 \,\square \cdots \square\, b_m; c_m \rightarrow S_m]$$

where b_j is the boolean portion of the jth guard ("**true**" if the jth guard is an I/O guard with no boolean portion), c_j is the communication portion of the jth guard (**skip** if the jth guard is purely boolean).

$$M[[b_1; c_1 \rightarrow S_1 \,\square \cdots \square\, b_m; c_m \rightarrow S_m]](X) =$$
$$= X^0 \cup f_1(g_1[C_2[X^1]]) \cup \cdots \cup f_m(g_m(C_2[X^m])) \cup C_2[X^{m+1}],$$

where

$$X^0 = \{x \mid x \in X \wedge [x^1 = \perp \vee x^1 = a_i]\},$$
$$X^{m+1} = \{y \mid y^1 = a_i \wedge \exists x \in X. x^1 \neq \perp \wedge x^1 \neq a_i \wedge x^2 = y^2$$
$$\wedge \neg b_1(x^1) \wedge \cdots \wedge \neg b_m(x^1)\}$$

(we assume that $b_1, \ldots, b_m$ are well defined if $x^1 \neq \perp$ and $x^1 \neq a_i$).

$$X^j = \{x \mid x \in X \wedge x^1 \neq \perp \wedge x^1 \neq a_i \wedge b_j(x^1)\}, \quad 1 \leq j \leq m.$$

f_j is the denotation of S_j $(1 \leq j \leq m)$. $g_1, \ldots, g_m$ will capture the effect of the I/O guards. $g_1, \ldots, g_m$ are simple semantic functions. The definition of g_j depends on whether the jth guard is purely boolean, input or output.

If the jth guard is purely boolean,

$$g_j(s_i, h_i) = \{\langle s_i, h_i\rangle\}.$$

If the jth guard is an output guard, say $b_j; P_j!e$, then,

$$g_j(s_i, h_i) = \{\langle s_i, h_i\hat{}(i, k, e(s_i), T)\rangle\},$$

where T, the set of other options, is defined as follows:

$$T=\{(i,k')\mid\exists l\leq m.l\in \mathrm{OG}\wedge b_l(s_i)\wedge \mathrm{CP}(l)=k'\wedge k'\neq k\}$$
$$\cup\{(k',i)\mid\exists l\leq m.l\in \mathrm{IG}\wedge b_l(s_i)\wedge \mathrm{CP}(l)=k'\}$$
$$\cup\{\iota\mid\exists l\leq m.l\in \mathrm{PB}\wedge b_l(s_i)\},$$

where OG, IG and PB are the sets of indices of the output, input and purely boolean guards respectively; CP(l) is the communication partner of P_i in the lth guard (if $l\in \mathrm{OG}\cup \mathrm{IG}$), i.e., CP($l$) is k if the lth guard is $b_l\,;P_k!e$ or $b_l\,;P_k?u$.

Finally, if the jth guard is an input guard,

$$g_j(s_i,h_i)=\{\langle s_i[u\leftarrow t],h_i\hat{\ }(k,i,t,T)\rangle\mid t\in N\},$$

where

$$T=\{(i,k')\mid\exists l\leq m.l\in \mathrm{OG}\wedge b_l(s_i)\wedge \mathrm{CP}(l)=k'\}$$
$$\cup\{(k',i)\mid\exists l\leq m.l\in \mathrm{IG}\wedge b_l(s_i)\wedge \mathrm{CP}(l)=k'\wedge k'\neq k\}$$
$$\cup\{\iota\mid\exists l\leq m.l\in \mathrm{PB}\wedge b_l(s_i)\}.$$

(*Note*: by the result at the end of Section 2, $M[[g_1\rightarrow S_1\,\square\cdots\square\, g_m\rightarrow S_m]]$ is a semantic function.)

The definition of the denotation of the selection statement seems somewhat complex, but, in fact, almost all of the complexity is due to the need for defining special notations to take care of the various cases that may arise, rather than any inherent complexity in the nature of the denotation.

(7) **Repetition:** Consider the repetition statement

$$*[b_1\,;c_1\rightarrow S_1\,\square\cdots\square\, b_m\,;c_m\rightarrow S_m].$$

The denotation of this statement is the least fixed point of the functional t defined as follows:

$$t[f](X)=C_1[X^0\cup f(Y^1)\cup\cdots\cup f(Y^m)\cup Y^{m+1}],$$

where

$$f_j=M[S_j],\quad 1\leq j\leq m,$$
$$X^0=\{x\mid x\in X\wedge[x^1=\perp\vee x^1=a_i]\},$$
$$Y^{m+1}=C_2[\{z\mid\exists x\in X.[x^1\neq\perp\wedge x^1\neq a_i\wedge \bigwedge_{l\in \mathrm{PB}}\neg b_l(x^1)$$
$$\wedge z^1=x^1\wedge z^2=x^2\hat{\ }(i,T,\tau,T')\}],$$

where

$$T=\{j\mid\exists l\leq m.[l\in \mathrm{IG}\vee l\in \mathrm{OG}]\wedge b_l(x^1)\wedge j=\mathrm{CP}(l)\},$$
$$T'=\{(i,j)\mid\exists l\leq m.l\in \mathrm{OG}\wedge b_l(x^1)\wedge j=\mathrm{CP}(l)\}$$
$$\cup\{(j,i)\mid\exists l\leq m.l\in \mathrm{IG}\wedge b_l(x^1)\wedge j=\mathrm{CP}(l)\},$$
$$Y^j=f_j(g_j(C_2[X^j]))\quad\text{for all } j,\ 1\leq j\leq m,$$

where

$$f_j = M[S_j], \quad 1 \leq j \leq m,$$

$$X^j = \{x \mid x \in X \wedge x^1 \neq \perp \wedge x^1 \neq a_i \wedge b_j(x^1)\}, \quad 1 \leq j \leq m,$$

and the g_j $(1 \leq j \leq m)$ are simple semantic functions; the definition of g_j depends on whether the jth guard is a purely boolean, output or input guard.

If the jth guard is purely boolean,

$$g_j(s_i, h_i) = \{\langle s_i, h_i \rangle\}.$$

If the jth guard is an output guard, say $b_j ; P_k!e$,

$$g_j(s_i, h_i) = \{\langle s_i, h_i \,\hat{}\,(i, k, e(s_i), T)\rangle\},$$

where

$$\begin{aligned} T = &\{(i, k') \mid \exists l \leq m. l \in \mathrm{OG} \wedge b_l(s_i) \wedge k' = \mathrm{CP}(l) \wedge k' \neq k\} \\ &\cup \{(k', i) \mid \exists l \leq m. l \in \mathrm{IG} \wedge b_l(s_i) \wedge k' = \mathrm{CP}(l)\} \\ &\cup \{(\iota \mid \exists l \leq m. l \in \mathrm{PB} \wedge b_l(s_i))\} \\ &\cup \{(i, T') \mid [\forall l \leq m. l \in \mathrm{PB} \Rightarrow \neg\, b_l(s_i)] \\ &\qquad \wedge [T' = \{k' \mid \exists l \leq m. l \in \mathrm{OG} \cup \mathrm{IG} \wedge b_l(s_i) \wedge k' = \mathrm{CP}(l)\}]\}. \end{aligned}$$

If the jth guard is an input guard, say $b_j ; P_k?u$, then

$$g_j(s_i, h_i) = \{\langle s_i[u \leftarrow t], h_i \,\hat{}\,(k, i, n, T) \mid n \in N\},$$

where T is defined in almost identical fashion as in the case of the output guard.

By the results of Section 2, t maps semantic functions to semantic functions and is a continuous functional, and it has a unique least fixed point which is a semantic function.

That completes the definition of the denotations of the various statements that may appear in the individual processes.

4. Semantics of CSP programs

Consider a CSP program $P :: [P_1 \| \ldots \| P_n]$. The semantics of the program will be a function

$$f_P : S_1 \times \cdots \times S_n \to P(S_1 \times \cdots \times S_n \cup \{\perp, d\}), \quad \text{where } S_i = S_i - \{\perp, a_i\};$$

i.e., for a given initial state $\langle s_1^1, \ldots, s_n^1 \rangle$ of $[P_1 \| \ldots \| P_n]$, f_P gives us the set of possible final states of the program; if the program can go into an infinite loop (either because of one (or more) of the processes going into an infinite loop, or because of 'infinite chattering'), $\perp$ will be one of the elements in the set of final states; if the program can get into other kinds of problems (deadlock between two or more processes, or abortion of one or more of the processes), the fail state "d" will be one of the elements in the set of final states. (*Note*: We do not distinguish between different kinds of failure, deadlock, abort, etc., since there does not seem to be any essential reason to do so, although it would be easy to modify the definition of f_p below to distinguish between the various types of problems that the program may get into.)

We shall neither define an order on the domains $S_1^- \times \cdots \times S_n^-$, $P(S_1^- \times \cdots \times S_n^- \cup \{\perp, d\})$, nor discuss such properties as monotonicity of functions such as f_p, continuity of functionals on such functions, since there seem to be many problems even at the informal operational level, in composing two CSP programs to obtain a new CSP program, or in constructing a new CSP programs by taking a given CSP program as the body of a loop etc., and we shall ignore such possibilities in this paper.

The function f_p is defined as follows:

$$f_p(\langle s_1^1, \ldots, s_n^1 \rangle) = B(f_1(\{\langle s_1^1, \varepsilon\rangle\}), \ldots, f_n(\{\langle s_n^1, \varepsilon\rangle\}))$$

where B is the n-fold 'binding operator', to be defined shortly, that will bind the semantics of $P_1, \ldots, P_n$ to obtain the semantics of the individual processes $P_1, \ldots, P_n$, respectively

$$B(X_1, \ldots, X_n) = Y_1 \cup Y_2 \cup Y_3,$$

where $X_j \in P(S_j \times H_j)$ $(j = 1, \ldots, n)$, and

$$Y_1 = \{\langle s_1, \ldots, s_n \rangle \mid s_1 \in S_1^- \wedge \cdots \wedge s_n \in S_n^-$$
$$\wedge \exists h_1, \ldots, h_n.[\forall i.\langle s_i, h_i\rangle \in X_i \wedge \mathrm{Compat}(h_1, \ldots, h_n)]\},$$

where

$$\mathrm{Compat}(h_1, \ldots, h_n) =$$
$$= \exists h.[h \in C^* \wedge \forall i.h/i = \mathrm{Trim}(h_i)$$
$$\wedge \forall k \leq \bar{h}.[\mathrm{Elem}(h, k) \in (i, T, \tau) \Rightarrow \forall j \in T.h[k+1 : \bar{h}]/j = \varepsilon]],$$
$$C = \{(i, j, l) \mid i \neq j, 1 \leq i, j \leq n, l \in N\}$$
$$\cup \{(i, T, \tau) \mid 1 \leq i \leq n, T \subseteq \{1, \ldots, i-1, i+1, \ldots, n\}\},$$

$\bar{h}$ is the number of elements in h; h/i the sequence obtained from h by omitting all elements except those of the form (i, j, l), (j, i, l) and (i, T, τ); $\mathrm{Trim}(h_i)$ the sequence obtained by dropping the fourth component of each element of h_i; $\mathrm{Elem}(h, k)$ the kth element of h; $h[k: l]$ the subsequence of h from the kth to lth element of h, i.e., $\langle \mathrm{Elem}(h, k), \ldots, \mathrm{Elem}(h, l)\rangle$.

Thus Y_1 corresponds to a situation in which each process terminates properly, the communication as recorded in each of the sequences being compatible with the communications in the other sequences.

Y_2 will correspond to a situation when two (or more) of the processes deadlock, or one (or more) of the processes 'aborts':

$$Y_2 = \{d \mid \exists s_1, \ldots, s_n, h_1, \ldots, h_n.[[\forall i \leq n.\langle s_i, h_i\rangle \in X_i]$$
$$\wedge [\exists i.s_i = a_i$$
$$\wedge [\exists h'_1, \ldots, h'_{i-1}, h'_{i+1}, \ldots, h'_n.[\forall j \neq i.h'_j \sqsubseteq h_j]$$
$$\wedge \text{Compat}(h'_1, \ldots, h'_{i-1}, h_i, h'_{i+1}, \ldots, h'_n)$$
$$\wedge \forall k \neq i.[\text{Elem}(h'_k, l) = (k, T, \tau, T')$$
$$\Rightarrow \forall j \in T.[j \neq i \wedge h'_j = h_j \wedge s_j \neq a_j \wedge s_j \neq \perp]]$$
$$\wedge [\text{Elem}(h_i, l) = (i, T, \tau, T')$$
$$\Rightarrow \forall j \in T.[h'_j = h_j \wedge s_j \neq a_j \wedge s_j \neq \perp]]]]]\}$$
$$\cup \{d \mid \exists s_1, \ldots, s_n, h_1, \ldots, h_n.[[\forall i \leq n.\langle s_i, h_i\rangle \in X_i]$$
$$\wedge [\exists h'_1, \ldots, h'_n.[\forall i \leq n.h'_i \sqsubseteq h_i \wedge \text{Compat}(h'_1, \ldots, h'_n)]$$
$$\wedge \forall i \leq n.[\text{Elem}(h'_i, l) = (i, T, \tau, T')$$
$$\Rightarrow \forall j \in T.h'_j = h_j \wedge s_j \neq \perp \wedge s_j \neq a_j]$$
$$\wedge \text{Incompat}(h''_1, \ldots, h''_n)]]\},$$

where $h''_i = h_i[\bar{h}'_i + 1 : \bar{h}_i]$, i.e., h''_i is the sequence got from h_i after stripping off the initial subsequence h'_i from it.

$$\text{Incompat}(h''_1, \ldots, h''_n) \equiv$$
$$\equiv [\exists i \leq n.h''_i \neq \varepsilon$$
$$\wedge [\forall i, j \leq n.[i \neq j \wedge \overline{h''_i} \geq 1 \wedge \overline{h''_j} \geq 1]$$
$$\Rightarrow \text{Options}(h''_i, 1) \cap \text{Options}(h''_j, 1) = \Phi]$$
$$\wedge [\forall i \leq n.\overline{h''_i} \geq 1$$
$$\Rightarrow [\iota \notin \text{Options}(h''_i, 1)$$
$$\wedge [(i, \tau) \in \text{Options}(h''_i, 1) \Rightarrow \exists j \in T.h''_i \neq \varepsilon]]]],$$

where

$$\text{Options}(h_i, 1) = \begin{cases} \{(i, j)\} \cup T & \text{if } \text{Elem}(h_i, 1) = (i, j, l, T), \\ \{(j, i)\} \cup T & \text{if } \text{Elem}(h_i, 1) = (j, i, l, T), \\ \{(i, T)\} \cup T' & \text{if } \text{Elem}(h_i, 1) = (i, T, l, T'). \end{cases}$$

Thus the first part of Y_2 corresponds to the situation when one of the processes aborts, while the second part corresponds to the situation when the program cannot continue because the various processes are attempting mutually incompatible communications.

$$Y_3 = \{\bot \mid \exists s_1, \ldots, s_n, h_1, \ldots, h_n.[[\forall i \leq n.\langle s_i, h_i\rangle \in X_i]$$
$$\wedge [\exists j.s_j = \bot$$
$$\wedge h_j \in C_j^* \Rightarrow [\exists h'_1, \ldots, h'_{j-1}, h'_{j+1}, \ldots, h'_n.[\forall i \neq j.h'_i \sqsubseteq h_i]$$
$$\wedge \text{Compat}(h'_1, \ldots, h'_{j-1}, h_j, h'_{j+1}, \ldots, h'_n)$$
$$\wedge \forall i \neq j.[\text{Elem}(h'_i, l) = (i, T, \tau, T')$$
$$\Rightarrow \forall k \in T.k \neq j \wedge s_k \neq \bot \wedge s_k \neq a_k \wedge h'_k = h_k]$$
$$\wedge [\text{Elem}(h'_j, l) = (j, T, \tau, T')$$
$$\Rightarrow \forall k \in T.h'_k = h_k \wedge s_k \neq \bot \wedge s_k \neq a_k]]$$
$$\wedge h_j \in C_j^\infty \Rightarrow [\forall m.[\exists h'_1, \ldots, h'_{j-1}, h'_{j+1}, \ldots, h'_n.[\forall i \neq j.h'_i \sqsubseteq h_i]$$
$$\wedge \text{Compat}(h'_1, \ldots, h'_{j-1}, h_j[1:m], h'_{j+1}, \ldots, h'_n)$$
$$\wedge \forall i \neq j.[\text{Elem}(h'_i, l) = (i, T, \tau, T')$$
$$\Rightarrow \forall k \in T.k \neq j \wedge h'_k = h_k \wedge s_k \neq \bot \wedge s_k \neq a_k]$$
$$\wedge \forall l \leq m.[\text{Elem}(h_j, l) = (j, T, \tau, T')$$
$$\Rightarrow \forall k \in T.h'_k = h_k \wedge s_k \neq \bot \wedge s_k \neq a_i]]]]]\},$$

where C_j^* is the set of finite sequences of elements of C_j, and C_j^∞ the set of all infinite sequences of elements of C_j.

Thus the first part of Y_3 (i.e., the part following $h_j \in C_j^* \Rightarrow$) corresponds to the case when P_j goes into an infinite loop after going through a finite number of communications, whereas the second part of Y_3 (the part following $h_j \in C_j^\infty \Rightarrow$) corresponds to the case when the loop is due to infinite chattering.

That completes the definition of B, and of the semantics of $[P_1 \| \ldots \| P_n]$. Our definition of the semantics of $[P_1 \| \ldots \| P_n]$ has been tailored to 'hide' the communications between the various processes, since these are *internal* to the program as a whole; on the other hand, the communications are *not* internal to the individual processes, and hence it was reasonable to include the communication sequence of a process in the semantics of the process. If desired, it would be relatively simple to modify the definition of the semantics so that the denotation f_p of $[P_1 \| \ldots \| P_n]$ has the functionality $f_p : S_n \times \cdots \times S_n \to P(S_1 \times \cdots \times S_n \times H)$, where H is the domain of (finite and infinite) sequences of communications between all pairs of processes; in other words, if h is an element of H, then h is obtained by 'merging' the sequences $h_1, \ldots, h_n$. We leave the changes that need to be made in the definition of B, in order to modify f_p in this fashion, to the interested reader.

5. Conclusions

In this paper we have presented a denotational semantics for CSP. Our semantics has the following advantages:

(a) The domains used in the semantics and the semantic functions are fairly simple. This may be contrasted with the domains and functions used by Francez et al. [3]. The approach of Pnueli et al. [6] is closer to our approach in that they use communication sequences in an essential fashion, and their domains are much simpler than those of Francez et al. [3]; in fact, the order on the domains of Pneuli et al. is even simpler than the order on our domains; however, we believe that, neither the denotations of individual processes nor the semantics of entire CSP programs as defined in Pneuli et al., are as closely related to their intuitive meanings as in our approach.

(b) Our semantics seems more 'abstract' than others that have been proposed. Consider, for instance, the process

$$P_1 :: x := 1 ; [x = 1 \rightarrow \textbf{skip} \,\Box\, \textbf{true} \rightarrow \textbf{skip}].$$

This will have the same semantics (in our approach) as the process $P_1 :: x := 1$ as indeed it should. This is not true of the semantics of Francez et al., where the semantics of the former process is a rather more complex tree than the semantics of the latter process. A similar remark applies to the semantics of Pnueli et al. [6]; in fact, in their semantics, even the process

$$P_1 :: x := 1 ; \textbf{skip}; \textbf{skip}; \textbf{skip}$$

would have a semantics different from the semantics of either of the processes given earlier.

Pnueli et al. justify this by saying that since the third process would take longer to execute than either of the other two processes, it should have a different semantics. But since the difference between the processes is only operational it would seem preferable to have identical denotational semantics for all three processes.

(c) Neither mixing of I/O and purely boolean guards nor the distributed termination convention of CSP causes any problems in our definitions.

There is one problem with our approach that is worth mentioning: we have required the elements of $P(S_i \times H_i)$ to be convex. This results in identical denotations for the following processes:

$$P_i :: [\textbf{true} \rightarrow \textbf{skip} \,\Box\, \textbf{true} \rightarrow P_j!5;\ P_j!5] ; * [\textbf{true} \rightarrow \textbf{skip}]$$

$$P_i :: [\textbf{true} \rightarrow \textbf{skip} \,\Box\, \textbf{true} \rightarrow P_j!5 \,\Box\, \textbf{true} \rightarrow P_j!5 ; P_j!5] ; * [\textbf{true} \rightarrow \textbf{skip}].$$

This seems rather undesirable, since the first process would necessarily go through with a second communication once it performs the first communication, while this is not true of the latter process. It is indeed possible to develop a somewhat more complex theory without imposing the requirement of convexity; however, even such a theory seems to have similar problems in more complex situations. In particular, the following processes have identical semantics, even in such a theory, although one would expect them to have distinct semantics:

$$P_i :: k := 0 ; * [k = 0 \rightarrow \mathbf{skip} \square k = 0 \rightarrow P_j!5 ; k := 1 \square k = 1 \rightarrow \mathbf{skip}$$
$$\square k = 1 \rightarrow P_j!5 ; k := 2] ;$$

$$P_i :: k := 0 ; * [k = 0 \rightarrow \mathbf{skip} \square k = 0 \rightarrow P_j!5 ; k := 1 \square k = 1 \rightarrow P_j!5 ; k := 2] ;$$

We believe that rather major changes would have to be made if these two processes are to have distinct semantics.

We imposed the requirement of completeness on the elements of $P(S_i \times H_i)$ in order to ensure continuity of the functionals needed in the definition of loops; informally, as explained in an earlier section, completeness amounts to requiring that if a loop can communicate an arbitrary number of times, it can also communicate forever. As a result, it would seem impossible, using our approach, to define a 'fair' semantics, since such a semantics would allow us to construct loops that terminate after communicating an arbitrary number of times. We believe that in order to deal with fairness, it would be necessary to introduce much more structure—perhaps in the form of a metric—on our basic domains, as De Bakker and Zucker [2] do.

Moreover, in our paper we have not considered the possibility of the parallel composition operator being used in one or more of the individual processes. It is indeed possible to generalize the theory to deal with such a construct, but we preferred not to develop such a theory at this stage, since there seems to be many unanswered questions even at the operational level, especially with respect to distributed termination, in such a language and these questions ought to be answered before attempting to define the denotational semantics of such a generalized language.

Before concluding it should be remarked that an approach quite similar to the one proposed in the current paper also works for a concurrent programming language in which processes interact through shared variables (rather than CSP type I/O statements). Such an approach would be preferable to (and 'more abstract' than) the standard approach of Milner [4] and Plotkin [5] involving powerdomains. This will be the topic of a future paper.

Acknowledgment

Much of this work was done while the author was on an extended visit to the Tata Institute of Fundamental Research, Bombay. The author would like to thank S. Arun Kumar, K. Lohya, S. Mahadevan and P. Pandya of the Tata Institute for many fruitful discussions.

Sincere thanks are also due to Marty Marlatt for her efficient typing of the paper.

References

[1] L. Boasson and M. Nivat, Adherence of languages, *JCSS* **20** (1980).

[2] J.W. de Bakker and J.I. Zucker, Processes and the denotational semantics of concurrency, *Information and Control* **54** (1982).

[3] N. Francez, C.A.R. Hoare, W.P. de Roever and D. Lehmann, Semantics of non-determinism, concurrency and communication, *JCSS* **19** (1979).

[4] R. Milner, Processes: A mathematical model of computing agents, *Logic Colloquium '73* (North-Holland, Amsterdam, 1975) pp. 157-174.

[5] G.D. Plotkin, A powerdomain construction, *SIAM J. Comput.* **5** (1976).

[6] A. Pnueli, N. Francez and D. Lehmann, A linear history semantics for languages for distributed programming, *IEEE 21st Symp. on Foundations of Computer Science*, 1980.