CONSENSITENCY AND COMPLETENESS OF A PROOF SYSTEM FOR CSP

by

Neelam Soundararajan

Abstract:

In Soundararajan [6] a proof system for dealing with partial correctness of Communicating Sequential Processes (CSP) was presented. Here we prove the consistency and (relative) completeness of this system, with respect to a formal operational semantics for CSP.
RESEARCH REPORT IN INFORMATICS

"Research report in Informatics" is a series of publications from University of Oslo, Norway, containing new results from computer science, numerical mathematics, software engineering and other parts of "informatics".

The series serves several objectives:

- to provide a faster way of publication than through journals,
- to allow usage of more space than in normal journal publications,
- to be used as study material in seminars, graduate education etc..

All texts will be in English.

The papers will be exchanged with similar productions from research groups elsewhere having corresponding interests.

Order for copies (duplication price) and other correspondence should be sent to:

Institutt for informatikk,
Universitetet i Oslo,
Postboks 1080, Blindern,
OSLO 3
Norway.
<table>
<thead>
<tr>
<th>No</th>
<th>Author(s)</th>
<th>Title</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Arne Wang</td>
<td>Generalized Types in High-level Programming Languages</td>
<td>Jan. 1975</td>
</tr>
<tr>
<td>2</td>
<td>Tom Lyche</td>
<td>Discrete Polynomial Spline Approximation Methods</td>
<td>Jan. 1975</td>
</tr>
<tr>
<td>3</td>
<td>Ole-Johan Dahl and Arne Jonassen</td>
<td>Analysis of an Algorithm for Priority Queue Administration</td>
<td>Jan. 1975</td>
</tr>
<tr>
<td>4</td>
<td>Olav Dahl</td>
<td>On the Problem of Solving Linear Algebraic Equations Associated with Matrices that are Polynomial Functions of a Square Matrix</td>
<td>Jan. 1975</td>
</tr>
<tr>
<td>5</td>
<td>Tom Lyche</td>
<td>Discrete Cubic Spline Interpolation</td>
<td>Feb. 1975</td>
</tr>
<tr>
<td>6</td>
<td>Arne Wang</td>
<td>An Axiomatic Basis for Proving Total Correctness of Goto-programs</td>
<td>April 1975</td>
</tr>
<tr>
<td>7</td>
<td>Tom Lyche</td>
<td>Computation of B-Spline Gram Matrices</td>
<td>June 1975</td>
</tr>
<tr>
<td>8</td>
<td>Arne Jonassen</td>
<td>Additional Notes on the Normal P-tree Forest</td>
<td>Sept. 1975</td>
</tr>
<tr>
<td>9</td>
<td>Arne Wang</td>
<td>Correctness of Transformations of Recursion to Iteration in Programs</td>
<td>Oct. 1975</td>
</tr>
<tr>
<td>10</td>
<td>Arne Wang</td>
<td>The Semantics of Programs. Weakest Preconditions Revisited</td>
<td>Nov. 1975</td>
</tr>
<tr>
<td>11</td>
<td>Tom Lyche</td>
<td>A Note on the Condition Numbers of the B-spline Bases</td>
<td>June 1976</td>
</tr>
<tr>
<td>14</td>
<td>Arne Jonassen</td>
<td>Priority Queue Processes with Biased Insertion of Exponentially Distributed Input Keys</td>
<td>May 1977</td>
</tr>
<tr>
<td>15</td>
<td>Arne Jonassen</td>
<td>A Formal Description of Data Table Processes</td>
<td>May 1977</td>
</tr>
<tr>
<td>No</td>
<td>Author(s)</td>
<td>Title</td>
<td>Publication Date</td>
</tr>
<tr>
<td>----</td>
<td>-----------</td>
<td>-------</td>
<td>------------------</td>
</tr>
<tr>
<td>16</td>
<td>Ragnar Winther</td>
<td>Iterative Methods for a Class of Linear Equations with Application to Galerkin Approximations of a Parabolic Control Problem.</td>
<td>May 1977</td>
</tr>
<tr>
<td>17</td>
<td>Stein Gjessing</td>
<td>Compile Time Preparations for Run Time Scheduling in Monitors.</td>
<td>June 1977</td>
</tr>
<tr>
<td>19</td>
<td>Arne Jonassen</td>
<td>A Study on the Reducibility of the Polynomials $x^n \pm x^m \pm 2p$.</td>
<td>Sept. 1977</td>
</tr>
<tr>
<td>20</td>
<td>Reiji Nakajima</td>
<td>Proving Abstract Specifications to be Valid for their Program Implementations.</td>
<td>Nov. 1977</td>
</tr>
<tr>
<td>22</td>
<td>Reiji Nakajima</td>
<td>Sype-Class of Types or Partial Types - for Program Specification Structuring; and their Formalization a First Order Logic.</td>
<td>Nov. 1977</td>
</tr>
<tr>
<td>23</td>
<td>Arne Jonassen</td>
<td>Analysis of the &quot;Dutch Flag Problem&quot;.</td>
<td>Nov. 1977</td>
</tr>
<tr>
<td>24</td>
<td>Arne Jonassen</td>
<td>Special Priority Queues with General Incident Sequences.</td>
<td>Nov. 1977</td>
</tr>
<tr>
<td>26</td>
<td>Olaf Owe</td>
<td>Notes on Partial Correctness.</td>
<td>Dec. 1977</td>
</tr>
<tr>
<td>29</td>
<td>Stein Gjessing</td>
<td>Monitors with Associated Processors</td>
<td>Feb. 1978</td>
</tr>
<tr>
<td>30</td>
<td>Sven Øivind Wille</td>
<td>Finite Element Formulations of Navier-Stokes Equations.</td>
<td>March 1978</td>
</tr>
<tr>
<td>31</td>
<td>Stein Gjessing</td>
<td>The Structure of the MUMI Multi Microcomputer System.</td>
<td>March 1978</td>
</tr>
<tr>
<td>No</td>
<td>Title</td>
<td>Author(s)</td>
<td>Date</td>
</tr>
<tr>
<td>----</td>
<td>----------------------------------------------------------------------</td>
<td>--------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>32</td>
<td>Maximal Clause Lenth Problem.</td>
<td>Vesko Marinov</td>
<td>April 1978</td>
</tr>
<tr>
<td>33</td>
<td>Can Program Proving be Made Practical?</td>
<td>Ole-Johan Dahl</td>
<td>May 1978</td>
</tr>
<tr>
<td>34</td>
<td>The Limiting Behaviour of P-trees with Incident Sequence $O1^n D^n$</td>
<td>Arne Jonassen</td>
<td>August 1978</td>
</tr>
<tr>
<td>35</td>
<td>The &quot;Chain Gang Schedule&quot;, Theoretical Solution.</td>
<td>Arne Jonassen</td>
<td>August 1978</td>
</tr>
<tr>
<td>36</td>
<td>The Polynomials $x^m x^n 2p$, A new Approach to the Question of Irreducibility.</td>
<td>Arne Jonassen</td>
<td>August 1978</td>
</tr>
<tr>
<td>37</td>
<td>A Newton form for trigonometric Hermite interpolation.</td>
<td>Tom Lyche</td>
<td>Sept. 1978</td>
</tr>
<tr>
<td>40</td>
<td>A Divided Difference Formula for the Error in Numerical Differentiation.</td>
<td>T. Dokken and T. Lyche</td>
<td>Dec. 1978</td>
</tr>
<tr>
<td>41</td>
<td>A study on some Pseudo-Random Generators for a 36-bit Computer.</td>
<td>Arne Jonassen, Per-Arne Rindalsholt, Rolf Spådahl and Gunnar Valsø</td>
<td>Dec. 1978</td>
</tr>
<tr>
<td>42</td>
<td>Conditional and Joint Possibilities of Higher Order.</td>
<td>Ellen Hisdal</td>
<td>July 1978</td>
</tr>
<tr>
<td>43</td>
<td>Concrete and Mathematical Sets, Measures of Second Order Possibilities.</td>
<td>Ellen Hisdal</td>
<td>Aug. 1978</td>
</tr>
<tr>
<td>44</td>
<td>Particularization - The Theory of Fuzzy Sets versus Classical Theories.</td>
<td>Ellen Hisdal</td>
<td>Nov. 1978</td>
</tr>
<tr>
<td>45</td>
<td>Monitors with arrays of condition variables and proof rules handling local quantities.</td>
<td>Stein Gjessing</td>
<td>June 1979</td>
</tr>
<tr>
<td>46</td>
<td>Microcomputer Interrupt Servicing in an Environment of Processes and Monitors.</td>
<td>Stein Gjessing</td>
<td>June 1979</td>
</tr>
<tr>
<td>47</td>
<td>Microcomputer Software Design and Programming using the Concepts of Processes and Monitors - a Case Study.</td>
<td>Stein Gjessing</td>
<td>June 1979</td>
</tr>
</tbody>
</table>
No 48 Ole-Johan Dahl: Time Sequences as a tool for describing program behaviour Aug. 1979
No 49 Olav Dahl: On the choice of pivot elements in Gaussian elimination March 1980
No 50 Olaf Owe: A specification Technique with Idealization March 1980
No 51 Ellen Hisdal: The IF THEN ELSE Statement and Interval-Valued Fuzzy Sets of Higher Type May 1980
No 52 Per M.A. Bothner: High-level data types in programming languages Oct. 1980
No 54 Robert W. Gunderson: Choosing the r-dimension for the FVC family of clustering algorithms March 1981
No 55 Vesko Marinov: Resolution theorem prover in a program verification environment May 1981
No 56 Neelam Soundararajan: Axiomatic semantics of communicating sequential processes May 1981
No 58 Neelam Soundararajan: Consistency and completeness of a proof system for CSP June 1981
No 59 Neelam Soundararajan: Proofs of correctness of CSP programs June 1981
1. Introduction

One of the most elegant languages for parallel programming is the language of Communicating Sequential Processes (CSP) introduced by Hoare[4]. In [6] we presented an axiomatic semantics for CSP; the sequences of communications exchanged by the processes play a rather prominent role in the semantics specified in [6]. From a formal point of view, the system of [6] has the important advantage that the proofs of the individual processes can be given independently of each other; this may be contrasted with the proof system of Apt et al [3] where the proofs of the individual processes must be verified to "cooperate" with each other before one can arrive at any conclusion regarding the entire program. From a practical point of view, the fact that proofs of the individual processes of a program may be given independently of each other results in fairly simple proofs of program properties. This has been, we feel, amply demonstrated by the proofs described in [7] of several CSP programs.

Our goal in this paper is a rather theoretical one: we wish to demonstrate that the system of [6] is consistent and complete. The paper is organised as follows: in section 2 we present our axiomatic semantics for CSP. The semantics presented in section 2 is slightly different from the one defined in [6]; in particular, the rule of inference for I/O guarded commands presented in section 2 is different from the corresponding rule in [6]; the reason for the change is that, with the modified rule, the problem of "conspiracy among proofs" of [6] is rather easily handled.

In section 3, we introduce an operational semantics of CSP, consistent with the intuitive semantics given in Hoare[4]. We then prove that the semantics of section 2 is consistent with respect to this operational semantics. In section 4 we try to prove the completeness of the axiomatic system; in order to do this, we first introduce a simpler axiomatic system; we then show that any proof in the new system can be converted into a proof in the earlier system; finally we show that the new system is complete with respect to the operational semantics, thus proving the completeness of the original system.

In this paper, no informal justification is given for the various
axioms and rules of inference of the axiomatic system; the interested reader may refer to [6] for the same.

2. Axiomatic semantics of CSP

Consider a CSP program \( P_1 // P_2 // \ldots // P_n \), \( P_1, \ldots, P_n \) being the communicating sequential processes. (We assume that \( P_1, \ldots, P_n \) do not themselves contain the "//" construct; thus parallelism exists only at the outermost level, the individual processes being strictly sequential). \( h_i \) will denote the communication sequence from \( P_i \) to the remaining \((n-1)\) processes. Thus \( h_i \) will be a sequence of elements of the form \((i,j,p)\), which denotes the number \( p \) being sent by \( P_i \) to \( P_j \) (by an output statement of the form \( P_j!p \) in the processes \( P_i \)), and elements of the form \((j,i,m)\), denoting the number \( m \) being received by \( P_i \) from \( P_j \) (in response to an input statement of the form \( P_j?x \) in \( P_i \)); \( h_i \) may also include another type of element, of the form \((i,j,t)\) whose purpose will become clear later.

In proving some property of \( P_i \), the system allows us to make assumptions regarding the sequence from \( P_j \) to \( P_i \). We shall use the symbol \( h'_{ji} \) to denote the sequence from \( P_j \) to \( P_i \); as \( P_i \) is executed, it will "consume" portions of \( h'_{ji} \) (one element per input or output statement of the form \( P_j?x \) or \( P_j!n \)); \( h_{ji} \) will denote the portion of the sequence from \( P_j \) to \( P_i \) that \( P_i \) has "already" consumed, \( h'_{ji} \) denoting the portion that \( P_i \) is "yet" to consume. Thus, at the "start" of \( P_i \), we shall have a predicate of the form \( I_i \land [h_i = \epsilon] \land [\bigwedge_i h_{ji} = \epsilon] \land A_{ji} (h'_{ji}) \), where \( I_i \) is a given predicate regarding \( s_i \), the local state of \( P_i \); \( h_i = \epsilon \) and \( \bigwedge_i h_{ji} = \epsilon \) indicate the facts that, at the start, \( P_i \) has neither produced, nor consumed anything. We shall denote the set of objects \( s_i, h_i, h_{ij}(j+i), h'_{ji}(j+i) \), collectively as the state \( \sigma_i \) of \( P_i \).

We are now ready to state the axioms and rules of inference for the various constructs that may appear in the individual processes. We shall state these as applicable to \( P_i \). (Note: We have assumed that the processes are numbered \( 1 \) through \( n \); if, instead, processes can be given arbitrary names, we would have to make the obvious changes in the system; thus the communication sequences will be made up of elements of the form \((X,Y,n)\) to denote the number \( n \) being sent from the process \( X \) to the process \( Y \). Note also that we do not consider declarations in our system. Declarations may be treated in the standard fashion).
A1. Input

\[ [(D(h'_{ji})=(j,i)) \land p'] P_j ? x \{p\} \]

where

\[ p' = P_{V(h'_{ji}), h_i \rightarrow (j,i), V(h'_{ji}), h_{ji} \rightarrow \text{first}(h'_{ji}), \text{rest}(h'_{ji})} \]

\( \text{first}(h) \) is the leftmost element of the sequence \( h \); \( \text{rest}(h) \) is the sequence got by deleting the leftmost element of \( h \).

"|" denotes concatenation of an element to the right end of a sequence. Similarly, "\(|" will denote concatenation of an element to the left end of a sequence, and "\(|" will denote concatenation of two sequences.

\[ D(h) = t' \text{ if } h = \varepsilon \]
\[ t \text{ if } \text{first}(h) = (k, l, t) \text{ for some } k, l. \]
\[ (k, l) \text{ if } \text{first}(h) = (k, l, m) \text{ for some integer } m. \]

\[ V(h) = 0 \text{ if } h = \varepsilon , \text{ or if } D(h) = t , \]
\[ m \text{ if } \text{first}(h) = (k, l, m). \]

(Note: We assume that only integers are being exchanged between the processes.)

The above axiom alone is not sufficient for input; we also need the following rule of inference:

R1. Rule of inference for input

\[ p \Rightarrow \exists h'_{ji} . [p' \land [D(h'_{ji})=(j,i)]] \]
\[ p \land [D(h'_{ji})=(j,i)] P_j ? x \{q\} \]
\[ \{p\} P_j ? x \{q\} \]
\[ h'_{ji} \]

where \( p' = P_{\overline{h'_{ji}}} \)

A2. Output

\[ [(D(h'_{ji})=(i,j)) \land [V(h'_{ji})=y] \land p'] P_j : y \{p\} \]

where

\[ p' = P_{h_i \rightarrow (i,j,y), h_{ji} \rightarrow \text{first}(h'_{ji}), \text{rest}(h'_{ji})} \].
R2. Rule of inference for output

\[ p \Rightarrow \exists h'_{ji} \cdot [p' \land D(h'_{ji}) = (i, j)] \land V(h'_{ji}) = y ] \]

\[ \{ p \land D(h'_{ji}) = (i, j) \land V(h'_{ji}) = y \} \quad P_j \downarrow y \{ q \} \]

\[ \{ p \} \quad P_j \uparrow y \{ q \} \]

where \( p' = p_{h'_{ji}} \)

A3. Assignment

\[ \{ p_x \} \quad x := e \{ p \} \]

A4. Skip

\[ \{ p \} \quad \text{skip} \{ p \} \]

Note: In the rules R3, R4, R5 and R6 below, \( q' \) will denote the following predicate:

\[ q' = \exists \sigma_i \cdot [q_{\sigma_i} \land [h_i \parallel h_i] ] \]

where \( h_i \parallel h \) is true if \( h \) is an initial subsequence of \( h \).

R3. Boolean guarded selection

\[ p \Rightarrow q' \]

\[ \{ p \land b_r \} \quad S_r \{ q \}, \quad r = 1, \ldots, m \]

\[ \{ p \} \quad [\sigma (r = 1, \ldots, m) b_r \rightarrow S_r ] \{ q \} \]

R4. Boolean repetition

\[ p \Rightarrow q' \]

\[ [p \land \gamma b] \Rightarrow q \]

\[ \{ p \land b_r \} \quad S_r \{ p \}, \quad r = 1, \ldots, m \]

\[ \{ p \} \quad [\sigma (r = 1, \ldots, m) b_r \rightarrow S_r ] \{ q \} \]

where \( b = \bigvee_{r=1}^{m} b_r \).

R5. I/O guarded selection

\[ p \Rightarrow q' \]

\[ \{ p \land b_r \} \quad \alpha_r \{ q_r \}, \quad \{ q_r \} \quad S_r \{ q \}, \quad r = 1, \ldots, m \]

\[ \{ p \} \quad [\sigma (r = 1, \ldots, m) b_r \rightarrow \alpha_r \rightarrow S_r ] \{ q \} \]

where \( \alpha_r \) are I/O statements.
R6. I/O guarded repetition

\[ p \Rightarrow q' \]
\[ [p \land \forall b \land [\forall b_r^m := (h'_{c(c(a_r^j) \neq \varepsilon)})]^{h_i}_{l \leq m} \Rightarrow [q_{h_i}, \Gamma, (i, (j \mid \exists r \leq m. (b_r \land c(a_r) = j)), t)]^t] \]
\[ \{ p \land b_r \land (h'_{c(a_r^j) \neq \varepsilon}) \} \alpha_r(q_r), \{ q_r \} S_r(q), r = 1, \ldots, m \]
\[ \{ p \} \alpha_r(q_r) \]

where

\[ c(a_r) = j \text{ if } a_r \text{ is } P_j'y \text{ or } P_j'x; \]

\[ b = \lor_{r=1}^m b_r. \]

The second line of R6 corresponds to the situation where the loop terminates because the boolean portions of the guards are all false. The third line corresponds to the case when the loop has to terminate despite some of the booleans being true, on account of the fact that the sequences from the other processes are all \( \varepsilon \). In this case, we extend the sequence \( h_i \) by the term \( (i, T, t) \) where \( T \) is the set of indices of those processes with which \( P_i \) was willing to continue communicating; the "t" is a special symbol which indicates that \( P_i \)'s loop terminated because the processes mentioned in \( T \) had terminated. This element needs to be added to \( h_i \) in order to avoid the problem of conspiring proofs of \([6]\). The first lines of R3 through R6 are needed to ensure consistency of the system.

R7. Sequential composition

\[ \{ p \} S_1(q_1), \{ q_1 \} S_2(q) \]
\[ \{ p \} S_1; S_2(q) \]

R8. Consequence

\[ p = p_1, \{ p_1 \} S(q_1), q_1 \Rightarrow q \]
\[ \{ p \} S(q) \]

R9. Conjunction

\[ \{ p \} S(q_1), \{ p \} S(q_2) \]
\[ \{ p \} S(q_1 \land q_2) \]
R10. Disjunction

\[
\{p_1\} S\{q\}, \{p_2\} S\{q\} \\
\{p_1 \lor p_2\} S\{q\}
\]

R11. Parallel composition

\[
\exists h'_j . C_{ji}, \quad j, i = 1, \ldots, n, \quad j \neq i
\]

\[
\left[ F_i \land \left( \tilde{h}_{ij} = h_i / \{(i,j),(j,i)\} \right) \rightarrow C_{ij} \right] \lor 1 \leq j \leq n, \quad j \neq i, \quad i = 1, \ldots, n
\]

\[
\left[ I_i \land (h_i = \varepsilon) \land \left( \land_{j \neq i} h_{ij} = \varepsilon \right) \land \left( \land_{j \neq i} C_{ji} \right) \right] P_i \{F_i\}, \quad i = 1, \ldots, n
\]

\[
\{ \land_{i=1}^{m} I_i \} \{ \land_{i=1}^{n} P_i \} \left( \land_{j \neq i} \right) \left[ \text{Reduce}(h_1, \ldots, h_n) = (\varepsilon, \ldots, \varepsilon) \right]
\]

The first line is needed to ensure consistency of the system. The second line says that the assumption \(C_{ij}\) made in the proof of \(P_j\), regarding the sequence from \(P_i\) to \(P_j\) must be justified by the post-condition of \(P_i\); \(\tilde{h}_{ij} = h_i / \{(i,j),(j,i)\}\) indicates that \(\tilde{h}_{ij}\) is obtained from \(h_i\) by retaining elements of the form \((i,j,m)\) or \((j,i,m)\), changing elements of the form \((i,T,t)\) into \((i,j,t)\) if \(j \in T\), and omitting all other elements.

The predicate \([\text{Reduce}(h_1, \ldots, h_n) = (\varepsilon, \ldots, \varepsilon)]\) expresses a consistency condition on the communication sequences of the various processes. \(\text{Reduce}\) is a function of \(n\) sequences whose value is also \(n\) sequences. \(\text{Reduce}(h_1, \ldots, h_n)\) is defined as follows: if two of the sequences \(h_i, h_j\) are such that \(\text{first}(h_i) = \text{first}(h_j)\) then,

\[
\text{Reduce}(h_1, \ldots, h_n) = \text{Reduce}(h'_1, \ldots, h'_n),
\]

where \(h'_i = \text{rest}(h_i), h'_j = \text{rest}(h_j), h'_k = h_k\) for all \(k \neq i,j\). If there is no such pair of sequences, and if \(\text{first}(h_i)\) is \((i,T,t)\) and for all \(j \in T, h_j = \varepsilon\) then,

\[
\text{Reduce}(h_1, \ldots, h_n) = \text{Reduce}(h'_1, \ldots, h'_n),
\]

where \(h'_i = \text{rest}(h_i)\) and \(h'_k = h_k\) for all \(k \neq i\). If there are no sequences satisfying either of the above conditions then

\[
\text{Reduce}(h_1, \ldots, h_n) = (h_1, \ldots, h_n).
\]

(Note: If there is more than one pair of sequences satisfying the first condition or more than one sequence satisfying the second, the reduc-
tion may be performed in any order, since the final result will be the same).

We could have added another clause to the final post-condition:
\[ \bigwedge_{i=1}^{n} \bigwedge_{j \neq i} h_{ij} = h_i / \{(i,j), (j,i)\} \], to express the fact that the sequence from \( P_i \) that \( P_j \) has consumed is, in fact, the sequence sent by \( P_i \) to \( P_j \); we did not do this, allowing this fact to follow from the notation; thus in any predicate in which both \( h_i \) and \( h_{ij} \) occur, we may assume the \( h_{ij} = h_i / \{(i,j), (j,i)\} \).

Before concluding this section, we introduce some concepts which will be useful in the next section. These concepts have been modelled on the analogous concepts for parallel programs introduced by Owicki[5].

Suppose \( P_i \) is a communicating sequential process. The primary components of \( P_i \) are

1) none if \( P_i \) is a skip, an assignment, or an I/O statement.
2) \( S_1, \ldots, S_r \) if \( P_i \) is \( S_1 ; S_2 ; \ldots ; S_r \).
3) \( S_1, \ldots, S_m \) if \( P_i \) is \([\sigma(\ell=1, \ldots, m) b_\ell \rightarrow S_\ell] \), or \([\sigma(\ell=1, \ldots, m) b_\ell \rightarrow S_\ell] \).
4) \( \alpha_i, S_1, \ldots, \alpha_m, S_m \) if \( P_i \) is \([\sigma(\ell=1, \ldots, m) b_\ell \alpha_\ell \rightarrow S_\ell] \), or \([\sigma(\ell=1, \ldots, m) b_\ell \alpha_\ell \rightarrow S_\ell] \).

The proper components of \( P_i \) are the primary components of \( P_i \), and their proper components. The components of \( P_i \) are \( P_i \) itself and its proper components.

Suppose pre and post are two functions which map components of \( P_i \) to assertions. Then pre and post are assertion functions for \((p)P_i(q)\) if the following conditions are satisfied:

1) \( p \rightleftharpoons \text{pre}(P_i) \) and \( \text{post}(P_i) \rightleftharpoons q \).

Conditions (2) through (10) must be satisfied for every component \( S \) of \( P_i \).

2) If \( S \) is skip then \( \text{pre}(S) \rightleftharpoons \text{post}(S) \).
3) If \( S \) is \text{x} := e then \( \text{pre}(S) \rightleftharpoons \text{post}(S) \rightleftharpoons e \).
4) If \( S \) is \( P_j \) \( y \) then \( \text{pre}(S) \rightleftharpoons [\text{pre}(S) \rightleftharpoons h^i_{ji} \leftarrow D(h^i_{ji}) = (i,j)] \land [V(h^i_{ji}) = y] \), and

\[ [\text{pre}(S) \land [D(h^i_{ji}) = (i,j)] \land [V(h^i_{ji}) = y] \Rightarrow \]

\[ \text{post}(S), h^i_{ji} \leftarrow (i,j,y), h^i_{ji} \leftarrow \text{first}(h^i_{ji}), \text{rest}(h^i_{ji}) \]
Suppose now that \{p\}P_1[q] is provable using the axioms and rules of inference specified earlier in this section. It should then be clear (and can be formally shown) that there exist assertion functions pre and post for \{p\}P_1[q]; and, conversely, if pre and post are assertion functions for \{p\}P_1[q], then \{p\}P_1[q] is provable.

Next suppose P :: [P_1 // ... // P_n] is a CSP program, and that using the axioms and rules of inference we are able to prove \{I\}P_1 // ... // P_n{F}. Then (R11 being the rule for parallel composition), we may assume that I can be written in the form \[ \bigwedge_{i=1}^n I_i, \] \( I_i \) being predicates regarding the local state s_i of P_i, F can be written as \[ [ \bigwedge_{i=1}^n F_i ] \bigwedge \text{Reduce}(h_1, ..., h_n) = (\epsilon, ..., \epsilon) \], F_i being a predicate on s_i, h_i, h_j (j \neq i) and h'_j, and that there exist predicates C_{i,j} (i \neq j) satisfying the requirements stated in the first two lines of R11, and that \( \{I_i \land (h_i = \epsilon) \land (\bigwedge_{j \neq i} C_{i,j})\} P_i(F_i), i = 1, ..., n \) are provable using our system. Thus there exist (n pairs of) assertion functions \( \text{pre}_i, \text{post}_i \) (i = 1, ..., n), \( \text{pre}_i, \text{post}_i \) being assertion functions for \( \{I_i \land (h_i = \epsilon) \land (\bigwedge_{j \neq i} C_{i,j})\} P_i(F_i) \). In the next section we shall assume that any proof of the program P is specified by giving
the assertion functions $\text{pre}_i, \text{post}_i, i = 1, \ldots, n$.

We conclude this section with one further remark: suppose $\text{pre}_i', \text{post}_i'$ are assertion functions for $\{p\}P_i\{q\}$; suppose also that $S$ is a component of $P_i$. Then the following facts are easily proved:

$\text{pre}(S) \Rightarrow \exists \sigma_i \exists h_i. [\sigma_i^- \land (h_i \triangleright h_i)], \text{ and } \text{post}(S) \Rightarrow \exists \sigma_i \exists h_i. [\sigma_i^+ \land (h_i \triangleleft h_i)].$

These two results will be of importance in proving the consistency of the system.

3. Consistency of the system

We must first introduce an operational model for the execution of CSP programs. The model we use is based on the model of Apt [1]. The model is defined by specifying the relation "→" between pairs consisting of a program and a state; essentially "→" denotes the execution of one step of the program. (If this step is a communication between $P_i$ and $P_j$, this will be the execution of an input statement $P_j?x$ in $P_i$ and an output statement $P_i!y$ in $P_j$).

For convenience, we shall allow the empty process $E$. (Thus after executing the final step of a process, we shall denote the process by $E$). We shall use the symbol $\tau$ (as well as $\tau', \tau^\prime$ etc.) to denote the current state; thus $\tau$ will be an n-tuple $(\tau_1, \ldots, \tau_n)$ where $\tau_i$ is the state of the $i^{th}$ process. $\tau_i$ will have two components: $s_i$, the local state, specifying the values of the variables of the $i^{th}$ process, and $h_i$, the communication sequence of the $i^{th}$ process. Thus $(P, \tau) \rightarrow (P', \tau')$ specifies that executing $P$ one step in the state $\tau$ can lead to the state $\tau'$ with $P'$ being the program yet to be executed. (of course, "→" does not specify a function; thus there may be more than one $(P', \tau')$, which may arise by execution of one step of $P$). With these preliminaries taken care of, we can now define "→".

Definition: $(S_1//\ldots//S_n, \tau) \rightarrow (S'_1//\ldots//S'_n, \tau')$ if any one of the following seventeen clauses is satisfied:

(In the various clauses, we ought to have included quantifiers of the kind $\exists i. 1 \leq i \leq n$, but these have been omitted; this omission should cause no confusion. Note also that more than one clause cannot be simultaneously satisfied).

1) $S_i = \text{skip}$, and $S'_i = E$, $S'_j = S_j$ for all $j \neq i$; and $\tau' = \tau$.  
2) $S_i = x:=e$, and $S'_i = E$, $S'_j = S_j$ for all $j \neq i, \tau_i = \tau_i[x=e], \tau'_j = \tau'_j$ for all $j \neq i$,

where $\tau'_i = \tau_i[x=e]$ indicates that $\tau'_i$ is the same state as $\tau_i$.  

except that the value of the variable \( x \) has been changed to be equal to the value of the expression \( e \), evaluated in the state \( \tau_i \).

In the following, we shall omit clauses of the kind \( S'_j = S_j \) for all \( j \neq i \); thus \( S'_k = S_k \) unless otherwise specified; similarly we shall omit \( \tau'_j = \tau_j \) for all \( j \neq i \), the understanding being that those components of \( \tau' \) that are not mentioned are identical to the corresponding components of \( \tau \).

3) \( S_i = [\Box (\ell=1, \ldots, m) b_\ell \rightarrow T^0_i] \), and

\[ \models b_\tau(\tau_i), S'_i \models T^r_i, \text{ and } \tau' = \tau. \]

\( \models b_\tau(\tau_i) \) indicates that the boolean \( b_\tau \) has the value true in state \( \tau_i \).

4) \( S_i = * [\Box (\ell=1, \ldots, m) b_\ell \rightarrow T^0_i] \), and

\[ \models (\bigwedge_{\ell=1}^m \neg b_\ell)(\tau_i), S'_i \models E, \text{ and } \tau' = \tau. \]

(all the guards are false, and hence the loop terminates).

5) \( S_i = *[\Box (\ell=1, \ldots, m) b_\ell \rightarrow T^0_i] \), and

\[ \models b_\tau(\tau_i), S'_i \models T^r_i; S_i, \text{ and } \tau' = \tau. \]

6) \( S_i = P_j ? x, S_j = P_i ! y, \) and

\( S'_i \models E, S'_j \models E, \tau'_j = \tau_j[h_i \vdash h_j \vdash (j, i, \tau_j(y))] \), \( \tau'_i = \tau_i [x \vdash \tau_j(y), h_i \vdash h_j \vdash (j, i, \tau_j(y))] \)

(The effect of performing the communication is to assign to \( x \), the current value of \( y \), and extend the sequences \( h_i, h_j \) by the term \((j, i, m), m \) being the value of \( y \) in the state \( \tau_j \)).

7) \( S_i = [\Box (\ell=1, \ldots, m) b_\ell \alpha_\ell \rightarrow T^0_i] \), and

\[ \models b_\tau(\tau_i), \alpha_\tau = P_j ? x, S_j \models P_i ! y, \text{ and } \]

\( S'_i \models \alpha_\tau; T^r_i, \text{ and } \tau' = \tau. \)

8) \( S_i = [\Box (\ell=1, \ldots, m) b_\ell \alpha_\ell \rightarrow T^0_i] \), and

\[ \models b_\tau(\tau_i), \alpha_\tau = P_j ! x, S_j \models P_i ? y, \text{ and } \]

\( S'_i \models \alpha_\tau; T^r_i, \text{ and } \tau' = \tau. \)

9) \( S_i = [\Box (\ell=1, \ldots, m) (b_\ell \alpha_\ell \rightarrow T^0_i)], S_j = [\Box (\ell'=1, \ldots, m') b_\ell' \alpha_\ell' \rightarrow T^0_i'] \), and

\[ \models b_\tau(\tau_i), \alpha_\tau = P_j ? x, \models b_\tau'(\tau_j), \alpha_\tau' = P_i ! y, \text{ and } \]

\( S'_i \models \alpha_\tau; T^r_i, S'_j \models \alpha_\tau' ; T^r'_i, \text{ and } \tau' = \tau. \)
10) \( S_i = \ast \left[ \circ (l=1, \ldots, m) b_{\lambda}; \alpha_{\lambda} \to T_{\lambda} \right], \) and
\[ l = b_{\lambda} (\tau_i), \quad \alpha_{\lambda} = P_{j_i} ? x, \quad S_j = P_i ! y, \quad \text{and} \]
\[ S'_i = \alpha_{\lambda}; T_{\lambda}; S_i, \quad \text{and} \quad \tau' = \tau. \]

11) \( S_i = \ast \left[ \circ (l=1, \ldots, m) b_{\lambda}; \alpha_{\lambda} \to T_{\lambda} \right], \) and
\[ l = b_{\lambda} (\tau_i), \quad \alpha_{\lambda} = P_{j_i} ! x, \quad S_j = P_i ? y, \quad \text{and} \]
\[ S'_i = \alpha_{\lambda}; T_{\lambda}; S_i, \quad \text{and} \quad \tau' = \tau. \]

12) \( S_i = \ast \left[ \circ (l=1, \ldots, m) b_{\lambda}; \alpha_{\lambda} \to T_{\lambda} \right], \]
\[ S_j = \ast \left[ \circ (l' = 1, \ldots, m') b'_{\lambda'}; \alpha'_{\lambda'} \to T'_{\lambda'} \right], \]
\[ l = b_{\lambda} (\tau_i), \quad l = b'_{\lambda'} (\tau_j), \quad \alpha_{\lambda} = P_{j_i} ! x, \quad \alpha'_{\lambda'} = P_i ? y, \quad \text{and} \]
\[ S'_i = \alpha_{\lambda}; T_{\lambda}; S_i, \quad \text{and} \quad S'_j = \alpha'_{\lambda'}; T'_{\lambda'}; S_j, \quad \text{and} \quad \tau' = \tau. \]

13) Same as (12) except that \( \alpha_{\lambda} = P_{j_i} ? x, \) and \( \alpha'_{\lambda'} = P_i ! y. \)

14) \( S_i = \ast \left[ \circ (l=1, \ldots, m) b_{\lambda}; \alpha_{\lambda} \to T_{\lambda} \right], \]
\[ S_j = \ast \left[ \circ (l' = 1, \ldots, m') b'_{\lambda'}; \alpha'_{\lambda'} \to T'_{\lambda'} \right], \]
\[ l = b_{\lambda} (\tau_i), \quad l = b'_{\lambda'} (\tau_j), \quad \alpha_{\lambda} = P_{j_i} ! x, \quad \alpha'_{\lambda'} = P_i ? y, \quad \text{and} \]
\[ S'_i = \alpha_{\lambda}; T_{\lambda}; S_i, \quad \text{and} \quad S'_j = \alpha'_{\lambda'}; T'_{\lambda'}; S_j, \quad \text{and} \quad \tau' = \tau. \]

15) \( S_i = \ast \left[ \circ (l=1, \ldots, m) b_{\lambda}; \alpha_{\lambda} \to T_{\lambda} \right] \]
\[ m \]
\[ \text{and} \quad \tau' = \tau. \]

16) \( S_i = \ast \left[ \circ (l=1, \ldots, m) b_{\lambda}; \alpha_{\lambda} \to T_{\lambda} \right], \)
\[ m \]
\[ \text{and} \quad \tau' = \tau. \]
\[ \text{and for all } \lambda \text{ such that } l = b_{\lambda} (\tau_i), \quad S_c (\alpha_{\lambda}) = E, \quad \text{and} \]
\[ S'_i = E, \quad \text{and} \quad \tau'_i = \tau_i [h_i + h_i \vdash (i, T, t)], \text{ where } T = \{ k | 3 \leq r \leq m \}
\[ | b_{\lambda} (\tau_i) \land k = c (\alpha_{\lambda}) \}, \]
\[ \text{where } c (\alpha_{\lambda}) = j \text{ if } \alpha_{\lambda} = P_{j_i} ? x \text{ or } P_{j_i} ! x. \]

This case corresponds to the termination of an I/O guarded loop in \( S_i \) on account of the fact that all processes with which \( S_i \) is willing to communicate, have terminated.

17) \( S_i = \tilde{S}_i ; S \), and \( \tilde{S}_i \neq E, \) and \( (S_i \mathbin{;} \ldots \mathbin{;} \tilde{S}_i \mathbin{;} \ldots \mathbin{;} S_n, \tau) \to (S'_i \mathbin{;} \ldots \mathbin{;} S'_n, \tau'), \)
\[ \text{and} \]
\[ [\tilde{S}_i \neq E, S'_i = \tilde{S}_i ; S] \text{ or } [\tilde{S}_i = E, S'_i = S], \text{ and } S'_j = S'_j. \]

In other words, this clause says that if
\( (S_i \mathbin{;} \ldots \mathbin{;} S_i \mathbin{;} \ldots \mathbin{;} S_n, \tau) \to (S'_i \mathbin{;} \ldots \mathbin{;} S'_n, \tau'), \) \( S_i \neq E, \) then
\( (S_i \mathbin{;} \ldots \mathbin{;} S_i \mathbin{;} \ldots \mathbin{;} S_n, \tau) \to (S'_i \mathbin{;} \ldots \mathbin{;} S'_i \mathbin{;} \ldots \mathbin{;} S'_n, \tau'). \)
The restriction \( \bar{S}_i \neq E \) is needed to ensure the validity of clause (16). Moreover, we may also assume, without loss of generality, that \( \bar{S}_i \) is of the form \( \text{skip} \), or \( x := e \), or \( P_j \) \( x \), or \( P_j \) \( y \), or 
\( \alpha \beta \text{\text{-}T}^{l} \), or \( *[\alpha \beta \text{\text{-}T}^{l}] \), or \( *[\alpha \beta \text{\text{-}T}^{l}] \), or \( *[\alpha \beta \text{\text{-}T}^{l}] \).

We are using \( E;S = S \) to handle the situation when \( \bar{S}_i = E \) in the above clause. Clause 17 takes care of sequential composition.

Suppose now that we have a CSP program \( P : P //_{.} //_{.} P_{n} \). Suppose \( s^I = (s^I_1, \ldots, s^I_n) \) is some given initial state of \( P \). Thus \( s^I_i \) is the initial local state of \( P_i \). We shall, in fact, take \( \tau^I = (\tau^I_1, \ldots, \tau^I_n) \), where \( \tau^I_i = (s^I_i, \varepsilon) \) to be the initial state of \( P \). Recall that the second component of \( \tau^I_i \) is the communication sequence \( h_i \) of \( P_i \); at the state of \( P \), the communication sequences of all the processes are \( \varepsilon \), justifying our definition of \( \tau^I_i \).

Suppose there is some execution of \( P \), starting in the above initial state \( \tau^I \) and terminating in some final state \( \tau^F \), all processes having terminated. Then it should be clear from the definition of "\( \rightarrow \)" that there exists a sequence of pairs \( (S^1, \tau^1), \ldots, (S^u, \tau^u) \) such that \( S^1 = P_1 //_{.} //_{.} P_n, \tau^1 = \tau^I, S^u = E //_{.} //_{.} E, \tau^u = \tau^F \), and \( (S^w, \tau^w) \rightarrow (S^{w+1}, \tau^{w+1}) \) for all \( w, 1 \leq w \leq u \). The sequence \( (S^1, \tau^1), \ldots, (S^u, \tau^u) \) will be called the computation sequence.

Next suppose, we have proved, using the system of the last section the following: \( \{ \prod_{i=1}^{n} I_i \} P_1 //_{.} //_{.} P_n \{ [ \prod_{i=1}^{n} F_i ] \} [\text{Reduce}(h_1, \ldots, h_n) = (\varepsilon, \ldots, \varepsilon)] \). Let \( \text{pre}_i, \text{post}_i (i = 1, \ldots, n) \) be assertion functions for

\[
(I_i \wedge (h_i = \varepsilon) \wedge (\wedge_{j \neq i} h_j = \varepsilon) \wedge (\wedge_{j \neq i} C_{ji})) \text{pre}_i[P_i],
\]

\( C_{ji} (j, i = 1, \ldots, n, j \neq i) \) satisfying the conditions of the first two lines of R11. We shall define certain assertion functions for \( S^V (v = 1, \ldots, u) \). Recall that \( S^V \) is of the form \( S^1 //_{.} //_{.} S^V_n \). Thus the assertion functions associated with \( S^V \) will be a set of \( n \) pairs of functions \( \text{pre}_i \), \( \text{post}_i, i = 1, \ldots, n \). \( S \) is of the form \( S^1 //_{.} //_{.} S^V_n, S^V_1 = P_i \). Hence we define, \( \text{pre}_i = \text{pre}_i \), and \( \text{post}_i = \text{post}_i \). Thus \( \text{pre}_i, \text{post}_i \) are assertion functions for \( (I_i \wedge (h_i = \varepsilon) \wedge (\wedge_{j \neq i} h_j = \varepsilon) \wedge (\wedge_{j \neq i} C_{ji} )) S^1_i [P_i] \).

Next consider the assertion functions for \( S^V, v > 1 \). \( \text{pre}_i \), \( \text{post}_i \) will be defined in terms of \( \text{pre}_i^{v-1} \), \( \text{post}_i^{v-1} \). Before defining \( \text{pre}_i, \text{post}_i \), we need to take care of a minor point: in our original definition of assertion functions we did not allow for the possibility that the process may be of the form \( E \). We allow for this by adding the following clause to the definition of assertion functions: if \( P_i \) is \( P \), then \( \text{pre}(P_i) = \text{post}(P_i) \); and if \( P_i \) is \( E \), the only component of \( P_i \) is \( E \).
Now we are ready to define \( \text{pre}^V_i, \text{post}^V_i \).

1) For every component \( T \) of \( S^V_1 \), which is also a component of \( S^V_1 \),

\[
\text{pre}^V_i(T) = \text{pre}^V_i(T), \quad \text{post}^V_i(T) = \text{post}^V_i(T).
\]

2) If \( S^V_1 \neq E \) and \( S^V_1 = E \), then \( \text{pre}^V_i(S^V_1), \text{post}^V_i(S^V_1) = \text{post}^V_i(S^V_1) \).

3) If \( S^V_1 = [a_{(l=1,\ldots,m)}b \rightarrow T_1], \) and \( S^V_1 = T_1; S^V_1 \), then

\[
\text{pre}^V_i(S^V_1) = \text{pre}^V_i(T_1), \quad \text{post}^V_i(S^V_1) = \text{post}^V_i(S^V_1).
\]

4) If \( S^V_1 = [a_{(l=1,\ldots,m)}b \rightarrow T_2], \) and \( S^V_1 = a_r; T_2 \), then

\[
\text{pre}^V_i(S^V_1) = \text{pre}^V_i(a_r), \quad \text{post}^V_i(S^V_1) = \text{post}^V_i(T_2).
\]

5) If \( S^V_1 = [a_{(l=1,\ldots,m)}b \rightarrow T_3], \) and \( S^V_1 = a_r; T_1; S^V_1 \), then

\[
\text{pre}^V_i(S^V_1) = \text{pre}^V_i(a_r), \quad \text{post}^V_i(S^V_1) = \text{post}^V_i(S^V_1).
\]

(1) through (5) take care of all clauses in the definition of "->" except for the last clause.

6) Suppose \( S^V_1 = [S^V_1; \ldots; S^V_n], \) \( S^V_1 = T_1, T_2 \) where \( T_1 \neq E \). Suppose also that \( [S^V_1; \ldots; T_i; \ldots; S^V_n] \rightarrow [S^V_1; \ldots; T_i; \ldots; S^V_n] \). Then

\[
[S^V_1; \ldots; T_i, T_2; \ldots; S^V_n] \rightarrow [S^V_1; \ldots; T_i, T_2; \ldots; S^V_n], \text{assuming } T_i \neq E.
\]

Denote \( [S^V_1; \ldots; T_i; \ldots; S^V_n] \) by \( S^V_1 \) and \( [S^V_1; \ldots; T_i; \ldots; S^V_n] \) by \( S^V_1 \).

Given assertion functions \( \text{pre}^V_j, \text{post}^V_j \) for \( S^V_1 \), define assertion functions \( \overline{\text{pre}}^V_j, \overline{\text{post}}^V_j \) for \( S^V_1 \) as follows:

\[
\overline{\text{pre}}^V_j = \text{pre}^V_j, \quad \overline{\text{post}}^V_j = \text{post}^V_j \quad \text{for all } j \neq i; \text{ also every component of } T \text{ is a component of } S^V_1.
\]

Thus define,

\[
\text{pre}^V_i(T) = \overline{\text{pre}}^V_i(T), \quad \text{post}^V_i(T) = \overline{\text{post}}^V_i(T) \quad \text{for every component } T \text{ of } T_1.
\]

Define functions \( \overline{\text{pre}}^V_j, \overline{\text{post}}^V_j \) for \( S^V \) in terms of the functions \( \overline{\text{pre}}^V_j, \overline{\text{post}}^V_j \) of \( S^V_1 \) using clauses (1) through (5) just specified.

Then define \( \overline{\text{pre}}^V_j, \overline{\text{post}}^V_j \) as follows:

For every component \( T \) of \( T_1 \),

\[
\overline{\text{pre}}^V_j(T) = \text{pre}^V_j(T), \quad \overline{\text{post}}^V_j(T) = \text{post}^V_j(T);
\]

for every component \( T \) of \( T_2 \),

\[
\overline{\text{pre}}^V_i(T) = \text{pre}^V_i(T), \quad \overline{\text{post}}^V_i(T) = \text{post}^V_i(T);
\]

\[
\overline{\text{pre}}^V_i(T_1; T_2) = \text{pre}^V_i(T_1), \quad \overline{\text{post}}^V_i(T_1; T_2) = \text{post}^V_i(T_2).
\]

Finally, we have to consider the case when \( T_1 = E \), so that \( S^V_i = [S^V_1; \ldots; T_2; \ldots; S^V_n] \). In this case we simply omit the clause "for every component \( T \) of \( T_1 \) ..." above; also \( T_1; T_2 \) should be replaced by \( T_2 \).
This completes the definition of $\text{pre}_i^v, \text{post}_i^v$. The following results can be shown more or less directly (if somewhat tediously) from the definitions:

a) $\text{post}_i^v(S_i^v) = \text{post}_{i-1}^{v-1}(S_i^{v-1})$ for all $i = 1, \ldots, n$, $v = 2, \ldots, u$.

b) $\text{pre}_i^v, \text{post}_i^v$ are assertion functions for $\{\text{pre}_i^v(S_i^v)\} S_i^v \{\text{post}_i^v(S_i^v)\}$.

What we shall try to show next is that $\tau_i^v$, the state of the program after $v-1$ steps satisfies the predicate,

(A) $\left[ \bigwedge_{i=1}^{n} \text{pre}_i^u(E) \right] \land \left[ \text{Reduce}(h_1, \ldots, h_n) = (\epsilon, \ldots, \epsilon) \right]$.

This will prove that $\tau_i^u$, the final state satisfies

$\left[ \bigwedge_{i=1}^{n} \text{pre}_i^u(E) \right] \land \left[ \text{Reduce}(h_1, \ldots, h_n) = (\epsilon, \ldots, \epsilon) \right]$, i.e.

$\left[ \bigwedge_{i=1}^{n} \text{post}_i^u(E) \right] \land \left[ \text{Reduce}(h_1, \ldots, h_n) = (\epsilon, \ldots, \epsilon) \right]$, (since $\text{pre}(E) \Rightarrow \text{post}(E)$)

i.e. $\left[ \bigwedge_{i=1}^{n} \text{post}(P_i) \right] \land \left[ \text{Reduce}(h_1, \ldots, h_n) = (\epsilon, \ldots, \epsilon) \right]$, thus proving the consistency of the system.

Before we can prove (A), we need to face another problem: $\tau_i^v$ has two components, $s_i^v$, the (current) local state of $P_i$, and $h_i^v$, the communication sequence of $P_i$, whereas $\text{pre}_i^v(S_i^v)$ is a predicate on $s_i^v$, which has the following components: the local state of $P_i, s_i^v$; the communication sequence $h_i^v, h_i^v(j = 1, \ldots, n, j \neq i)$; and $h_i^v(j = 1, \ldots, n, j \neq i)$. Thus we need to add to $\tau_i^v$, appropriate values for $h_i^v$ and $h_i^v$, before we can hope to prove (A). Recall that $h_i^v$ is the sequence from $P_j$ that $P_i$ has consumed, which, as pointed out in the discussion in section 2 following the rule of inference Rule 11 for parallel composition, is identical to $h_i^v/\{(j, i), (i, j)\}$, the sequence that $P_j$ has sent to $P_i$; and $h_i^v$ is just the second component of $\tau_i^v$. On the other hand $h_i^v$ is an entirely mythical variable, introduced by the proof system. Thus what we shall show is that there exists some $h_i^v(j \neq i)$ such that $\tau_i^v, h_i^v$ and $h_i^v$ together satisfy the predicate (A). This will prove that there exists some $h_i^v$, such that $\tau_i^v$ and $h_i^v$ together satisfy

$\left[ \bigwedge_{i=1}^{n} \text{post}(P_i) \right] \land \left[ \text{Reduce}(h_1, \ldots, h_n) = (\epsilon, \ldots, \epsilon) \right]$

showing that $\tau_i^v$ is consistent with the final post-condition proved using the axiomatic system.

Next, we formalize the notation we have been using so far. $\tau_i^v$ is the state of the program after execution of $v-1$ steps of the computation sequence. $\tau_i^v$, the $i$th component of $\tau_i^v$, is the state of the $i$th process after execution of $v-1$ steps of the computation sequence.
\( \tau_i \) consists of two components - \( s_i^V \) and \( h_i^V \). We shall take \( h_{ji}^V(j \neq i) \) to be equal to \( h_j^V / \{(j,i),(i,j)\} \). We then need to show that there exists \\
\( h_{ji}^V(j \neq i) \) such that \\
\( \Sigma \) \( \text{pre}^V_i \left( S_i^V, [\sigma_i + \sigma_j^V] \wedge [\text{Reduce}(h_1^V, \ldots, h_n^V) = (\varepsilon, \ldots, \varepsilon)] \right) \right) \right) \] \\
where \( \sigma_i^V = (s_i^V, h_i^V, h_j^V(j = i), h_j^V(j = i)) \), and \( \sigma_i + \sigma_j^V \) denotes "replace \( \sigma_i \) by \( \sigma_j^V \)." \\

First suppose \( v = 1 \). Then \( \text{pre}_1^V(S_1^V) = \text{pre}(P_1) \); and \( \text{pre}(P_1) \) is \\
\( I_i \wedge (h_i = \varepsilon) \wedge (\Lambda j \neq i, h_j = \varepsilon) \wedge (\Lambda j \neq i, C_{ji}) \) . . . (3) \\

Also \( \tau_i^1 = I_i \), \( \tau_i^1 \) being \( (s_i^1, \varepsilon) \), and \( s_i^1 \) is such that \( \models I_i \left[ s_i^1 + s_i^1 \right] \).

Thus the first two clauses of (3) are satisfied by \( \tau_i^1 \), and hence by \( \tau_i^1 \). The third clause is also satisfied since \( h_j^1 = \varepsilon \), and hence \( h_{ji}^1 = \frac{h_j^1}{\{(j,i),(i,j)\}} = \varepsilon \). Moreover, since \( C_{ji} \) satisfies the first line of rule R11, there clearly exists some \( h_{ji}^1 \) such that \( \models C_{ji} \left[ h_{ji}^1 + h_{ji}^1 \right] \).

This proves the required result in the case of \( v = 1 \). (The clause \( \text{Reduce}(h_1^V, \ldots, h_n^V) = (\varepsilon, \ldots, \varepsilon) \) is trivially true since \( h_i^V = \varepsilon, i = 1, \ldots, n \).)

Next, suppose \( v > 1 \). We shall denote \( v - 1 \) by \( w \). Then we must show that if there exist \( h_{ji}^w(j, i = 1, \ldots, n, j \neq i) \) such that \\
\( \Lambda \text{pre}_i^w(S_i^w)[\sigma_i + \sigma_j^w] \wedge [\text{Reduce}(h_1^w, \ldots, h_n^w) = (\varepsilon, \ldots, \varepsilon)] \) . . . (4) \\
is true, then there exist \( h_{ji}^w(j, i = 1, \ldots, n, j \neq i) \) such that \\
\( \Lambda \text{pre}_i^w(S_i^w)[\sigma_i + \sigma_j^w] \wedge [\text{Reduce}(h_1^w, \ldots, h_n^w) = (\varepsilon, \ldots, \varepsilon)] \) . . . (5) \\
is true. Moreover, we have \( (S_1^w / \ldots / S_n^w, \tau^w) \rightarrow (S_1^w / \ldots / S_n^w, \tau^w) \). Thus we have to prove the required result (5), assuming (4) for each of the seventeen clauses defining the relation "$\rightarrow$". We shall write \( (S^w, \tau^w) \rightarrow_d (S^v, \tau^v) \) to indicate that \( (S^w, \tau^w) \) and \( (S^v, \tau^v) \) are related according to the clause numbered "d" in the definition of "$\rightarrow$". Thus \( (S^w \tau^w) \rightarrow_d (S^v, \tau^v) \) indicates that \\
\( S_i^w = \text{skip}, S_i^v = E, S_j^w = S_j^v \) for all \( j \neq i \), and \( \tau^v = \tau^w \).

We first consider the clause \( [\text{Reduce}(h_1^w, \ldots, h_n^w) = (\varepsilon, \ldots, \varepsilon)] \). 

A look at the different clauses defining "$\rightarrow$", shows that \( (S^w, \tau^w) \rightarrow_d (S^v, \tau^v) \) implies that \( (h_1^w, \ldots, h_n^w) = (h_1^w, \ldots, h_n^w) \), unless \( d \) is either 6 or 16. Thus, if \( d \) is not 6 or 16, the second clause of (4) gives us \\
\( \text{Reduce}(h_1^w, \ldots, h_n^w) = (\varepsilon, \ldots, \varepsilon) \) i.e. \( \text{Reduce}(h_i^w, \ldots, h_n^w) = (\varepsilon, \ldots, \varepsilon) \). If \( d \) is 6, then there exist \( i, j \) such that \\
\( h_k^w = h_k^w \) for all \( k \neq i, j \); \( h_i^w = h_i^w \) \( - \{(j,i),(i,j)\} \); \( h_j^w = h_j^w \) \( - \{(j,i),(i,j)\} \) for some \( m \).
Hence \( \text{Reduce}(h^V_1, \ldots, h^V_n) = \text{Reduce}(h^W_1, \ldots, h^W_i \vdash (j, i, m), \ldots, h^W_j \vdash (j, i, m), \ldots, h^W_n) = \text{Reduce}(\varepsilon, \ldots, (j, i, m), \ldots, (j, i, m), \ldots, \varepsilon) = (\varepsilon, \ldots, \varepsilon). \)

(Note: The above proof is valid only if for all \( k \neq i, j \), \((k, T, t)\) is an element of \( h^W_k \) implies \( i, j \notin T \); this is, indeed, true since \( S^1_i \ldots S^w_i \) and \( S^1_j \ldots S^w_j \) are all different from \( E \)).

The analogous proof in the case \( d = 16 \) is left to the reader.

This verifies that \([\text{Reduce}(h^V_1, \ldots, h^V_n) = (\varepsilon, \ldots, \varepsilon)] \) implies \([\text{Reduce}(h^V_1, \ldots, h^V_n) = (\varepsilon, \ldots, \varepsilon)] \). Next we must consider the clause \[ \bigwedge_{i=1}^{n} \text{pre}^V_i(S^V_i)[\sigma_i \vdash \sigma_i^V] \] of (5).

1. Suppose \((S^w, \tau^w) \rightarrow (S^v, \tau^v)\). Then \( S^v_i = \text{skip}, S^v_j = E, S^v_j = S^v_j \) for all \( j \neq i \), and \( \tau^v = \tau^w \). Moreover, \( \text{pre}^V_i(S^v_i) = \text{post}^w_i(S^w_i) \), and since \( S^v_i = \text{skip}, \text{pre}^w_i(S^w_i) = \text{post}^w_i(S^w_i) \); and \( \text{pre}^V_j(S^v_j) = \text{pre}^w_j(S^w_j) \) for all \( j \neq i \).

Let \( h^V_{jk} = h^V_{jk} \) for all \( j, k \neq 1, \ldots, n \) \( j = k \). Then, clearly

\[ \bigwedge_{j=1}^{n} \text{pre}^w_j(S^w_j)[\sigma_j \vdash \sigma_j^w] \rightarrow \bigwedge_{j=1}^{n} \text{pre}^V_j(S^v_j)[\sigma_j \vdash \sigma_j^V]. \]

2. The proofs in the cases \((S^w, \tau^w) \rightarrow (S^v, \tau^v)\), in the cases \( d = 2, 3, 4, 5, 7, 8, 9, 15 \) are equally straightforward and are left to the reader.

3. The remaining cases involve communication between processes, and are more difficult to deal with. Let us first consider the case when \( d = 6 \) i.e. \( S^w_i = P_j (x, S^w_j = P_j (y, S^v_i = E, S^v_j = E, \)

\[ \tau^v_i = \tau^w_i [x + \tau^v_j(y), h_i + h_j \vdash (j, i), \tau^w_j) \], \tau^v_j = \tau^w_j [h_i + h_j \vdash (j, i), \tau^w_j(y)]. \]

It is trivial to see that \( \text{pre}^w_k(S^w_k)[\sigma_k \vdash \sigma_k^w] \rightarrow \text{pre}^V_k(S^v_k)[\sigma_k \vdash \sigma_k^v] \) for all \( k \neq i, j \),

since \( S^v_k = S^w_k \) and \( \tau^v_k = \tau^w_k \).

Next, we shall show \( \text{pre}^w_j(S^w_j)[\sigma_j \vdash \sigma_j^w] \rightarrow \text{pre}^V_j(S^v_j)[\sigma_j \vdash \sigma_j^v] \):

Since \( S^w_j = P_j (y, \text{pre}^w_j(S^w_j) \) must satisfy the requirement

\[ \text{pre}^w_j(S^w_j) = \exists h^w_{ij} \cdot [\text{pre}^w_j(S^w_j)[h^w_{ij} + \overline{h^w_{ij}}] \wedge [D(h^w_{ij}) = (j, i)] \wedge [V(h^w_{ij}) = y]]. \]

Hence, we have,

\[ \exists h^w_{ij} \cdot \text{pre}^w_j(S^w_j)[\sigma_j \vdash \sigma_j^w] \rightarrow \exists h^w_{ij} \cdot [\text{pre}^w_j(S^w_j)[\sigma_j \vdash \sigma_j^w] \wedge [D(h^w_{ij}) = (j, i)] \wedge [V(h^w_{ij}) = y]] \]

where \( \sigma_j^w \) is essentially \((\tau^w_j, h^w_{ij})\) and \( \overline{\sigma}_j^w \) is essentially \((\tau^w_j, \overline{h^w}_{ij})\).

(In fact, of course, \( \sigma_j^w \) and \( \overline{\sigma}_j^w \) contain other components such as \( h^w_{kj}, k = 1, \ldots, n, k \neq j \).)
Also \( \tau_j^V = \tau_j^W[h_j + h_j^\bot - (j,i,\tau_j^W(y))] \); and, since \( S_j^W = P_i' y \),
\[
[\text{pre}_{j}^W(S_j^W) \wedge [D(h'_{ij}) = (i,j)] \wedge [V(h'_{ij}) = y]] \Rightarrow \text{post}_{j}^W(S_j^W)[h_j + h_j^\bot - (j,i,\tau_j^W(y)), h_{ij}^l + \text{rest}(h_{ij}^w)];
\]
and \( \text{post}_{j}^W(S_j^W) = \text{pre}_{j}^V(S_j^V) \); thus, if we take \( \sigma_j^V = (\tau_j^V, \text{rest}(h_{ij}^w)) \), \( h_{ij}^w \) being from line (6), then, we have shown that there exists \( h_{ij}^w \) such that \( \text{pre}_{j}^V(S_j^V)[\sigma_j^V] \).

Finally, we must show that \( \text{pre}_{i}^V(S_i^V)[\sigma_i^V + \sigma_i^V] \) also holds, for an appropriate \( \sigma_i^V \). In order to do this, we need to make use of two results: a) There exists a sequence \( h_j \) such that
\[
[h_j^V, h_j^l] \wedge [\exists S_j, h_{ij}^l, h_{ij}^l \cdot \text{post}(P_j)[\sigma_j + \sigma_j]]
\]
where \( \sigma_j = (\tau_j, h_j^l, h_{ij}^l, h_{ij}^l) \). This follows from the fact that we have shown that there exists \( h_{ij}^l \) such that
\[
\text{pre}_{j}^V(S_j^V)[\sigma_j + \sigma_j^V], \text{ where } \sigma_j^V = (\tau_j^V, h_{ij}^l, h_{ij}^l),
\]
and the remarks at the end of the last section.

b) if \( h_{ji}^l \) is such that \( C_{ji}[h_{ji} + h_{ji}^l] \), and \( h_{ji}^l = h_{ji}^l \), and there exists \( h_{ji}^l \) such that \( \text{pre}_{i}^W(S_i^W)[\sigma_i + \sigma_i^W] \), where \( \sigma_i^W = (\tau_i^W, h_{ji}^l, h_{ji}^l) \), then \( \text{pre}_{i}^W(S_i^W)[\sigma_i + \sigma_i^W] \), where \( \sigma_i^W = (\tau_i^W, h_{ji}^l, h_{ji}^l) \).

We shall prove this result towards the end of this section.

Using results (a) and (b), we may assume that there exists \( h_{ji}^l \) such that \( \text{first}(h_{ji}^l) = (j,i,\tau_j^W(y)) \), and
\[
\text{pre}_{i}^W(S_i^W)[\sigma_i + \sigma_i^W], \text{ where } \sigma_i^W = (\tau_i^W, h_{ji}^l, h_{ji}^l).
\]
This implies, by the axiom for \( P_i? x \), that the following predicate is true: \( \text{post}_{i}^W(S_i^W)[\sigma_i + \sigma_i^W] \), where \( \sigma_i^W = (\tau_i^W, h_{ji}^l, \text{rest}(h_{ji}^w)) \), and \( \text{pre}_{i}^V(S_i^V) = \text{post}_{i}^W(S_i^W) \).

Thus we have shown that there exists an \( h_{ji}^V = \text{rest}(h_{ji}^w) \), such that \( \text{pre}_{i}^V(S_i^V)[\sigma_i + \sigma_i^V] \), \( \sigma_i^V = (\tau_i^V, h_{ji}^l, h_{ji}^l) \).

4) Next, consider the case \( d = 10 \). Thus,
\[
S_i^W = *[{\alpha}_{l=1\ldots m} h_l; \alpha \rightarrow T^l], \quad S_j^W = P_i' y, \quad l = b_r(\tau_1^W), \alpha_r = P_j ? x, \quad \text{and} \quad S_i^V = \alpha_r; T^l; S_i^W.
\]
we have, \( \exists h_{ji}^w \cdot \text{pre}_{j}^W(S_j^W)[\sigma_j + \sigma_j^W], \quad \sigma_j^W = (\tau_j^W, h_{ji}^l, h_{ji}^l). \)
Since \( S_j^W = P_i' y \), we may assume \( \text{first}(h_{ji}^w) = (j,i,\tau_j^W(y)) \).
Then, from the conditions satisfied by $\text{pre}^w_j$, we have

$$\text{post}^w_j(s^w_j[(\sigma^w_j + \tilde{\sigma}^w_j)], \tilde{\sigma}^w_j = (s^w_j, h^w_j(j,i, r^w_j(y)), h^w_j(j,i, r^w_j(y)), \tilde{h}^w_{ij})$$

where $\tilde{h}^w_{ij} = \text{rest}(h^w_{ij})$.

Hence, by the remark at the end of the last section, we may assume that there exists $\tilde{h}^w_{ij}$ such that

$$[\exists \tilde{h}^w_{ij} . h^w_j(j,i, r^w_j(y)) \vdash \tilde{h}^w_{ij} = \tilde{h}^w_{ij}] \wedge C_{j,i} [\tilde{h}^w_{ji} + \tilde{h}^w_{ij}]$$

Then, by the result (b) above, we have,

$$\text{pre}^v_i(S^v_i)[\sigma^w_i + \sigma^v_i], \text{ where } \sigma^v_i = (\tau^v_i, h^v_{ij}(j,i, r^v_j(y)), \tilde{h}^v_{ij})$$

This, in conjunction with $\models b_{\tau^v_i}$ gives us $\text{pre}^v_i(T^v_i)[\sigma^w_i + \sigma^v_i]$, which gives us $\text{pre}^v_i(S^v_i)[\sigma^w_i + \sigma^v_i]$.

The remaining cases may be dealt with in essentially similar fashion. Thus we have shown that for all $v, 1 \leq v \leq u$, there exist $h^v_j(j,k=1,\ldots,n, j \neq k)$, such that the following predicate is true: $[\bigwedge_{i=1}^n \text{pre}^v_i(S^v_i)[\sigma^w_i + \sigma^v_i]]$

[Reduce($h^v_1,\ldots,h^v_n$) = $(\epsilon,\ldots,\epsilon)$], where $\sigma^v_i = (\tau^v_i, h^v_{ij}, h^v_{ji})$, where $h^v_{ij} = h^v_{ji}/(i,j),(j,i)$]. Thus, in particular, taking $v = u$, we have shown that there exists $h^u_{jk}$ such that

$$\bigwedge_{i=1}^n \text{pre}^u_i(S^u_i)[\sigma^w_i + \sigma^u_i] \wedge [\text{Reduce}(h^u_1,\ldots,h^u_n) = (\epsilon,\ldots,\epsilon)]$$

is true; and since, $S^u_i \equiv E$ for all $i = 1,\ldots,n$, and hence $\text{pre}^u_i(S^u_i) = \text{post}^u_i(S^u_i)$, and $\text{post}^v_i(S^v_i) = \text{post}(P_i) = F_i$ for all $i = 1,\ldots,n$, and $v = 1,\ldots,u$ we have shown that the following predicate is true:

$$[\bigwedge_{i=1}^n F_i[\sigma^w_i + \sigma^u_i] \wedge [\text{Reduce}(h^1,\ldots) = (\epsilon,\ldots)],$$

thus proving the consistency of the axiomatic system.

Two further remarks are in order: 1) In the above discussion, we have not considered the case $d = 17$. The reason for this is the following: we may consider clause 17 in the definition of $\rightarrow$ as an abbreviation of several other clauses, each of which will be quite similar to one of the other sixteen clauses. An example of such a clause would be:

$$S_i = [x := e; S_i]; \quad S_i' = S_i; \quad S_j' = S_j \text{ for all } j \neq i; \quad \tau'_i = \tau_i[x := e]; \quad \tau'_j = \tau_j$$

Another example would be:

$$S_i = [\eta(\epsilon = 1,\ldots,m) b_j; \sigma^w_j \rightarrow \tau^w_j]; S_i, \quad \text{ and}$$

$$\models b_{\tau_i}, \sigma^w_i = P_j(x), S_j = P_i \vdash e, S_j' = S_j, \text{ and}$$

$$S_i' = S_i; T^r_i; S_i' \text{ and } S_k' = S_k \text{ for all } k \neq i, \text{ and } \tau' = \tau.$$
Each of these cases may be handled in exactly the same fashion as the corresponding case among the first sixteen clauses. Note also that we do not need to consider the case where

\[ S_i = [T_i; T'_i]; S'_i, \]  

since this may be rewritten as \( T_i; [T'_i; S'_i] \).

2) Finally, we still need to verify the result (b) stated earlier in the section and used extensively in the proof of consistency. Recall that the result states the following: if for some \( i, j \), there exists \( h'_j \) such that \( C_{ji} [h'_j + \bar{h}'_j] \) is true, and \( h'_j \vdash \bar{h}'_j = \bar{h}'_j \), where \( h'_j = h'_j / ((i, j), (j, i)) \), and there exist \( h'_j[k, =1, \ldots, n, k \neq \ell] \) such that

\[
\forall m \leq n \quad \text{pre}^V_m (S^V_m [\sigma_m + \bar{\sigma}_m]) \text{ is true, where } \sigma_m = (\tau_m)^V, h'_k, \bar{h}'_k, \text{ then } \forall m \leq n \quad \text{pre}^V_m (S^V_m [\sigma_m + \bar{\sigma}_m]) \text{ is also true, where } \bar{\sigma}_m = \sigma_m \text{ for all } m \neq i, \text{ and } \bar{\sigma}_i = \sigma_i [h'_j + \bar{h}'_j].
\]

The result will be proved by induction on \( v \). The result is trivially true if \( v = 1 \), since \( \text{pre}^V_1 (S^V_1) = I_1 \wedge (h_1 = \varepsilon) \wedge (\wedge_{k \neq m} k_m) \) so that any \( \bar{\sigma}_i = (\tau_i, h'_k, = \varepsilon, \bar{h}'_k) \), where \( C_{ki} [h'_k + \bar{h}'_k] \), satisfies \( \text{pre}^V_1 (S^V_1) \). Next suppose the result is true for \( v < 1 \). (Again, we shall denote \( v = 1 \) by \( \omega \), and \( (S^W, \tau^W) \rightarrow (S^V, \tau^V) \). If \( S^W = S^V_i \), i.e. the progress in the computation took place in some process other than \( i \), the result is again trivial, since \( \text{pre}^V_1 (S^V_i) = \text{pre}^W_1 (S^W_i) \), and \( \tau_i = \tau_i^W \), and \( h'_k = h'_k \) for all \( k \neq i \).

Next, suppose \( S^W = x = e, S^V = E, \tau_i = \tau_i^W (x + e) \). Then \( h'_i = h'_i^W, h'_k = h'_k^W \) for all \( k \neq i \). Then, since the result is true for \( w \), we may assume that \( \text{pre}^V_1 (S^V_i) [\sigma_i + \bar{\sigma}_i] \) is true, where \( \sigma_i = (\tau_i^w, h'_i, h'_w, \bar{h}'_i) \), where \( h'_j = h'_j^w \). This implies \( \text{pre}^V_1 (S^V_i) [\sigma_i + \bar{\sigma}_i] \), where \( \sigma_i = (\tau_i^w, h'_i, h'_w, \bar{h}'_i) \), which is the required result. All other cases not involving communication between \( P_i \) and \( P_j \) may be treated similarly.

Finally, suppose \( S^W = P_1 \wedge x, S^W_j = P_j \wedge y, S^V = E, S^V_j = E, \tau_i = \tau_i^w (x + \tau_j^w (y), h'_i, h'_w) \), \( \tau_j = \tau_j^w (h'_j, h'_w) \). We also have \( h'_ji = h'_j^w \), \( h'_ji = h'_j^w \). Thus, by the inductive hypothesis the following predicate is true:

\[
\text{pre}^V_1 (S^V_i) [\sigma_i + \bar{\sigma}_i], \quad \text{where } \bar{\sigma}_i = (\tau_i^w, h'_i, h'_w, h'_j, = (j, i, \tau_j^w (y))) \text{ and } \bar{\sigma}_i = (\tau_i^w, h'_i, h'_w, h'_j, = (j, i, \tau_j^w (y))).
\]

This gives, post^V_1 (S^V_i) [\sigma_i + \bar{\sigma}_i], \( \bar{\sigma}_i = (\tau_i^w, h'_i, h'_w, h'_j, = (j, i, \tau_j^w (y))) \), \( h'_j = \bar{h}'_j \), which, post^W_1 (S^V_i) being identical to \( \text{pre}^V_1 (S^V_i) \), is the required result. The other cases involving communication are dealt with similarly.

Thus we have proved result (b), and hence the consistency of the axiomatic system.
4. Completeness of the axiomatic system.

This section is divided into three subsections: in 4.1 we introduce a new axiomatic system, and show that any proof in the new system can be converted into an equivalent proof in the system of section 2. In 4.2 we define a new operational semantics for CSP and show that it is equivalent to the operational semantics defined in section 3. In 4.3 we prove that the new axiomatic system is complete with respect to the operational semantics of section 4.2, thus proving the completeness of the system of section 2.

4.1. A new axiomatic semantics for CSP

Again, consider a CSP program $P_1//...//P_n$, $P_1,...,P_n$ being the communicating sequential processes. $h_1$ will denote, as before, the communication sequence of process $P_i$. $h_1,...,h_n$ will be the only sequences we shall deal with in the new axiomatic semantics. For ease of reference, we shall refer to the system of section 2 by the name SYS1, the system to be introduced in this section being named SYS2.

As in section 2, the axioms and rules of inference for constructs appearing in the individual processes will be stated in a form applicable to $P_i$.

A1. Input

$$\{p'\}P_j?x\{p\}$$

where

$$p' = \forall z. P_z, h_1 \vdash (j, i, z)$$

The universal quantification is over the set of all integers, since we assume that only integers are being communicated. The reason for the universal quantification is that we have no information on what $P_j$ will send to $P_i$, and hence must consider all possible values.

A2. Output

$$\{P_{h_1} \vdash (i, j, y)\} P_j!y\{p\}.$$ 

A3. Assignment

$$\{p^x_e\}x := e\{p\}$$

A4. Skip

$$\{p\} skip\{p\}$$
R1. **Boolean guarded selection**

\[
(p \land b_r) \Rightarrow S_r(q), \ r = 1, \ldots, m
\]

\[
\{p\}[\alpha(\ell = 1, \ldots, m)b_{\ell} \Rightarrow S_{\ell}]{q}
\]

R2. **Boolean guarded repetition**

\[
(p \land b_r) \Rightarrow S_r(p), \ r = 1, \ldots, m
\]

\[
\{p\}[^{\alpha(\ell = 1, \ldots, m)b_{\ell} \Rightarrow S_{\ell}}]{p \land \bigwedge_{\ell = 1}^{m} \alpha_{\ell}}{q}
\]

R3. **I/O guarded selection**

\[
(p \land b_r) \alpha_r; S_r(q), \ r = 1, \ldots, m
\]

\[
\{p\}[\alpha(\ell = 1, \ldots, m)b_{\ell} \Rightarrow \alpha_{\ell} \Rightarrow S_{\ell}]{q}
\]

R4. **I/O guarded repetition**

\[
(p \land b) \Rightarrow q
\]

\[
(p \land b) \Rightarrow q_{h_i} \mid -(i, \{j | \exists r \leq m. [b_r \land c(\alpha_r) = j] \}, t)
\]

\[
(p \land b_r) \alpha_r; S_r(p), \ r = 1, \ldots, m
\]

\[
\{p\}[^{\alpha(\ell = 1, \ldots, m)b_{\ell} \Rightarrow \alpha_{\ell} \Rightarrow S_{\ell}}]{q}
\]

where \(b = \bigvee_{\ell = 1}^{m} b_{\ell}\), and \(c(\alpha_r) = j\) if \(\alpha_r\) is \(P_j?x\) or \(P_j'y\).

R5. **Sequential composition**

\[
\{p\} S_1(q'), \{q'\} S_2(q)
\]

\[
\{p\} S_1; S_2(q)
\]

R6. **Consequence**

\[
p \Rightarrow p', \{p'\} S(q'), \ q' \Rightarrow q
\]

\[
\{p\} S(q)
\]

R7. **Conjunction**

\[
\{p\} S(q_1), \{p\} S(q_2)
\]

\[
\{p\} S(q_1 \land q_2)
\]

R8. **Disjunction**

\[
\{p_1\} S(q), \{p_2\} S(q)
\]

\[
\{p_1 \lor p_2\} S(q)
\]
R9. Parallel composition

\[ \{ I_i \land (h_i = \epsilon) \} P_i \{ F_i \}, \ i = 1, \ldots, n \]

\[ \{ \bigwedge_{j=1}^{n} I_j \} [P_1 / \ldots / P_n \{ \bigwedge_{j=1}^{n} F_j \land [\text{Reduce}(h_1, \ldots, h_n) = (\epsilon, \ldots, \epsilon)] \} \}

where \( I_i \) is a predicate on the local state \( s_i \).

Next, we must show that, given a proof of

\[ \{ \bigwedge_{j=1}^{n} I_j \} [P_1 / \ldots / P_n \{ \bigwedge_{j=1}^{n} F_j \land [\text{Reduce}(h_1, \ldots, h_n) = (\epsilon, \ldots, \epsilon)] \} \ldots (7) \]

in the above system, we can find an equivalent proof in the system SYS1 of section 2. We may assume that the proof of (7) is given by giving the proofs of

\[ (\Pi) \{ I_i \land (h_i = \epsilon) \} P_i \{ F_i \}, \ i = 1, \ldots, n \quad \ldots (8) \]

(The (\Pi) denotes that the proof is based on SYS2. In general, we shall write (\Pi) \{p\}S\{q\} to denote that \{p\}S\{q\} can be proved using SYS2. A similar notation will be used for SYS1). If we can show that

\( (\Pi) \{p\}S\{q\} \Rightarrow (I)\{p\}S\{q\} \), then our problem would be trivially solvable; however, the above implication is not true for arbitrary constructs \( S \) that may appear in \( P_i \), and we have to modify our approach. What we shall show is the following:

\[ (\Pi)\{p\}S\{q\} \Rightarrow (I)\{pv R_i\} S \{q \lor R_i\} \quad \ldots (9) \]

where \( R_i = \bigwedge_{j \neq i} [f(h_{j_1}) = g(h_i, i, j)+1], \)

where \( f(h_{j_1}) = 0 \), if \( h_{j_1} = \epsilon \)

\[ f(\text{rest}(h_{j_1})), \text{if} \ \text{first}(h_{j_1}) = (j, i, t) \]

\[ f(\text{rest}(h_{j_1}))+1, \text{if} \ \text{first}(h_{j_1}) = (j, i, m) \text{ or} \ \text{first}(h_{j_1}) = (i, j, m) \text{ for some } m. \]

and,

\[ g(h_i, i, j) = 0, \text{if} \ h_i = \epsilon: \]

\[ g(\text{rest}(h_i))+1, \text{if} \ \text{first}(h_i) = (j, i, m) \text{ or} \ (i, j, m) \text{ for some } m \]

\[ g(\text{rest}(h_i)) \text{ otherwise.} \]

We shall prove (9) by induction on the structure of \( S \). Suppose, \( S = \text{skip} \)

Then, \( (\Pi) \{p\}S\{q\} \Rightarrow [p \Rightarrow q] \Rightarrow (I)\{p\}S\{q\} \). Also, we have \( R_i \Rightarrow R_i \), and hence \( (I) \{R_i\}S\{R_i\} \); using the rules of disjunction and consequence, this gives, \( (\Pi) \{p\}\text{skip}\{q\} \Rightarrow (I)\{p \lor R_i\}\text{skip} \{q \lor R_i\} \). If \( S = x := e \), the result follows in similar fashion, using the fact \( R_i^x_{\text{e}} = R_i \). Next, suppose
S = P_j : y. Then, from the axiom for output, we have, (II) \{P_j : y \{q\} \Rightarrow [p \Rightarrow q_{h_i}^h \{i, j, y\}]\}. In order to be able to prove (I) \{p \lor R_i \} \lor y \{q \lor R_i \}, we must have the following:

\[ [p \lor R_i] = \exists h_i'. [[p \lor R_i] \lor [D(h_i') = (i, j)] \land [V(h_i') = y]] \ldots (10) \]

and, \[[p \lor R_i] \land [D(h_i') = (i, j)] \land [V(h_i') = y] = [q \lor R_i] \lor [h_i' \lor (i, j, y), h_i' \lor (i, j, y), \text{rest}(h_i')]. \ldots (1)\]

(10) is trivially true, since p and R_i do not impose any restrictions on h_i'. We may rewrite \[q \lor R_i \lor h_i' \ldots \] of (11) as follows:

\[ h_i \lor h_i' \lor h_i' \ldots \]

\[ q_{h_i'} \lor (i, j, y) \lor R_i \lor (i, y), h_i' \lor (i, j, y), \text{since q has no references to h_i', h_i', and R_i has no references to h_i'}. \] From the definition of R_i, we have \[R_i = \exists \text{rest}(h_i') \lor (i, y), h_i' \lor (i, y), h_i' \lor (i, y). \] Also, we have \[p \Rightarrow q_{h_i'} \lor (i, y), \text{since (II) \{P_j : y \{q\} is true. From these results, it is easy to see that (11) is true, and hence we have (I) \{P_j : Y \{q\}. The case S = P_j : y may be handled in similar fashion. The proof in case S = T, T' is straightforward. Next, consider the case S = \{a(\ell = 1, \ldots, m)b_\ell \rightarrow T_\ell\}. (II) \{p\}S\{q\} implies (II) \{p \land b_\ell \}T_\ell \{q\}, \ell = 1, \ldots, m\} \].

Hence, we have, \[(I) \{p \land b_\ell \} \lor V R_i \]T_\ell \{q \lor R_i \}. Then, we can conclude (I) \{p \lor R_i \}S \lor q \lor R_i \}, provided we can show the following:

\[ [p \lor R_i] = \exists \text{rest}(h_i') \lor [q \lor R_i] \lor [h_i' \lor h_i'] \ldots (12) \]

The above implication is easily seen to be true, since any \(h_i' \) will satisfy the predicate \(R_i \) if \(f(h_i') = g(h_i', i, j)+1 \) for all \(j \neq i\).

This proves (I) \{p \lor R_i \}[a(\ell = 1, \ldots, m)b_\ell \rightarrow T_\ell\} \{q \lor R_i \}. The other cases can be handled in exactly the same fashion.

Thus, we have shown, (II) \{p\}S\{q\} \Rightarrow (I) \{p \lor R_i \}S \lor q \lor R_i \}, for all constructs S which may appear in P_i. Hence, we have,

\[(II) \{I_{h_1} \land (h_1 = \varepsilon)\}P_i \{F_i\} \Rightarrow (I) \{\{I_{h_1} \land (h_1 = \varepsilon)\} \lor V R_i\}P_i \{F_i \lor R_i\} \Rightarrow
\]

\[(I) \{I_{h_1} \land (h_1 = \varepsilon)\} \lor (h_1 = \varepsilon)\}P_i \{F_i \lor R_i\}. \ldots (13)\]

Also, we have, (II) \{ \land I_{h_1} \}P \{ \land F_i \land [\text{Reduce}(h_1, \ldots, h_n) = (\varepsilon, \ldots, \varepsilon)] \} \Rightarrow
\]

\[(II) \{I_{h_1} \land (h_1 = \varepsilon)\} \lor (I) \{I_{h_1} \land (h_1 = \varepsilon)\} \lor (h_1 = \varepsilon)\} \lor (I) \{ \land I_{h_1} \}P \{ \land F_i \lor R_i \} \land [\text{Reduce}(h_1, \ldots, h_n) = (\varepsilon, \ldots, \varepsilon)] \ldots (14)\]

From the definition of \text{Reduce} and R_i, it is easy to see the following:
[Reduce(h₁,...,hₙ)=(ε,...,ε)] → [\( \bigwedge_{j=1}^{n} f(h_{j_i}) = g(h_{i_j},i,j) \)] → [\( \bigwedge_{i=1}^{n} R_i \)].

Thus (14) gives us,

\[
(\Pi) \{ \bigwedge_i I_i \} P \{ \bigwedge_i F_i \} \land [\text{Reduce}(h₁,...,hₙ)=(ε,...,ε)] \implies
\]

\[
(\Pi) \{ \bigwedge_i I_i \} P \{ \bigwedge_i F_i \} \land [\text{Reduce}(h₁,...,hₙ)=(ε,...,ε)]
\]

which proves that given a proof of a program P in SYS2, we can give an equivalent proof in SYS2.

4.2. An alternative operational semantics for CSP

In this subsection, we shall define a new operational semantics for CSP. To distinguish this semantics from the semantics defined in section 3, we shall refer to the semantics to be introduced now as SEM2, and the semantics of section 3 as SEM1. SEM2 will be defined by specifying a relation corresponding to the various CSP constructs. The relation corresponding to a construct S, which will be also denoted by S will be a set of pairs of the kind \((τ,τ')\), indicating that if we execute S starting in the initial state τ, then one of the possible final states is τ'.

Suppose \(S = \text{skip}\). Then, clearly the relation corresponding to S is \(\{(τ,τ)\}\). We shall write this as,

\(\text{skip} = \{(τ,τ)\}\).

Next, consider the assignment statement:

\(x := e = \{(τ,τ')|τ' = τ[x + e]\}\)

where \(τ[x + e]\) is the state obtained from τ by replacing the value of the variable x by the value of the expression e.

\(T;T' = \{(τ,τ')|\exists τ'. [(τ,τ') ∈ T \land (τ',τ'') ∈ T']\}\).

\(P_j!y = \{(τ,τ')|τ' = τ[h_i + h_i |-(i,j,y)]\}\),

where \(τ[h_i + h_i |-(i,j,y)]\) is the state got by extending the sequence \(h_i\) by \((i,j,m), m\) being the value of y in the state τ. (We are, of course, defining the relations corresponding to constructs appearing in the process \(P_i\)).

\(P_j?x = \{(τ,τ')|\exists z. τ' = τ[x + z, h_i + h_i |-(j,i,z)]\}\)

\([0(λ=1,...,m)b_x \rightarrow T^λ] = \{(τ,τ')|\exists r < m. [b_r(τ) \land (τ,τ') ∈ T^λ]\}\)

where \(b_r(τ)\) indicates that the boolean expression \(b_r\) has the value "true" in the state τ.
In order to define boolean guarded loops, we introduce a preliminary definition:

$$\ast\ast[(\alpha(\ell=1,\ldots,m)b_\ell;\alpha_\ell \rightarrow T_\ell)] = \{ (\tau, \tau') | \exists k \in \mathbb{N} . [b_\ell(\tau) \land (\tau, \tau') \in \alpha_\ell; T_\ell] \}.$$ 

Similarly, to define I/O guarded loops, we first introduce an auxiliary definition:

$$\ast\ast[(\alpha(\ell=1,\ldots,m)b_\ell;\alpha_\ell \rightarrow T_\ell)] = \{ (\tau, \tau') | \exists k \in \mathbb{N} . (\tau, \tau') \in \alpha_\ell; T_\ell] \}$$

and

$$\ast[(\alpha(\ell=1,\ldots,m)b_\ell;\alpha_\ell \rightarrow T_\ell)] = \{ (\tau, \tau"') | \exists k \in \mathbb{N} . [((\tau, \tau") \in \ast\ast[(\alpha(\ell=1,\ldots,m)b_\ell;\alpha_\ell \rightarrow T_\ell)] \land$$

$$\land \left[ \bigwedge_{r=1}^m b_r(\tau') \right] \lor$$

Finally, we must show the equivalence of the operational semantics SEM2 just introduced, and the semantics SEM1 defined in section 3. The reader who is not interested in the proof of equivalence of SEM1 and SEM2 may skip the rest of this section and proceed to section 4.3.

We first introduce a relation "→" which will be analogous to the relation defined in section 3; the difference between the two is that the relation to be defined next corresponds to the execution of one step of a process considered in isolation, whereas the relation defined earlier deals with all the processes.

**Definition:** $(S_i, \tau_i) \rightarrow (S_i', \tau_i')$ if any of the following clauses is satisfied:

1) $S_i = \text{skip}, \quad S_i' = E, \quad \tau_i' = \tau_i.$
2) $S_i = x := e, \quad S_i' = E, \quad \tau_i' = \tau_i[x := e].$
3) $S_i = P_j, y, S'_i = E, \tau_i = \tau_i [h_i + h_i \rightarrow (i, j, y)]$.

4) $S_i = P_j x, S'_i = E, \tau'_i = \tau_i [x \leftarrow m, h_i + h_i \rightarrow (j, i, m)],$ for some $m$.

5) $S_i \equiv [\sigma(\lambda = 1, \ldots, m) b_{\xi} \rightarrow T^\xi], \lambda = b_{\tau_1} (\tau_1), S'_i \equiv T^\xi; S'_i, \tau'_i = \tau_i$.

6) $S_i \equiv [\sigma(\lambda = 1, \ldots, m) b_{\xi} \rightarrow T^\xi], \lambda = b_{\tau_1} (\tau_1), S'_i \equiv T^\xi; S'_i, \tau'_i = \tau_i$.

7) $S_i \equiv [\sigma(\lambda = 1, \ldots, m) b_{\xi} \rightarrow T^\xi], \lambda = b_{\tau_1} (\tau_1), S'_i \equiv \alpha; T^\xi, \tau'_i = \tau_i$.

8) $S_i \equiv [\sigma(\lambda = 1, \ldots, m) b_{\xi} \rightarrow T^\xi], \lambda = b_{\tau_1} (\tau_1), S'_i \equiv \alpha; T^\xi; S'_i, \tau'_i = \tau_i$.

9) $S_i \equiv [\sigma(\lambda = 1, \ldots, m) b_{\xi} \rightarrow T^\xi], \lambda = b_{\tau_1} (\tau_1), S'_i \equiv \alpha; T^\xi; S'_i, \tau'_i = \tau_i$.

10) $S_i \equiv [\sigma(\lambda = 1, \ldots, m) b_{\xi} \rightarrow T^\xi], \lambda = b_{\tau_1} (\tau_1), S'_i \equiv \alpha; T^\xi; S'_i, \tau'_i = \tau_i$.

11) $S_i \equiv [\sigma(\lambda = 1, \ldots, m) b_{\xi} \rightarrow T^\xi], \lambda = b_{\tau_1} (\tau_1), S'_i \equiv \alpha; T^\xi; S'_i, \tau'_i = \tau_i$.

Next, we state a useful lemma:

**Lemma:** If $(\tau_i, \tau_i^F) \in P_i$, then there exists a sequence of pairs

$(\tau_i^1, S_i^1), \ldots, (\tau_i^k, S_i^k)$

such that $\tau_i = \tau_i^I, S_i = P_i, \tau_i^F = S_i^k, S_i = E$, and

for all $r < k$: $(\tau_i^r, S_i^r) \rightarrow (\tau_i^{r+1}, S_i^{r+1})$.

The proof is by a simple induction on the structure of $P_i$, and we omit the details. A sequence of the kind introduced in the above lemma will be called a "local computation sequence for $P_i$" to distinguish it from the computation sequence to be called a "global computation sequence", for the entire program introduced in section 3.

Also, in what follows we use $\rightarrow$ to denote the relation defined above, as well as the relation defined in section 3; the context will make clear which relation is intended; thus in $(S_i, \tau_i) \rightarrow (S'_i, \tau'_i)$, the intended relation is the one defined above; and, in $(S_1/\ldots/S_n, \tau) \rightarrow (S'_1/\ldots/S'_n, \tau')$ the intended relation is the one of section 3. And, $(S_i, \tau_i \{S'_i, \tau'_i\})$ will indicate that $(S_i, \tau_i)$ and $(S'_i, \tau'_i)$ are related according to clause (d) in the definition of \(\rightarrow\). The next lemma states the equivalence of SEM1 and SEM2.

**Lemma:** If $(S_i^1, \tau_i^1), \ldots, (S_i^n, \tau_i^n)$ are local computation sequences for $P_1/\ldots/P_n$, so that $S_i^1 = P_i, h_i^1 = \varepsilon, S_i = E$, and

$[\text{Reduce}(h_1^1, \ldots, h_n^1) = (\varepsilon, \ldots, \varepsilon)]$,

then there exists a global computation sequence $(S_1^u, \tau_1^u), \ldots, (S_n^u, \tau_n^u)$, such that $S_i = S_i^u, \tau_i = \tau_i^u$,

$S_i = E, \tau_i = (\tau_i^1, \ldots, \tau_i^n)$, and for all $v \leq u$, there exist $v_1, \ldots, v_n$ such that $v_i \leq u_i, i = 1, \ldots, n$, and
\[ S^u_i = S^u_i, \quad \tau^v = (\tau^v_1, \ldots, \tau^v_n), \quad [\text{Reduce } h^v_1, \ldots, h^v_n = (\varepsilon, \ldots, \varepsilon)] \] 

and there exists \( h^v_i \), \( \bar{h}^v_i \) such that,

\[ h^v_i = h^v_i \upharpoonright h^v_i \text{ for all } i = 1, \ldots, n, \text{ and } [\text{Reduce } (h^v_1, \ldots, h^v_n) = (\varepsilon, \ldots, \varepsilon)]. \]

**Proof:** We shall prove the result by constructing the sequence 
\([ (S^1, \tau^1), \ldots, (S^n, \tau^n) ] \). First, define \( S^1 = S^1_{/ /} \ldots / / S^n_1 = (\tau^1_1, \ldots, \tau^n_1) \).

Thus the result stated in the lemma clearly holds for \( v = 1 \).

Next, suppose we have constructed \((S^v, \tau^v)\), and that

\[ S^v = S^v_1 / / \ldots / / S^v_n, \quad \tau^v = (\tau^v_1, \ldots, \tau^v_n), \quad [\text{Reduce } h^v_1, \ldots, h^v_n = (\varepsilon, \ldots, \varepsilon)], \]

\[ h^v_i = h^v_i \upharpoonright h^v_i, \quad [\text{Reduce } (h^v_1, \ldots, h^v_n) = (\varepsilon, \ldots, \varepsilon)]. \]

Then, we have the following possibilities:

1) \( v_i = u_i \) for all \( i = 1, \ldots, n \). Then \( S^v_i = E, \quad \tau^v_i = \tau^v_i = u_i \), and the construction of the global computation sequence with the required properties is complete.

2) If (1) does not apply and there exists some \( j \) such that

\[ (S^v, \tau^v) d (S^v, \tau^v) \text{ for } d = 1, 2, 5, 6, 7 \text{ or } 10, \text{ then define } \]

\[ S^{v+1} = (S^1 / / \ldots / / S^n_j / / \ldots / / S^n_n), \quad \tau^{v+1} = (\tau^v_1, \ldots, \tau^v_j, \ldots, \tau^n_n), \]

and \( h^{v+1} = h^{v+1} \). If there is more than one \( j \) satisfying the stated condition, we choose any one of them, and define \((S^{v+1}, \tau^{v+1})\), and \( h^{v+1} \) as above. It is then easy to see that the lemma holds for \( v + 1 \).

3) If neither (1) nor (2) holds, then note first that for all \( j \) such that \( h^v_j = \varepsilon \), we have \( S^u_j = E = S^u_j \) and \( \tau^v_j = \tau^v_j \). Also, the facts that

\[ h^v_i = h^v_i \upharpoonright h^v_i, \quad i = 1, \ldots, n, \text{ and } [\text{Reduce } (h^v_1, \ldots, h^v_n) = (\varepsilon, \ldots, \varepsilon)], \]

in conjunction imply that at least one of the following clauses is true:

a) There exists \( j \) such that \( S^v_j = \ast[\ast{\lambda = 1, \ldots, m} b^v_\lambda \ast \ast{T}^v_\lambda], = v b^v \lambda (\tau^v_j) \)

\[ S^v_j = E, \quad \text{first}(h^v_j) = (j, T, t), \quad T = \{ k \exists r \ast[r \leq m \ast \ast{b^v (\tau^v_j) \ast(k = c(\lambda))}]) \}

and \( h^v_k = \varepsilon \) for all \( k \in T \) and hence \( S^v_k = E \) for all \( k \in T \).

In this case, define \( S^{v+1} = (S^1 / / \ldots / / E / / \ldots / / S^n_n) \), (i.e. all components of \( S^{v+1} \) except the \( j \)th one is the same as the corresponding component of \( S^v \), the \( j \)th component of \( S^{v+1} \) being \( E \)); and \( \tau^{v+1} = (\tau^v_1, \ldots, \tau^v_j, \ldots, \tau^n_n) \); and \( h^{v+1} = h^v_i \) for all \( i \neq j \), and \( h^v_j = \text{rest}(h^v_j) \).

Again, if this clause applies, it should be clear that the lemma is satisfied for \( v + 1 \).

b) There exists \( i, j \) such that \( S^v_i = P^i ? x, \quad S^v_j = P^j i, \quad \tau^v_j(y) = m, \quad \text{first}(h^v_i) = (j, i, m), \quad \text{first}(h^v_j) = (j, i, m). \)
In this case, define $S_i^{v+1} = E$, $S_j^{v+1} = E$, $S_k^{v+1} = S_k^V$ for all $k \neq i, j$; and $\tau_i^{v+1} = \tau_i^V$, $\tau_j^{v+1} = \tau_j^V$, and $\tau_k^{v+1} = \tau_k^V$, for all $k \neq i, j$.

Once again, it is straightforward to check that the lemma is satisfied for $v+1$, if this clause applies.

c) There are several other possible cases corresponding to I/O guarded selections, and I/O guarded loops; we consider a typical case, leaving the remaining ones, which may be dealt with similarly, to the interested reader:

There exist $i, j$ such that $S_i^{v} = [\sigma(\kappa = 1, \ldots, m) b \mid \alpha \rightarrow T^\rho_i]$, $S_j^{v} = [\sigma(p = \ldots, m') b' \mid \alpha' \rightarrow T^\rho_j]$, $\tau_i ^{v} = b \tau_i^V$, $\tau_j ^{v} = b \tau_j^V$, $\alpha = \alpha_p \tau_i^V$, $\alpha' = \alpha_p \tau_j^V$, $\alpha^V = \alpha_p \tau_i^V$, $\alpha'^V = \alpha_p \tau_j^V$, and $\alpha^V = \alpha_p \tau_i^V$, $\alpha'^V = \alpha_p \tau_j^V$, $\alpha^V = \alpha_p \tau_i^V$, $\alpha'^V = \alpha_p \tau_j^V$, $\alpha^V = \alpha_p \tau_i^V$, $\alpha'^V = \alpha_p \tau_j^V$. In this case, define $S_i^{v+1} = \alpha \tau_i^V$, $S_j^{v+1} = \alpha' \tau_j^V$, $S_k^{v+1} = S_k^V$ for all $k \neq i, j$, and $\tau^{v+1} = \tau^V$, $\tau_k^{v+1} = \tau_k^V$ for all $k$.

Again, it is routine to verify that the lemma is satisfied for $v+1$, if this clause applies.

Note also that irrespective of whether $S_i^{v+1}$ is defined according to clause (2), or according to one of the clauses in (3), the following is true:

if $S_i^V = (S_{i_{1}}^{v} /\ldots/ S_{i_{n}}^{v})$, and $S_i^{v+1} = (S_{i_{1}}^{v'} /\ldots/ S_{i_{n}}^{v'})$, then $v_i^{v'} > v_i$ for all $i = 1, \ldots, n$; and there exists $j \leq n$ such that $v_j^{v'} > v_j$.

Hence, $u_1, \ldots, u_n$ each being finite, and $v_i$ being less than or equal to $u_i$, we may conclude that ultimately clause (1) of the above construction will apply, giving us the global computation sequence

$(S_{i_{1}}^{v} /\ldots/ S_{i_{n}}^{v})$, $(S_{i_{1}}^{u} /\ldots/ S_{i_{n}}^{u})$ such that $S_{i_{i}}^{v} = \tau_i^V$ for $i = 1, \ldots, n$;

$S_i^{u} = E$ for $i = 1, \ldots, n$; $\tau_i^V = (\tau_i^{I_1}, \ldots, \tau_i^{I_n})$, $\tau_i^u = (\tau_i^{F_1}, \ldots, \tau_i^{F_n})$. This concludes the proof of the lemma.

The lemma along with the earlier one shows that if $((\tau_1^{I}, \ldots, \tau_n^{I}),$ $(\tau_1^{F}, \ldots, \tau_n^{F})$) belongs to the relation corresponding to the program $P_i /\ldots/ P_n$ as defined by SEM2, then there exists a (global) computation sequence of $P_i /\ldots/ P_n$, as defines by SEM1, such that starting in the initial state $(\tau_1^{I}, \ldots, \tau_n^{I})$, and following the computation sequence leads to the final state $(\tau_1^{F}, \ldots, \tau_n^{F})$. 
In order to complete the proof of equivalence of SEM1 and SEM2, we ought to prove the converse of the above result. This converse result is not, however, needed for the purposes of this paper, as the following argument shows: given a program $P_1//..//P_n$, and a predicate $p$ on the initial state of $P_1//..//P_n$, we shall define (in section 4.3) a predicate $sp(p,P_1//..//P_n)$ such that for every state $\tau'$ satisfying $sp(p,P_1//..//P_n)$, there exists a $\tau$ satisfying $p$, and $(\tau,\tau')$ belongs to the relation corresponding to $P_1//..//P_n$ as defined by SEM2; and show that \{p\}P_1//..//P_n\{sp(p,P_1//..//P_n)\} is provable using the axiomatic system SYS2. This will prove the (relative) completeness of SYS2 with respect to SEM2, and hence, by the result stated in the last paragraph, with respect to SEM1.

4.3 Completeness of the axiomatic system

In this section we prove the (relative) completeness of SYS2 with respect to SEM2. First we define the notion of the "strongest post-condition" $sp(p,S)$, given a predicate (the precondition) $p$ and a construct $S$ that may appear in the process $P_i$.

Definition: Given a predicate $p$ and a construct $S$ which may appear in $P_i$, $sp(p,S)$ is defined as follows:

1) $sp(p,\text{skip}) = p$;

2) $sp(p,x:=e) = \{\tau'_i | \exists \tau_i. \left[ (\tau'_i \in p \land \tau_i = \tau_i[x:=e]\right]\}$;

(Note: We treat predicates as characterising sets of states; thus $p$ above characterises the set: \{ $\tau_i | \tau_i \in p$ \}, and we define $sp(p,x:=e)$ by specifying the set that it characterises).

3) $sp(p,P_j . y) = \{\tau'_i | \exists \tau_i. \left[ (\tau'_i \in p \land \tau_i = \tau_i[y:=y]\right]\}$;

4) $sp(p,\text{P}_j ? x) = \{\tau'_i | \exists \tau_i. \left[ (\tau'_i \in p \land \exists m. \tau_i = \tau_i[x+m,h_i+h_i\leftarrow (j,i,m)\right]\right]\}$;

5) $sp(p,\text{[a(1,...,m)]b}_\xi \rightarrow T^\theta) = \{\tau'_i | \exists \tau_i. \left[ (\tau'_i \in p \land \exists m. (b_\xi(\tau_i))^\theta \tau_i \in sp(\tau_i, T^\theta)\right]\}$;

where, in $sp(\tau_i, T^\theta)$, $\tau_i$ denotes a set consisting of the single state $\tau_i$.

6) $sp(p,**[a(1,...,m)]b_\xi \rightarrow T^\theta) = \{\tau'_i | \exists \tau_i. \exists r. \left[ (\tau'_i \in p \land (\tau'_i, ..., \tau_i)^\theta \tau_i = \tau_i \land \forall r' < r. [\exists m. (b_\xi(\tau_i))^\theta \tau_i \tau_i' \in sp(\tau_i, T^\theta)\right]\}$;

7) $sp(p,*[a(1,...,m)]b_\xi \rightarrow T^\theta) = \{\tau'_i | \exists \tau_i. \left[ (\tau'_i \in p \land \exists m. (b_\xi(\tau_i))^\theta \tau_i \in sp(\tau_i, T^\theta)\right]\}$;

8) $sp(p,[a(1,...,m)]b_\xi : \alpha_\xi \rightarrow T^\theta) = \{\tau'_i | \exists \tau_i. \left[ (\tau'_i \in p \land \exists m. (b_\xi(\tau_i))^\theta \tau_i \in sp(\tau_i, T^\theta)\right]\}$;
9) \( \text{sp}(p,\ast [\alpha_{1}=1,\ldots,m]b_{1},\alpha_{1} \rightarrow T_{1}^{\ast}) = (\tau_{1}^{\ast} | \exists \tau_{1,1} | \ldots, \tau_{1,n} | [\tau_{i} = \tau_{1,1} \wedge \tau_{i,1} = \tau_{1}^{\ast}]) \wedge \\
\forall \tau_{i} < \tau_{1} \exists \tau_{i}^{\ast} \leq \tau_{i} \in \text{sp}(\tau_{i}, \alpha_{1}, \alpha_{1}, \beta_{1}^{\ast}) \{ \tau_{1} = \tau_{1}^{\ast} \wedge \tau_{1}^{\ast} \rightarrow T_{1}^{\ast}) \} \}
\)

10) \( \text{sp}(p,\ast [\alpha_{1}=1,\ldots,m]b_{1},\alpha_{1} \rightarrow T_{1}^{\ast}) = \{ \tau_{1}^{\ast} | \exists \tau_{1,1} | \ldots, \tau_{1,n} | \tau_{1}^{\ast} = \tau_{1,1} \wedge \tau_{1,1} \in \text{sp}(\tau_{1}, \alpha_{1}, \alpha_{1}, \beta_{1}^{\ast}) \} \wedge \\
\begin{cases} \\
\bigwedge_{\tau_{i} = 1}^{m} \tau_{i}^{\ast} \rightarrow T_{1}^{\ast} \end{cases} \cup \\
\{ \tau_{1}^{\ast} | \exists \tau_{1,1} | \ldots, \tau_{1,n} | \tau_{1}^{\ast} = \tau_{1,1} \wedge \tau_{1,1} \in \text{sp}(\tau_{1}, \alpha_{1}, \alpha_{1}, \beta_{1}^{\ast}) \} \wedge \\
\begin{cases} \\
\bigwedge_{\tau_{i} = 1}^{m} \tau_{i}^{\ast} \rightarrow T_{1}^{\ast} \end{cases} \}
\)

11) \( \text{sp}(p,T^{1},T^{2}) = \{ \tau_{1}^{\ast} | \exists \tau_{1,1} \tau_{1,1}^{\ast} | \tau_{1}^{\ast} = \tau_{1,1} \wedge \tau_{1,1} \in \text{sp}(\tau_{1}, T^{1}) \wedge \tau_{1,1}^{\ast} \in \text{sp}(\tau_{1}^{\ast}, T^{2}) \} \).

Next, we extend the definition of sp to handle a set of parallel processes.

**Definition:** Suppose \( I_{1}, \ldots, I_{n} \) are given predicates characterising the initial states (not including the communication sequences) of the processes \( P_{1}, \ldots, P_{n} \) of CSP program \( P_{1} /\ldots// P_{n} \), then

\[
\text{sp}(I_{1} \wedge \ldots \wedge I_{n}, P_{1} /\ldots// P_{n}) = \{ (\tau_{1}, \ldots, \tau_{n}) | \exists \tau_{1,1}, \ldots, \tau_{1,n} | \left[ \forall i \leq n. (s_{i} \in I_{i} \wedge h_{i} = \epsilon) \right] \wedge \\
\left[ \forall i \leq n. (\tau_{i} = \tau_{i,1} \wedge \tau_{i,1} \in \text{sp}(\tau_{i}, P_{i})) \wedge (\text{Reduce}(h_{i}', \ldots, h_{n}') = (\epsilon, \ldots, \epsilon)) \right] \}
\]

where \( s_{i} \) is the state component of \( \tau_{i}, h_{i} \) is the sequence component of \( \tau_{i}, \) and \( h_{i}' \) the sequence component of \( \tau_{i}^{\ast} \).

The following two theorems are the key results of this section.

**Theorem:** If \( S \) is a CSP program or a construct appearing in one of the processes of such a program, and \( \tau^{\ast} \in \text{sp}(p, S) \), \( p \) being a given predicate then there exists a \( \tau \in p \) such that \( (\tau, \tau^{\ast}) \in S \) (i.e. \( (\tau, \tau^{\ast}) \) is a member of the relation corresponding to \( S \).

The proof is by a straightforward induction on the structure of \( S \). First we consider the case when \( S \) is a construct appearing in a process. Then we have the following possibilities:

1) \( S \equiv \text{skip} \). Suppose \( \tau_{1}^{\ast} \in \text{sp}(p, S) \). Then, since \( \text{sp}(p, \text{skip}) = p, \tau_{1}^{\ast} \in p \). Also by definition (of SEM2), \( \text{skip} = \{(s_{1}, s_{1})\} \). Hence, \( (\tau_{1}^{\ast}, \tau_{1}^{\ast}) \in \text{skip} \).

2) \( S \equiv x := e \). Suppose \( \tau_{1}^{\ast} \in \text{sp}(p, S) \). Then there exists \( \tau_{1} \) such that

\[
[\tau_{1} \in p] \wedge [\tau_{1}^{\ast} \equiv \tau_{1}[x := e]] ; \text{and} \quad [x := e] = \{(s_{1}, s_{1}) | s_{1} = s_{1}[x := e] \},
\]

proving the required result.
3) \( S \equiv P_j \neq x \). Then, \( \tau'_i \in \text{sp}(p, S) \) implies there exists \( \tau_i \) and \( m \) such that
\[
[\tau_i \in p] \land [\tau'_i = \tau_i[x + m, h_i + h_i |- (j, i, m)]]; \text{ hence } (\tau'_i, \tau_i) \in P_j \neq x, \text{ since } \\
P_j \neq x \triangleq \{ (\tau'_i, \tau_i) \mid \exists m. \tau'_i = \tau_i[x + m, h_i + h_i |- (j, i, m)] \}.
\]

4) \( S \equiv \{ \circ (\ell = 1, \ldots, m) \} \). Then \( \tau'_i \in \text{sp}(p, S) \) implies that there exists \( \tau_i \) and \( r \) such that
\[
[\exists r(\tau_i)] \land [\tau_i \in p] \land [\tau'_i \in \text{sp}(\tau_i, T^r)]; \text{ hence } \\
(\tau_i, \tau'_i) \in S, \text{ since } S = \{ (\tau_i, \tau'_i) \mid \exists r. [\exists r(\tau_i) \land (\tau_i, \tau'_i) \in T^r] \}.
\]

5) \( S \equiv T^1 \land T^2 \). Then, \( \tau'_i \in \text{sp}(p, S) \Rightarrow \exists \tau_i \exists \tau''_i \exists \tau''_{i'} [\tau_i \in p \land \exists \tau''_i [\tau''_i \in \text{sp}(\tau_i, T^1) \land \\
\tau''_{i'} \in \text{sp}(\tau''_i, T^2)]] \).

Hence \( (\tau_i, \tau'_i) \in S, \text{ since } S = \{ (\tau'_i, \tau''_i, \tau''_{i'}) \mid (\tau'_i, \tau''_i) \in T^1 \land (\tau''_{i'}, \tau''_{i'}) \in T^2 \} \).

The remaining cases are equally simple and are left to the reader.

Finally, consider the case \( S \equiv P_1 \neq p \). We shall assume that \( p \) is of the form \( \bigwedge_{i=1}^{n} I_i \), where \( I_i \) characterises the initial state (not including the communication sequence) of \( P_i \).

Then, \( \tau' = (\tau'_1, \ldots, \tau'_n) \in \text{sp}(p, S) \) implies that there exists \( \tau = (\tau_1, \ldots, \tau_n) \) such that
\[
[\forall i \leq n. [s_i \in I_i \land h_i = \varepsilon \land \tau_i \in \text{sp}(\tau_i, P_i)]] \land [\text{Reduce}(h'_1, \ldots, h'_n) = (\varepsilon, \ldots, \varepsilon)].
\]

And \( P_1 \neq p \equiv \{ ((\sigma_1, \ldots, \sigma'_1), (\sigma''_1, \ldots, \sigma''_n)) \mid [\forall i \leq n. [h_i = \varepsilon \land (\sigma_i, \sigma_i') \in P_i]] \land \\
[\text{Reduce}(h''_1, \ldots, h''_n) = (\varepsilon, \ldots, \varepsilon)] \}.
\]

Hence \( ((\tau_1, \ldots, \tau_n), (\tau'_1, \ldots, \tau'_n)) \in P_1 \neq p \), thus proving the theorem.

Corollary. If \( \models \{p\} S \{q\} \) is true (in SEM2) for given predicates \( p, q \) and CSP program (or CSP construct appearing in a process) \( S \), then \( \text{sp}(p, S) \models q \).

The corollary follows directly from the above theorem, and the intended meaning of \( \models \{p\} S \{q\} \) is true".

The name \( \text{sp} \) (for strongest post-condition) was, of course, chosen in anticipation of the above corollary. The next theorem proves the completeness of SYS2 (with respect to SEM2, and hence, by the result of section 4.2, with respect to SEM1). By the corollary, what we need to show, in order to establish completeness, is the following:
\( \models \{p\} S \{\text{sp}(p, S)\} \). Note also that since, by the results of section 4.1, any proof in SYS2 can be converted into an equivalent proof in SYS1, we are in fact proving completeness of SYS2 as well as of SYS1.

Theorem: for any given predicate \( p \), and CSP construct \( S \), \( \{p\} S \{\text{sp}(p, S)\} \) is provable in SYS2.
Proof: The proof is again by induction the structure of $S$.

1) $S = \text{skip}$: $\text{sp}(p,S) = p$; hence, by axiom A1 of section 4.1, $\vdash p \{p\} S \{\text{sp}(p,S)\}$

2) $S = x := e$: $\text{sp}(p,S) = \{\tau_1 \mid \exists \tau_i. [\tau_i \in p \land \tau_i' = \tau_i[x := e]]\}$. In order to show $\vdash \{p\} S \{\text{sp}(p,S)\}$, we need to show $p \Rightarrow [\text{sp}(p,S)]^x_e$. Is do this, we must first define, given the predicate of, the set corresponding to $q^x_e$:

$$q^x_e = \{\sigma \mid \sigma[x + e] \in q\}$$

Then $[\text{sp}(p,x := e)]^x_e = \{\sigma_1 \mid \sigma_1[x + e] \in \text{sp}(p,x := e)\}$.

Suppose $\tau_i \in p$; suppose also that $\tau_i' = \tau_i[x := e]$; (Note: we assume that $\tau_i$ is a well-defined state; i.e. the expression $e$ has a proper value in state $\tau_i$). Then by definition of $\text{sp}(p,x := e)$, $\tau_i' \in \text{sp}(p,x := e)$.

Hence, by definition of $[\text{sp}(p,x := e)]^x_e$, $\tau_i \in [\text{sp}(p,x := e)]^x_e$. Thus we have $\vdash (\tau_i \in p) \Rightarrow (\tau_i \in [\text{sp}(p,x := e)]^x_e)$ i.e. $p \Rightarrow [\text{sp}(p,x := e)]^x_e$. Hence, $\vdash \{p\} x := e \{\text{sp}(p,x := e)\}$ by axiom A2.

3) $S = P_j ? x$: $\text{sp}(p,S) = \{\tau_i' \mid \exists \tau_i, m. [\tau_i \in p \land \tau_i' = \tau_i[x + m, h_i + h_i' [-(j,i,m)]]\}$

$$[\text{sp}(p,S)]^x_{m,h_i}[m,h_i[ -(j,i,m)] = \{\tau_i \mid \tau_i[x + m, h_i + h_i' [-(j,i,m)] \in \text{sp}(p,P_j ? x)\}$$

We need to show that $p \Rightarrow \forall m. [\text{sp}(p,P_j ? x)]^x_{m,h_i}[m,h_i[ -(j,i,m)]$.

Suppose $\tau_i \in p$. We must show that $\forall m. \tau_i \in [\text{sp}(p,P_j ? x)]^x_{m,h_i}[m,h_i[ -(j,i,m)]$

Let $\tau_i' = \tau_i[x + m, h_i + h_i'[-(j,i,m)]$. Then, by definition of $\text{sp}(p,P_j ? x)$, $\tau_i' \in \text{sp}(p,P_j ? x)$. Hence, by definition of $[\text{sp}(p,P_j ? x)]^x_{m,h_i}[m,h_i[ -(j,i,m)]$, $\tau_i \in [\text{sp}(p,P_j ? x)]^x_{m,h_i}[m,h_i[ -(j,i,m)]$; and this argument is independent of the value of $m$. Hence, $\forall m. [\tau_i \in [\text{sp}(p,P_j ? x)]^x_{m,h_i}[m,h_i[ -(j,i,m)]$. Hence, $p \Rightarrow \forall m. [\text{sp}(p,P_j ? x)]^x_{m,h_i}[m,h_i[ -(j,i,m)]$. Therefore, by axioms A3, we have $\vdash \{p\} P_j ? x \{\text{sp}(p,P_j ? x)\}$.

4) $S = P_j ! y$: this case is similar to (3) and we omit the details.

5) $S = \{a(\ell = 1, \ldots, m)b_{\ell} \rightarrow T_{\ell}\}$. The inference rule of SYS2 for this case is,

$$\frac{\{p \land b_{r}\} \vdash^r \{q\}, \ r = 1, \ldots, m}{\{p\} \vdash \{a(\ell = 1, \ldots, m)b_{\ell} \rightarrow T_{\ell}\}\{q\}}$$
Also \( \text{sp}(p,S) = \{ \tau_1' \mid \exists \tau_1', r. [\tau_1' \in p \land b_{r}(\tau_1') \land \tau_1' \in \text{sp}(\tau_1', T^r)] \} \)

\[
= \text{sp}(p \land b_1, T^1) \cup \cdots \cup \text{sp}(p \land b_m, T^m) \\
= \bigvee_{\ell=1}^{m} \text{sp}(p \land b_{\ell}, T^\ell).
\]

By the inductive hypothesis we have, for \( \ell=1,\ldots,m, \neg\{p \land b_\ell\} T^\ell \{ \text{sp}(p \land b_\ell, T^\ell) \}
\)

and, hence, by the rule of consequence \( \neg\{p \land b_\ell\} T^\ell \{ \bigvee_{\ell=1}^{m} \text{sp}(p \land b_{\ell}, T^\ell) \} \), \( \ell=1,\ldots,m \).

Hence, by the rule of inference above, we have,

\[
\neg\{p[\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell] \{ \bigvee_{\ell=1}^{m} \text{sp}(p \land b_{\ell}, T^\ell) \} \text{ is } i.e. \neg\{p\} S\{\text{sp}(p,S)\}.
\]

6) \( S = *\{\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell\} \). The rule of inference is,

\[
\frac{\{q \land b_r\} T^r \{q\} , \ r=1,\ldots,m}{\{q\} \ast\{\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell\} \{q \land \neg b\}}
\]

where \( \neg b = \bigwedge_{\ell=1}^{m} \neg b_{\ell} \). q is usually called the "loop invariant".

On account of the rule of consequence, we may rewrite the above rule as

\[
p \rightarrow q
\]

\[
q_r \rightarrow q, \ r=1,\ldots,m
\]

\[
\frac{\{q \land b_r\} T^r \{q_r\} , \ r=1,\ldots,m}{\{p\} \ast\{\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell\} \{q \land \neg b\}}
\]

Now \( \text{sp}(p,S) = \{ \tau_1' \mid \exists \tau_1', r. [\tau_1' \in p \land \tau_1' \in \text{sp}(\tau_1', T^r)] \} \land \neg b(\tau_1') \).

\[
= \{ \tau_1' \mid \tau_1' \in \text{sp}(p, **\{\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell\}) \land \neg b(\tau_1') \}.
\]

Define \( q = \text{sp}(p,**\{\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell\}) \).

Suppose \( \tau_1 \in p \). Then, it is easy to see, from the definition of \( \text{sp}(p,**\{\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell\}) \), that \( \tau_1 \in q \). Thus \( p \rightarrow q \). Next, define,

\( q_r = \text{sp}(q \land b_r, T^r) \). Hence, by the inductive hypothesis, \( \neg\{q \land b_r\} T^r \{q_r\} \).

Moreover, it is easy to see from the definition of \( q \), that

\( [\tau_1 \in q_r \rightarrow \tau_1 \in q] \) i.e. \( [q_r \rightarrow q] \). Finally, from the definition of \( \text{sp}(p,**\{\bigcirc(\ell=1,\ldots,m)b_{\ell} \rightarrow T^\ell\}) \) \( \equiv q \land \neg b \). Hence, we have \( \neg\{p\} S\{\text{sp}(p,S)\} \).

7) \( S = T_1 \cap T_2 \). Then, \( \text{sp}(p,S) = \{ \tau_1' \mid \exists \tau_1', \tau_2'. [\tau_1' \in p \land \tau_2' \in \text{sp}(\tau_1', T^1) \land \tau_1' \in \text{sp}(\tau_2', T^2)] \} \)

\[
= \text{sp}(\text{sp}(p,T^1),T^2).
\]

Then, by the rule for sequential composition, we have \( \neg\{p\} T_1 \cap T_2 \{\text{sp}(p,T_1 \cap T_2)\} \).

The cases for I/O guarded selection and repetition are handled as in (5) and (6) respectively, and we omit the details.
8) Finally, consider the case \( S = P_1 \ldots P_n, \ p = \bigwedge_{i=1}^n I_i \). 

Then \( sp(p,S) = (\tau_1', \ldots, \tau_n') \models \exists \tau_1, \ldots, \tau_n. \left[ \bigwedge_{i=1}^n (s_i \in I_i \land (h_i = \varepsilon) \land \tau_i' \in sp(\tau_i, P_i)) \right] \land \left[ \text{Reduce}(h_1', \ldots, h_n') = (\varepsilon, \ldots, \varepsilon) \right] \).

By the inductive hypothesis, we may assume,

\[ \vdash \{ I_i \land (h_i = \varepsilon) \} P_i \{ \text{sp}(I_i \land h_i = \varepsilon), P_i \} \]

Hence, by the rule for parallel composition, we have

\[ \vdash \{ \bigwedge_{i=1}^n I_i \} P_1 \ldots P_n \{ \text{sp}(\bigwedge_{i=1}^n (I_i \land h_i = \varepsilon), P_i) \land \text{Reduce}(h_1', \ldots, h_n') = (\varepsilon, \ldots, \varepsilon) \} \]

i.e. \( \vdash \{ p \} P_1 \ldots P_n \{ \text{sp}(p, P_1 \ldots P_n) \} \).

That completes the proof of the theorem.

Before concluding, some remarks are in order: the reader familiar with Apt et al[2] would not have failed to notice how closely our proof of completeness resembles their proof of completeness (of a simple sequential language); our proof was, in fact, inspired by the proof in Apt et al[2]. Note also that, as in Apt et al[2], we may restrict our assertions to be recursively enumerable; the system would still be complete, since the intermediate assertions are all r.e., if the given pre and post conditions are r.e. Also, the system is complete if we restrict our assertions to be first order, since all intermediate assertions can be converted to first order predicates; the only difficult case is the loop invariants and they may be handled by using an appropriate godelization of tuples of states such as \( \langle \tau_1', \ldots, \tau_n' \rangle \). It should be noted that if we restrict our assertions to be recursive then, as follows from the arguments in Apt et al[2], our system is not complete, since no auxiliary variables, which can serve as loop counters, are allowed in our system.

Finally we would like to consider the following question: why have we introduced SYS1, when SYS2 is simpler, complete, and can be easily proved to be consistent (independently of the consistency of SYS1)? The answer is that, in our experience with actual examples, SYS1 has, occasionally, proved to be easier to use than SYS2. Surprisingly, however, in the case of programs where each of the processes receives, as well as sends numbers to the same set of processes, the proofs in the two systems are quite similar to each other. It is only in the case of programs where each (or, at least, most) of the processes receive numbers from one set of processes, and send out numbers to
another set of processes, that proofs in SYS1 are usually simpler than
in SYS2. The set partitioning and gcd programs of [7] are examples
of the former kind, and the matrix multiplication and prime number gen-
eration programs of [7] are examples of the latter kind, and the reader
may wish to look at the proofs given in [7] of these programs to see the
validity of our remarks.

Acknowledgements

The author is highly obliged to Prof. O.-J. Dahl for a number of
discussions and suggestions; in particular, the axioms A1, and A2,
of SYS2 were suggested by Prof. Dahl. A number of other members of
the Informatics group at the University of Oslo also participated in
some of these discussions. The author sincerely thanks Mrs. Madsen for
her usual efficient typing of the manuscript. The research reported in
this paper was performed while the author was supported by a post-doctor
ate fellowship of the Royal Norwegian Council for Scientific and In-
dustrial Research (NTNF).

References

1. Apt, K.R., Formal justification of a proof System for Communicating
Sequential Processes, draft, Erasmus University, Rotteram, The
Netherlands, 1981.
are not enough - or are they ?, Theoretical Computer Sc., 8(1979), 73-7
3. Apt, K.R., Francey, N., and de Roever, W.P., A proof system for communi-
5. Owicky, S., Axiomatic proof techniques for parallel programs,
6. Soundararajan, N., Axiomatic Semantics of communicating Sequential
processes, research report, Institute for Informatics, University of
7. Soundararajan, N., Proofs of correctness of CSP programs,
research report, Institute for Informatics, University of Oslo (1981).