Partial Correctness of Communicating Sequential Processes.

by Zhou Chao Chen and C.A.R. Hoare.

~~July 1980~~
Jan 1981

Abstract:

We introduce a programming notation to describe the behaviour of groups of parallel processes, communicating with each other over a network of named channels. An assertion is a predicate with free channel names, each of which stands for the sequence of values which have been communicated along that channel up to some moment in time. A process invariantly satisfies an assertion if that assertion is true before and after each communication by that process. We present a system of inference rules for proving that processes satisfy assertions, and illustrate their use on some examples. The validity of the inference rules is established by constructing a model of the programming notation, and by proving each inference rule as a theorem about the model. Limitations of the model and proof system are discussed in the conclusion.

CR categories:

Key words and phrases: program correctness, parallel programming, axiomatic semantics, denotational semantics, communicating processes.

PRG. 45, Banbury Rd. Oxford OX2 6PE.

0. Introduction

The possibility of using multiple processors, simultaneously carrying out a single task, has opened a new dimension in computer programming. To assist the programmer in exploiting the possibility, it must be made available within the context of a high level language; and one such approach is (informally) described in [2]. But informal descriptions are notoriously unreliable, and some of the intricacies of parallelism are notoriously subtle. For sequential programming languages, (these problems have been solved by the techniques of) denotational semantics [5]. Furthermore, the axiomatic method, which provides a basis for proofs that programs expressed in a language has been extended to parallel programs [6]. (This paper) will meet their specification, makes an ambitious attempt to give both a denotational and an axiomatic definition

for a language involving parallelism, and proves that the definitions are consistent.

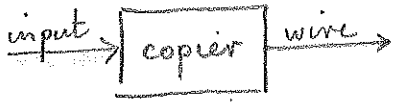
To achieve this goal, the language has been kept very simple; for example it does not (include) local variables, assignments, or even sequential composition; and loops are constructed by tail recursion. In spite of these omissions, the expressive power of the language can be illustrated by non-trivial examples [1].

A more serious deficiency is that the proof method establishes only partial correctness, and cannot prove (or even express) the absence of deadlock. There does not seem to be any easy way of extending the method to deal with this problem. However, the fact that we evade the problem of "fairness" seems to be a merit.

1 Processes and their description. 31

We regard a process as a potential component of a network of processes connected by named channels, along which they communicate with each other. Each (occurrence of a) (communication between a process and one of its neighbours in the network is denoted as a pair "c.m", where "m" is the value of the message and "c" is the name of the channel along which it passes. For example, "output.3" denotes communication of the value 3 on the channel named "output", and "input.3" denotes communication of the same value on a different channel. For the sake of simplicity, we do not distinguish the direction of communication: transmission of a message on a channel and its receipt by another process on the same channel are regarded as the same event, which occurs only when both processes are ready for it. Thus "wire.ACK" denotes simultaneous transmission and receipt of an acknowledgement signal ACK along a "wire" channel.

The sequence of communications in which a process engages up to some moment in time can be recorded as a trace of the behaviour of that process. For example, a process named "copier" is connected to its neighbours by two channels named "input" and "wire":



The task of the copier is just to copy messages from the input channel to the wire. Thus the following are possible traces of its behaviour:

(i) $\langle\rangle$, i.e., the empty trace, describing its behaviour before it has input anything.

(ii) $\langle input.3, wire.3\rangle$ is a sequence of two communications, describing its behaviour when it has copied its first message, (which has) (value 3.

(iii) $\langle input.27, wire.27, input.0, wire.0, input.3\rangle$ describes a different possible behaviour of the copier.

Another example is a process named "recopier" which simply copies messages from "wire" to "output". Its possible traces include:

$\langle\rangle$, $\langle wire.3, output.3\rangle$, $\langle wire.27, output.27, wire.0\rangle$, etc.

In this paper, we regard a process as being defined not by its internal states and transitions, but rather by its externally observable behaviour; or, more precisely, by the set of all (possible) traces of its communications with its neighbours. In the case of the copier process, this set will include (for example) all traces of the form ($\langle input.m\rangle$ or) $\langle input.m, output.m\rangle$, where m ranges over all possible message values. Thus a process can be identified with a formal language over an alphabet of communications. Such languages can conveniently be defined by a notation similar to the production rules of a formal grammar, as will be shown in the remainder of this section.

1.1. Preliminaries.

We shall assume that the reader is familiar with the following kinds of syntactic category, and their usual interpretation.

(1) Constants, denoting particular values, e.g., 3 or 27.

(2) Variables, denoting unknown values, e.g., i, j, k, x, y, z.

(3) Expressions, built from variables, constants, and operators, each of which defines a value in terms of its constituent variables, e.g. $(3*x+y)$. Note: expressions are not allowed to contain process names or channel names.

(4) Names and expressions denoting sets of values or types, e.g.

tabulate

NAT denotes the natural numbers $\{0, 1, 2, \ldots\}$
$\{0..3\}$ denotes the finite range $\{0, 1, 2, 3\}$
{ACK, NACK} denotes a pair of acknowledgement signals.

In a practical programming notation, a strict typing system would be desirable to ensure consistency of variables, expressions, and messages passing along each channel. For simplicity, in this paper we shall henceforth ignore the matter.

We now introduce the following new syntactic categories. The forms of the identifiers and variables and expressions are familiar; and we rely on the good will of the reader to distinguish them by context or meaning.

(5) Process names, serving as non-terminal symbols of a grammar, e.g. copier, recopier, sender, receiver.

(6) Process array names, such as q, mult... If e is an expression, then q[e] is a subscripted process name, denoting a particular process for each distinct value of e.

(7) Process equations of the form $p \triangleq P$, where p is a process and P is an expression defining the behaviour of the process. If the name p occurs inside the expression P, the equation is recursive in the familiar sense.

(8) Process array equations, of the form "$q[i:M] \triangleq P$", where q is a process array name, M is a set-valued expression (or type), i is a variable ranging over M, and P is an expression defining the behaviour of a process. P may contain occurrences of the variable i; it is the different values of i that can differentiate the behaviour of distinct elements of the array q. As before, an occurrence of $q[e]$ inside P is understood in the usual recursive sense.

(9) Lists of equations for processes and process arrays, which declare and define a set of processes and process arrays, possibly by mutual recursion.

Note. Process names will be used only for recursive definition or for abbreviation, and never to specify the source or destination of a communication. These are specified indirectly by use of channel names, as described below

(10) Channel names, e.g. input, wire, output.

(11) Channel array names, e.g. row, col. If e is an expression, then row[e], col[e] are subscripted channel names, denoting a particular distinct channel for each distinct value of e.

(12) Channel arrays, of the form "$c[M]$" where c is a channel array name, and M denotes a set of possible subscript values e.g. col[0..3] denotes the set {col[0], col[1], col[2], col[3]}

(13) Lists of channels, including channel names, channel arrays, and subscripted channel names. These are used to declare or specify the sets of channels connecting pairs or networks of processes.

1.2, Process Expressions.

It remains to specify the most important feature of the notational system, namely the process expressions which appear on the right hand side of (process) equations, and thus define the behaviour of the processes named on the left hand side. The exposition of this section is quite informal. Formally speaking, each process expression defines a set of traces of its possible behaviour, in terms of the values of its free variables, as described in 3.1 and 3.2.

(1) STOP is the process that never does anything. Its only trace is <>.

(2) A process name denotes the process specified by the process expression appearing on the right hand side of its defining equation.

(3) A subscripted process name $q[e]$ denotes the process Q', where the definition of q has the form $q[i:M] \triangleq Q$, and Q' is formed from Q by replacing each occurrence of i by the value of e, provided that this is in M. ~~Thus the behaviour of each element of the array is differentiated.~~

(4) If c is a channel name (possibly subscripted), and e is an expression, and P is a process expression, then "$(c!e \rightarrow P)$" is a process expression. It denotes the process which first transmits the value of e on channel c, and then behaves like P.

e.g., $(wire!j \rightarrow copier)$, $(col[i]!(3*i+j) \rightarrow mult[i])$.

(5) If x is a variable, and M is a set expression, and c is a channel name (possibly subscripted), and P is a process expression, (in general containing the variable x) then "$(c?x:M \rightarrow P)$" is a process which first communicates on channel c any value of the set M, (provided M is nonempty). If x denotes the value communicated, P specifies the subsequent behaviour of the process. This models input of a value from channel c to the variable x, which serves as a local (bound) variable in P. The actual value given to x is usually determined by an output "$c!e$" performed by the process of the network located at the other end of the channel c.

e.g. $(input?x:NAT \rightarrow (wire!x \rightarrow copier))$

$(col[i-1]?y:NAT \rightarrow (col[i]!(3*x+y) \rightarrow mult[i]))$

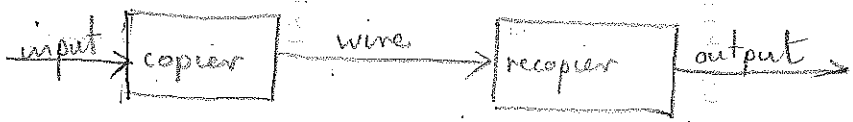
Note. In future, brackets may be omitted on the convention that the arrows is right associative, e.g.

$wire?x:NAT \rightarrow output!x \rightarrow recopier$

(6) If P and Q are process expressions, then so is $(P|Q)$. It denotes a process that behaves either like P or like Q; the choice between them may be regarded as non-determinate.
e.g. $((wire!ACK \rightarrow output!x \rightarrow receiver)|(wire!NACK \rightarrow receiver))$
Note. In future the inner brackets may be omitted, on the convention that $\rightarrow$ binds tighter than $|$.

(7) Let X be the set of channel names occurring in P and let Y be the set of channel names occurring in Q. Then $(P_X||_Y Q)$ denotes a network constructed from processes P and Q, which are connected to each other by channels in the intersection of both sets X and Y. However P may still be (externally) connected to other neighbours by channels in the set $(X-Y)$, and Q may be externally connected by channels in the set $(Y-X)$. Thus each external communication by $(P_X||_Y Q)$ is either made by P on a channel of $(X-Y)$, and is ignored by Q, or viceversa. However any internal communication between P and Q uses one of the channels of $X \cap Y$; A communication on such a channel c requires simultaneous participation by both P and Q; one of them determines the value transmitted by an output "c!e", and the other is prepared to accept any value (of the set M) by (an) input ".c?x:M".

eg. A network diagram:

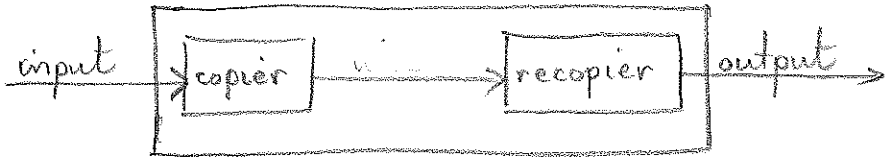


is denoted by the expression $(copier_X||_Y recopier)$
where $X = \{input, wire\}$ and $Y = \{wire, output\}$.
Note. When the content of the sets X and Y are clear from the context, or from an accompanying diagram, it is convenient to omit them.

(7) Let L be a list of channels which are used for internal communication between processes of a network P. Then (chan L; P) is a process in which all internal communications

along any of the channels in L are removed from the externally recordable traces of P. Such communications is expected to occur independently and automatically, whenever the processes connected by the channel are all ready for it. If more than one such communication is possible, the choice between them is non-determinate.

The effect of declaring channels local to a network can be pictured by enclosing the network in a "black box", and removing the names of the internal channels, For example



is a pictorial representation of the process expression

(chan wire; (copier || recopier))

Note. Our decision to ignore the direction of communication leaves open the possibility that a channel may have a single process which outputs on it and many other processes which input from it. All such inputs occur simultaneously with the output. In theory, it is possible that all processes connected by a channel can simultaneously input from it, with a highly non-determinate result. In our examples we shall avoid such phenomena; a practical programming language should (be designed to) make them impossible.

1.3. Examples of process definitions.

(1) A process which endlessly copies numbers from a channel named "input" to a channel named "wire",

copier ≙ (input?x:NAT → wire!x → copier).

A similar process is:

recopier ≙ (wire?y:NAT → output!y → recopier).

(2) A "sender" process inputs a value y on a channel named "input" and then behaves like $q[y]$:

$$sender \triangleq (input?y : M \rightarrow q[y]).$$

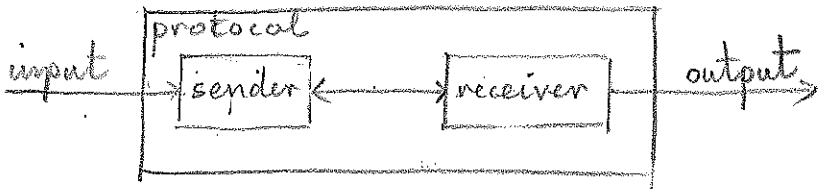
(3) The process $q[x]$ (for any x in M) first transmits the value x along the channel named "wire"; it then inputs from the wire either an ACK signal or a NACK signal. In the first case, its subsequent behaviour is the same as that of the sender. In the other case, it retransmits the message as often as necessary, until it gets ACK:

$$q[x:M] \triangleq (wire!x \rightarrow (wire?y:\{ACK\} \rightarrow sender \mid wire?y:\{NACK\} \rightarrow q[x]))$$

(4) A "receiver" process inputs messages on the wire. It then either returns an "ACK" signal and outputs the message, or it returns a "NACK" signal and expects the message to be retransmitted. The choice between these alternatives is non-determinate:

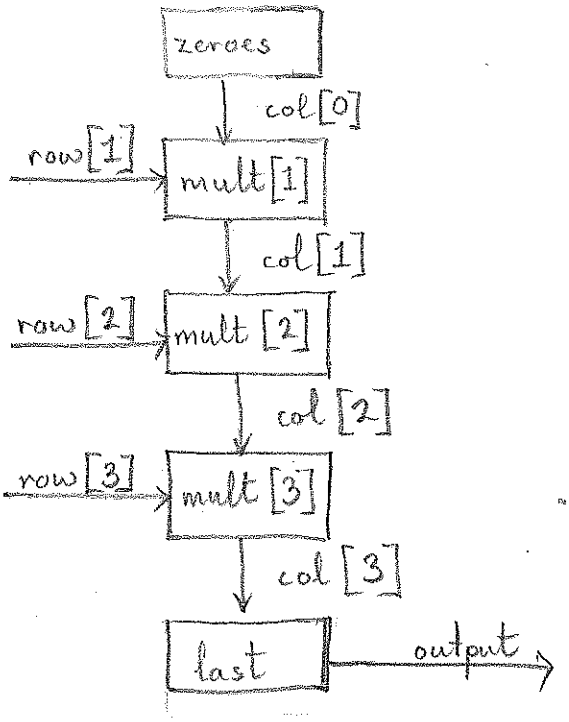
$$receiver \triangleq (wire?z:M \rightarrow (wire!ACK \rightarrow output!z \rightarrow receiver \mid wire!NACK \rightarrow receiver))$$

(5) A communications protocol is implemented as a sender and a receiver connected by a single-wire bi-directional channel; communications on this channel are regarded as local and are concealed.



$$protocol \triangleq (\underline{chan}\ wire; (sender \parallel receiver))$$

(6) A network of multipliers $mult[i:1..3]$ is designed to input the successive rows of a matrix along channels $row[1..3]$ and transmit along an "output" channel the scalar product of each row multiplied by a fixed vector $v[1..3]$. The overall structure of the network is as shown in the following diagram.



Each process mult[i] inputs a value x from row[i], multiplies it by v[i], adds the product to a partial sum y which it has input from col[i-1], and outputs the result on col[i]. These actions are then repeated.

$$mult[i:1..3] \triangleq (row[i]?x:NAT \rightarrow col[i-1]?y:NAT \rightarrow col[i]!(v[i]*x+y) \rightarrow mult[i])$$

The two other processes look after the boundary conditions

$$zeroes \triangleq (col[0]!0 \rightarrow zeroes)$$

$$last \triangleq (col[3]?y:NAT \rightarrow output!y \rightarrow last)$$

These processes can be assembled in a network

$$network \triangleq (zeroes \,||\, mult[1] \,||\, mult[2] \,||\, mult[3] \,||\, last)$$

where $W = \{col[0]\}$; $Z = \{col[3], output\}$;

$X_i = \{col[i-1], col[i], row[i]\}$; $Y_i = X_i - \bigcup_{j=1}^{i} Y_j$

Finally internal communication can be localised.

$$multiplier \triangleq (\underline{chan}\ col[0..3];\ network).$$

2. Inference Rules for partial correctness of processes. §2.

If P is a process expression and R is an assertion, we define "P sat R" as meaning that the assertion R is true before and after every communication by P. In general, R will be a predicate containing constants, variables, expressions and logical connectives. If a variable occurs free in both P and R, then it is understood as the same variable, and "P sat R" must be true for all values it can take. /// Note, we do not allow process names to appear in assertions.

new line

We intend that channel names should appear as free variables of R; they denote the sequence of values communicated by P along that channel up to some moment in time. For example, we write "$s \leq t$" to mean that the sequence t begins with s, i.e.

$$s \leq t \quad =_{df} \quad \exists u\,(s\,u = t)$$

Now the assertion "wire ≤ input" means that the sequence of values transmitted along the wire is nothing but a copy of some initial segment of what has been transmitted along the input channel. This assertion is always true of the copier process, so we can validly claim that "copier sat wire ≤ input". Similarly, we can claim that "recopier sat output ≤ wire" and that "protocol sat output ≤ input". We shall give a set of inference rules for proofs of the validity of such claims in the remainder of this section.

But first we define some useful operators on sequences.

(1) If s is a sequence and x is a message value, $x^\frown s$ is the sequence whose first message is x and whose remainder is s.

(2) $\#s$ is the length of the sequence s; thus for example

$$\text{copier sat}(\#\text{input} \leq \#\text{wire} + 1)$$

(3) s_i (for $i \in \{1..\#s\}$) is the value of the i^{th} message of s; thus

$$\text{multiplier sat}\ (\forall i{:}\,NAT.\ 1 \leq i \leq \#\text{output} \Rightarrow \text{output}_i = \sum_{j=1}^{3} v[j] * \text{row}[j]_i\ )$$

Note: free channel names in P and R are regarded as bound in "P sat R". This is because "P sat R" has to be true for all possible sequences of messages communicated by P along those channels.

2.1 Inference rules.

Let Γ and Δ be lists of predicates, including possibly predicates of the form "P sat R". Then an inference is a formula of the form "$\Gamma \vdash \Delta$", which means that all the predicates of Δ can be validly inferred from the set of assumptions listed in Γ. An inference rule has the form:

$$\frac{\Gamma 1 \vdash \Delta 1}{\Gamma 2 \vdash \Delta 2} \ ;$$

which means that whenever the inference above the line is valid, the inference below the line is valid too. We shall take for granted the familiar inference rules for natural deduction, for example, if x is not free in Γ, then

$$\frac{\Gamma \vdash R}{\Gamma \vdash \forall x \in M.\ R} \qquad (\forall\text{-introduction})$$

(1) If none of the free channel names of T are free in Γ, then

$$\frac{\Gamma \vdash T}{\Gamma \vdash P \text{ sat } T} \qquad (\text{triviality})$$

The inference above the line states that T is always true (on assumptions Γ). It follows that T is true before and after every communication of P.

Example: $\vdash wire \leq wire$
Therefore $\vdash copier$ sat $wire \leq wire$

(2) If none of the free channel names of $R \Rightarrow S$ are free in Γ, then

$$\frac{\Gamma \vdash P \text{ sat } R,\ R \Rightarrow S}{\Gamma \vdash P \text{ sat } S.} \qquad (\text{consequence})$$

If R is invariantly true of P, and whenever R is true so is S, then S is also invariantly true of P. Example: Let Γ = copier sat $wire \leq input$

Then $\Gamma \vdash copier$ sat $wire \leq input$, $(wire \leq input \Rightarrow x^\frown wire \leq x^\frown input)$

and therefore $\Gamma \vdash copier$ sat $x^\frown wire \leq x^\frown input$.

(3) $$\frac{\Gamma \vdash P \text{ sat } R,\ P \text{ sat } S}{\Gamma \vdash P \text{ sat } (R \& S)} \qquad (\text{conjunction})$$

If R is always true of P and so is S, then so is (R&S).

(4) The process STOP always leaves all channels empty.
Let $R_{<>}$ be formed from R by replacing all channel names by the constant empty sequence $<>$.

$$\frac{\Gamma \vdash R_{<>}}{\Gamma \vdash \text{STOP } \underline{\text{sat}}\ R} \qquad \text{(emptyness)}$$

Example $\vdash <> \leq <>$
Therefore $\vdash$ STOP $\underline{\text{sat}}$ wire $\leq$ input.
Similarly $\vdash$ STOP $\underline{\text{sat}}$ $((3^\frown(4^\frown c)) \leq <3,4> \ \&\ d \leq e)$

(5) The process $(c!e \to P)$ behaves like P, except that the sequence of communications along channel c has the value of e prefixed to it.
Let $R^c_{e^\frown c}$ be formed from R by replacing all occurrences of the channel name c by the expression $e^\frown c$.

$$\frac{\Gamma \vdash R_{<>},\ P \ \underline{\text{sat}}\ R^c_{e^\frown c}}{\Gamma \vdash (c!e \to P)\ \underline{\text{sat}}\ R} \qquad \text{(output)}$$

Example: $\vdash (3^\frown <>) \leq <3,4> \ \&\ <> \leq <>$, STOP $\underline{\text{sat}}$ $(3^\frown(4^\frown c)) \leq <3,4> \ \&\ d \leq e$
therefore $\vdash (c!4 \to \text{STOP})\ \underline{\text{sat}}\ ((3^\frown c) \leq <3,4> \ \&\ d \leq e)$.
similarly $\vdash (c!3 \to c!4 \to \text{STOP})\ \underline{\text{sat}}\ (c \leq <3,4> \ \&\ d \leq e)$

Note. If c is a subscripted channel name from an array $d[M]$, then $R^{d[f]}_{e^\frown d[f]}$ is taken to be $R^d_{\lambda i: M.\ \text{if } i=f \text{ then } e^\frown d[f] \text{ else } d[i]}$, where i is a fresh variable (not free in f, or e). This applies in the next rule too.

(6) The command $(c?x:M \to P)$ is like $(c!x \to P)$, except that it is prepared for communication of any value of x drawn from the set M. It must therefore satisfy its invariant for all such values.
Let v be a fresh variable which is not free in P, R or c.

$$\frac{\Gamma \vdash R_{<>},\ \forall v \in M.\ P^x_v \ \underline{\text{sat}}\ R^c_{v^\frown c}}{\Gamma \vdash (c?x:M \to P)\ \underline{\text{sat}}\ R.} \qquad \text{(input)}$$

Example. Let Γ = copier $\underline{\text{sat}}$ (wire $\leq$ input)
Then $\Gamma \vdash$ $<> \leq v^\frown <>$, copier $\underline{\text{sat}}$ $(v^\frown \text{wire} \leq v^\frown \text{input})$ (proved before)
$\therefore \Gamma \vdash <> \leq <>, \forall v \in M.\ (\text{wire}!v \to \text{copier})\ \underline{\text{sat}}\ (\text{wire} \leq v^\frown \text{input})$ (output, $\forall$ intr)
$\therefore \Gamma \vdash (\text{input}?x:M \to \text{wire}!x \to \text{copier})\ \underline{\text{sat}}\ (\text{wire} \leq \text{input})$. (input)

Suggestion: read this proof backwards.

(7) The process $(P|Q)$ behaves like P or like Q. It satisfies an invariant whenever both alternatives satisfy it.

$$\frac{\Gamma \vdash P \underline{sat}\ R,\ Q \underline{sat}\ R}{\Gamma \vdash (P|Q) \underline{sat}\ R} \qquad \text{(alternative)}$$

An example will be given later.

(8) Let X be a list of channels, including all channels mentioned in R and let Y be a list of channels, including all channels mentioned in S. Suppose that P satisfies R and Q satisfies S. Then, when they run in parallel, we claim that $(P_X||_Y Q)$ satisfies the conjunction $(R\&S)$. Clearly, communication by P on any channel of the set $(X-Y)$ satisfies R, and does not affect S, because S does not mention any of these channels. Similarly, communication by Q on channels of $(Y-X)$ preserves the truth of R as well as S. But communication on a channel of $X \cap Y$ which connects P with Q requires simultaneous participation of both P and Q; P ensures that it maintains the truth of R and Q ensures that it maintains the truth of S. So it must satisfy both $R\&S$.

$$\frac{\Gamma \vdash P \underline{sat}\ R,\ Q \underline{sat}\ S}{\Gamma \vdash (P_X||_Y Q) \underline{sat}\ (R\&S)} \qquad \text{(parallelism)}$$

Example: $\vdash$ copier $\underline{sat}$ wire $\leq$ input, recopier $\underline{sat}$ output $\leq$ wire (assume)

therefore $\vdash (\text{copier}_X||_Y \text{recopier})$ $\underline{sat}$ (output $\leq$ wire & wire $\leq$ input) (parallelism)

and so $\vdash (\text{copier}_X||_Y \text{recopier})$ $\underline{sat}$ output $\leq$ input (consequence)

(where $X = \{\text{input}, \text{wire}\}$ and $Y = \{\text{output}, \text{wire}\}$)

(9) Let R be an assertion which does not mention any channel of the list L. Suppose P $\underline{sat}$ R; then the truth of R is unaffected by communications on any of the channels of L; and remains true even when all such communications are concealed.

$$\frac{\Gamma \vdash P \underline{sat}\ R}{\Gamma \vdash (\underline{chan}\ L; P) \underline{sat}\ R} \qquad \text{(chan)}$$

Example. $\vdash (\text{copier}\ {}_X\|_Y\ \text{recopier})$ sat output ≤ input (already shown)

Therefore $\vdash (\text{chan wire}; \text{copier}\ {}_X\|_Y\ \text{recopier})$ sat output ≤ input.

(10) Consider a process name p, defined recursively by the equation $p \triangleq P$. We allow such definitions to appear in the list of assumptions of an inference. Suppose we wish to prove that "p sat R". As always, it is necessary to show that R is true of empty sequences. Also it is necessary to show that the expression defining the behaviour of p satisfies R. But in proving that P sat R, we will encounter recursive occurrences of the name p. In order to complete the proof, we will need to know something about the behaviour of p. The inference rule given below allows us to assume about p the very thing that we are trying to prove about it, namely p sat R.

If p is not free in Γ,

$$\frac{\Gamma \vdash R_{\langle\rangle} \quad ; \quad \Gamma, p\ \text{sat}\ R \vdash P\ \text{sat}\ R}{\Gamma,\ p \triangleq P \vdash p\ \text{sat}\ R} \qquad \text{(recursion)}$$

Example: Let P stand for $(\text{input}?x{:}\text{NAT} \to \text{wire}!x \longrightarrow \text{copier})$

$\vdash \langle\rangle \le \langle\rangle$ (theorem)

copier sat wire ≤ input $\vdash$ P sat wire ≤ input (already proved)

Therefore copier $\triangleq$ P $\vdash$ copier sat wire ≤ input. (recursion).

This rule extends to process array definitions. If q is not free in Γ,

$$\frac{\Gamma \vdash (\forall x \in M.\ S_{\langle\rangle}) \quad ; \quad \Gamma, (\forall x \in M.\ q[x]\ \text{sat}\ S) \vdash (\forall x \in M.\ Q\ \text{sat}\ S)}{\Gamma,\ q[x{:}M] \triangleq Q \vdash (\forall x \in M.\ q[x]\ \text{sat}\ S).}$$

and also to longer lists of equations, for example: if both p and q are not free in Γ

$$\frac{\Gamma \vdash R_{\langle\rangle}, (\forall x \in M.\ S_{\langle\rangle}) \quad ; \quad \Gamma, p\ \text{sat}\ R, (\forall x \in M.\ q[x]\ \text{sat}\ S) \vdash P\ \text{sat}\ R, (\forall x \in M.\ Q\ \text{sat}\ S)}{\Gamma,\ p \triangleq P,\ q[x{:}M] \triangleq Q \vdash p\ \text{sat}\ R, (\forall x \in M.\ q[x]\ \text{sat}\ S)}$$

split here and align Γ

Note. The inference rules for recursion depend on two subsidiary inferences, here separated by semicolon.

2.2. Examples.

(1) Let $\Delta 1$ be the list of definitions.

$$sender \triangleq (input?x{:}M \to q[x]),$$
$$q[x{:}M] \triangleq (wire!x \to (wire?y{:}\{ACK\} \to sender \mid wire?y{:}\{NACK\} \to q[x]))$$

Let f be a function from $(M \cup \{ACK, NACK\})^*$ to M^*, such that the value of $f(s)$ is obtained from s by cancelling all occurrences of ACK, and all consecutive pairs $\langle x, NACK\rangle$, e.g.

$$f(\langle x, NACK, y, ACK\rangle) = y$$

Thus $f(\langle\rangle) = \langle\rangle$, $f(\langle x\rangle) = \langle x\rangle$, $f(x^\frown ACK^\frown wire) = x^\frown f(wire)$, and $f(x^\frown NACK^\frown wire) = f(wire)$.

We want to prove that $\Delta 1 \vdash sender \ \underline{sat}\ f(wire) \le input$.

Proof. Let $A1$ be the list of predicates

$$sender \ \underline{sat}\ f(wire) \le input, \quad \forall x \in M.\ q[x] \ \underline{sat}\ f(wire) \le x^\frown input.$$

We shall prove the stronger lemma that $\Delta 1 \vdash A1$ by rule (recursion). It is easy to check [illegible]. The main part of the proof is displayed in table 1. (the first subsidiary inference that $\vdash f(\langle\rangle) \le \langle\rangle$, $\forall x \in M.\ f(\langle\rangle) \le x^\frown\langle\rangle$.)

(2) Let $\Delta 2$ be the definition.

$$receiver \triangleq (wire?x{:}M \to (wire!ACK \to output!x \to receiver \mid wire!NACK \to receiver))$$

We wish to prove that $\Delta 2 \vdash receiver \ \underline{sat}\ output \le f(wire)$

The proof is left as an excercise.

(3) Let $\Delta 3$ be the definition $protocol = (\underline{chan}\ wire;\ sender \underset{X}{\|}_{Y} receiver)$

We wish to prove that $\Delta 1, \Delta 2, \Delta 3 \vdash protocol \ \underline{sat}\ output \le input$.

(1) $sender \ \underline{sat}\ f(wire) \le input$ (already proved from $\Delta 1$)

(2) $receiver \ \underline{sat}\ output \le f(wire)$ (" " " $\Delta 2$)

(3) $(sender \| receiver) \ \underline{sat}\ (f(wire) \le input \ \&\ output \le f(wire))$ (parallelism (1), (2))

(4) $(sender \| receiver) \ \underline{sat}\ output \le input$ (consequence (3), trans $\le$)

(5) $(\underline{chan}\ wire;\ sender \| receiver) \ \underline{sat}\ output \le input$ (chan (4))

(6) $protocol \ \underline{sat}\ output \le input$ ($\Delta 3$, recursion (5), $\langle\rangle \le \langle\rangle$)

Prove the second subsidiary inference :

$$sender \ \underline{sat}\ f(wire) \le input \ , \quad \forall x \in M.\ q[x] \ \underline{sat}\ f(wire) \le x \hat{} input$$

$$\vdash (input?x:M \to q[x]) \ \underline{sat}\ f(wire) \le input \ ,$$

$$\forall x \in M.\ (wire!x \to (wire?y:\{ACK\} \to sender \mid wire?y:\{NACK\} \to q[x])) \ \underline{sat}\ f(wire) \le x \hat{} input$$

(1)	$sender \ \underline{sat}\ f(wire) \le input$	(assumption)
(2)	$\forall x \in M.\ q[x] \ \underline{sat}\ f(wire) \le x \hat{} input$	(assumption)
(3)	$f(\langle\rangle) \le \langle\rangle$	(def f)
(4)	$(input?x:M \to q[x]) \ \underline{sat}\ f(wire) \le input$	(input (2), (3))
(5)	$x \in M \Rightarrow q[x] \ \underline{sat}\ f(wire) \le x \hat{} input$	($\forall$-elim (2))
(6)	$x \in M$	(assumption)
(7)	$q[x] \ \underline{sat}\ f(wire) \le x \hat{} input$	($\Rightarrow$-elim (5), (6))
(8)	$f(wire) \le input \Rightarrow f(x \hat{} ACK \hat{} wire) \le x \hat{} input$	(def f)
(9)	$f(wire) \le x \hat{} input \Rightarrow f(x \hat{} NACK \hat{} wire) \le x \hat{} input$	(def f)
(10)	$sender \ \underline{sat}\ f(x \hat{} ACK \hat{} wire) \le x \hat{} input$	(consequence (1), (8))
(11)	$\forall v \in \{ACK\}.\ sender \ \underline{sat}\ f(x \hat{} v \hat{} wire) \le x \hat{} input$	($\forall$-int (10))
(12)	$q[x] \ \underline{sat}\ f(x \hat{} NACK \hat{} wire) \le x \hat{} input$	(consequence (7), (9))
(13)	$\forall v \in \{NACK\}.\ q[x] \ \underline{sat}\ f(x \hat{} v \hat{} wire) \le x \hat{} input$	($\forall$-int (12))
(14)	$f(\langle x \rangle) \le \langle x \rangle$	(def f)
(15)	$(wire?y:\{ACK\} \to sender) \ \underline{sat}\ f(x \hat{} wire) \le x \hat{} input$	(input (11), (14), y not free in sender)
(16)	$(wire?y:\{NACK\} \to q[x]) \ \underline{sat}\ f(x \hat{} wire) \le x \hat{} input$	(input (13), (14), y not free in q[x])
(17)	$(wire?y:\{ACK\} \to sender \mid wire?y:\{NACK\} \to q[x]) \ \underline{sat}\ f(x \hat{} wire) \le x \hat{} input$	(alternative (15), (16))
(18)	$f(\langle\rangle) \le \langle x \rangle$	(def f)
(19)	$(wire!x \to (wire?y:\{ACK\} \to sender \mid wire?y:\{NACK\} \to q[x])) \ \underline{sat}\ f(wire) \le x \hat{} input$	(output (17), (18))
(20)	$x \in M \Rightarrow (19)$	($\Rightarrow$-int (6), (19))
(21)	$\forall x \in M.\ (19)$	($\forall$-int (20))

The desired inference is just (1), (2) $\vdash$ (4), (21).

Table 1

3. Validity of the inference system.

The validity of an inference system is established by defining a mathematical model (or interpretation) of the formulae of the system, and proving that the inference rules correspond to mathematically provable facts about the model. For the predicate calculus, an interpretation of a formula is known as an environment, i.e., a mapping from free variables of the formula onto points of some appropriate mathematical structure. For programs expressed in a programming language, it is desirable that an interpretation should bear some resemblance to the behaviour of an intended implementation of the program. The potential behaviour of a communicating process is described by giving the set of all its possible traces, i.e., a prefix-closed set of sequences of communications.

3.1. Prefix closures.

Let A be the set of all (possible) communications, that is, all pairs "c.m" where c is a channel name and m is a message value. For any subset B of A, B^* is defined as the set of all finite sequences constructed from elements of B. A prefix closure is any subset P of A^* which satisfies the two conditions

$$\langle\rangle \in P$$

$$st \in P \Rightarrow s \in P \quad \text{for all } s,t \text{ in } A^*$$

From this it follows that:

$\{\langle\rangle\}$ and A^* are prefix closures

If P is a prefix closure, then $\{\langle\rangle\} \subseteq P \subseteq A^*$

If P_x is a prefix closure for all x in M, then $\bigcup_{x\in M} P_x$ and $\bigcap_{x\in M} P_x$ are also prefix closures.

Thus prefix closures form a complete lattice, and any set of recursive equations using continuous operators will have a unique least solution. In fact, all the operators we use will satisfy the stronger condition of distributing through arbitrary unions, as do the operations $\cap$ and $\cup$.

$$(\bigcup_{x\in M} P_x) \cap Q = \bigcup_{x\in M}(P_x \cap Q), \quad (\bigcup_{x\in M} P_x) \cup Q = \bigcup_{x\in M}(P_x \cup Q).$$

If P is a prefix closure, and $a \in A$, we define

$$(a \to P) = \{\langle\rangle\} \cup \{a^\frown s \mid s \in P\}.$$

Theorem. $(a \to P)$ is a prefix closure.

Proof. By inspection, $\langle\rangle \in (a \to P)$

Let $st \in (a \to P)$. If $st = \langle\rangle$ then $s = \langle\rangle$, so $s \in (a \to P)$.

If $s \neq \langle\rangle$ then $s = a^\frown s'$ for some s', and $st = a^\frown s't$ where $s't \in P$.

Since P is prefix-closed, $s' \in P$. Hence $a^\frown s'$, which equals s, is in $(a \to P)$.

Theorem. $(a \to \bigcup_{x\in M} P_x) = \bigcup_{x\in M} (a \to P_x)$ (distributivity of $\to$)

Proof.

$$\begin{aligned} LHS &= \{\langle\rangle\} \cup \{a^\frown s \mid s \in \bigcup_{x\in M} P_x\} && \text{(def } \to\text{)} \\ &= \{\langle\rangle\} \cup \bigcup_{x\in M} \{a^\frown s \mid s \in P_x\} && \text{(set theory)} \\ &= \bigcup_{x\in M} (\{\langle\rangle\} \cup \{a^\frown s \mid s \in P_x\}) && (\text{ " }) \\ &= RHS && \text{(def } \to\text{)} \end{aligned}$$

If C is a set of channel names, and s is in A^*, then we define $s\backslash C$ as the sequence formed from s by omitting all communications along any of the channels of C. Thus:

$$\langle\rangle\backslash C = \langle\rangle, \quad (c.m^\frown s)\backslash C = \begin{cases} c.m^\frown(s\backslash C) & \text{if } c \in C \\ s\backslash C & \text{if } c \notin C \end{cases}$$

$$st\backslash C = (s\backslash C)(t\backslash C), \quad u\backslash C = st \Rightarrow \exists v\, w.\ u = vw \ \&\ v\backslash C = s \ \&\ w\backslash C = t$$

If P is a prefix closure, then we define

$$P\backslash C = \{s\backslash C \mid s \in P\}, \quad P/C = \{s \mid s\backslash C \in P\}$$

Theorem. $P\backslash C$ and P/C are prefix closures, and they are distributive in P.

Proofs are omitted; they are similar to the previous proof.

If P contains no communication along any channel of C, then P/C is the set of traces formed by interleaving a trace of P with an arbitrary sequence of communications on the channels of C, which are, as it were, ignored by P. Let P communicate only on channels in X, and Q communicate only on channels in Y. Then define $P\ {}_X{\wedge}_Y\ Q = (P/(Y-X)) \cap (Q/(X-Y))$. Let s be a trace of this set. It follows that $s\backslash X \in P$ and $s\backslash Y \in Q$. Thus every communication of s along any channel of X "requires" participation of P; similarly, every communication along channels of Y "requires" participation of Q; therefore communications along a common channel of $X \cap Y$ requires simultaneous participation of both of them. We use this operator to model parallel composition of processes.

Theorem. ${}_X{\wedge}_Y$ is a distributive operator. Proof trivial.

$P\backslash C$ clearly models the effect of localization of channels in C.

Denotational

3.2. Semantics of process expressions

The semantics of process expressions is defined by a function which maps an arbitrary process expression onto its meaning, namely, a prefix closure, containing all possible traces of the behaviour of the given process. But a process expression in general contains free variables and process names, and the meaning of the expression will depend on the meanings of these variables and names. So the semantic function is based on

← an environment ρ, which maps names onto their meanings; more precisely, it maps variable names onto values, process names onto prefix closures, and process array names onto arrays of prefix closures. We stipulate that its domain does not include channel names. If ρ is an environment and x is a name and v is a meaning of a sort appropriate for x, then $\rho[v/x]$ is defined as the environment which maps x to v and every other name to the same meaning as given by ρ:

$$\rho[v/x](y) = v \text{ if } y = x$$
$$= \rho(y) \text{ if } y \neq x.$$

If e is an expression, we extend the definition of ρ to let $\rho[\![e]\!]$ stand for the value that e takes when the free variables of e take the values ascribed to them by ρ. Thus, for example,

$$\rho[\![3]\!] = 3, \quad \rho[\![e+f]\!] = \rho[\![e]\!] + \rho[\![f]\!], \text{ etc.}$$

Note. ← parameters which are syntactic objects like expressions are contained in double square brackets $[\![\ ]\!]$, as is usual in denotational semantics.

Now it remains to extend further the definition of ρ to apply also to process expressions. This is done by considering separately each possible syntactic structure for the process expression P, using recursion where necessary to deal with its substructure.

(1) $\rho[\![STOP]\!] = \{\langle\rangle\}$

(2) $\rho[\![p]\!] = \rho(p)$ if p is a process name

(3) $\rho[\![p[e]]\!] = \rho(p)[\rho[\![e]\!]]$ if p is a process array name

(4) $\rho[\![c]\!] = c$ if c is a channel name

(5) $\rho[\![c[e]]\!] = c[\rho[\![e]\!]]$ if c is a channel array name

(6) $\rho[\![c!e \to P]\!] = ((\rho[\![c]\!].\rho[\![e]\!]) \to \rho[\![P]\!])$

(7) $\rho[\![c?x:M \to P]\!] = \{\langle\rangle\} \cup \bigcup_{v \in \rho[\![M]\!]} ((\rho[\![c]\!].v) \to (\rho[v/x])[\![P]\!])$

(8) $\rho[\![P|Q]\!] = \rho[\![P]\!] \cup \rho[\![Q]\!]$

(9) $\rho[\![P \underset{X}{\|}_{Y} Q]\!] = \rho[\![P]\!] \underset{\rho[\![X]\!]}{\cap} \rho[\![Q]\!]$

so that $\rho[\![P]\!]$ is the prefix closure denoted by P when the free variables of P take the values ascribed by ρ.

(10) $\rho[\![\underline{\text{chan}}\ X;\ P]\!] = \rho[\![P]\!] \setminus \rho[\![X]\!]$

3.3 Semantics of inference rules.

Let s be a sequence of communications. We define $ch(s)$ as the function which maps every channel name "c" onto the sequence of messages whose communication along c is recorded in s. Thus if

$s = \langle input.27,\ wire.27,\ input.0,\ wire.0,\ input.3\rangle$

then $ch(s)(input) = \langle 27, 0, 3\rangle$

$ch(s)(wire) = \langle 27, 0\rangle$

$ch(s)(c) = \langle\rangle$ for $c \neq wire$ and $c \neq input$.

In general, $ch(\langle\rangle) = \lambda c.\ \langle\rangle$

$ch(c.m^\frown s) = ch(s)[(m^\frown(ch(s)(c)))/c]$

If ρ is an environment (which does not ascribe values to channel names), then $(\rho + ch(s))$ is an environment in which channel names have the values ascribed to them by $ch(s)$; i.e.,

$(\rho + ch(s))[\![x]\!] = ch(s)(x)$ if x is a channel name
$= ch(s)(c.\rho[\![e]\!])$ if x is a subscripted channel name $c[e]$
$= \rho[\![x]\!]$ otherwise

This is the environment which is used to calculate the truth or falsity of an assertion, R, according to the normal semantics of the predicate calculus e.g.

$(\rho + ch(s))[\![R \,\&\, S]\!] = ((\rho + ch(s))[\![R]\!]) \,\&\, ((\rho + ch(s))[\![S]\!])$,

$(\rho + ch(s))[\![input \leq wire]\!] = ch(s)(input) \leq ch(s)(wire)$,

$(\rho + ch(s))[\![\forall x \in M.\ R]\!] = \forall v.\ v \in \rho[\![M]\!] \Rightarrow (\rho[v/x] + ch(s))[\![R]\!]$

The predicate "$P\ \underline{\text{sat}}\ R$" states that all traces of the process P satisfy the predicate R, i.e.

$$\rho[\![P\ \underline{\text{sat}}\ R]\!] = \forall s\ \ s \in \rho[\![P]\!] \Rightarrow (\rho + ch(s))[\![R]\!].$$

and $\rho[\![\forall x \in M.\ P\ \underline{\text{sat}}\ R]\!] = \forall v.\ v \in \rho[\![M]\!] \Rightarrow \rho[v/x][\![P\ \underline{\text{sat}}\ R]\!]$.

If T is a predicate containing free channel names, we similarly define $\rho(T) = \forall s\, (\rho + ch(s))[\![T]\!]$, i.e., T has to be true for all possible sequences of values passing along the channels.

We now need to define the semantics of a possibly recursive process definition $p \triangleq P$. We define $\rho[\![p \triangleq P]\!]$ as being true if and only if the value ascribed by ρ to the name p is indeed the intended recursively defined process, that is, the least solution (in the domain of prefix closures) to the equation $p \triangleq P$. Since all the operators from which P is constructed are continuous, this can be computed as the union of a series of successive approximations, $a_0, a_1, a_2, \ldots$, where

$$a_0 = \rho[\![STOP]\!]$$

$$a_{i+1} = (\rho[a_i/p])[\![P]\!].$$

(Here a_i allows recursion only to depth i, after which it stops)

$$\rho[\![p \triangleq P]\!] = \left(\rho(p) = \bigcup_{i \geq 0} a_i\right)$$

This technique applies also to process array definitions such as $q[x:M] \triangleq Q$. Here each approximation a_i is itself a process array, and so is defined using λ-notation

$$a_0 = \lambda v : M.\, \rho[\![STOP]\!]$$

(This is the array such that $a_0[v] = \rho[\![STOP]\!]$ for all v in M.)

$$a_{i+1} = \lambda v : M.\, \rho[a_i/q][v/x][\![Q]\!]$$

$$\rho[\![q[x:M] \triangleq Q]\!] = \left(\rho(q) = \lambda v : M.\ \bigcup_{i \geq 0} (a_i[v])\right)$$

If $R_1, R_2, \ldots R_n$ is a list of predicates, then

$$\rho[\![R_1, R_2, \ldots, R_n]\!] = \rho[\![R_1]\!] \,\&\, \rho[\![R_2]\!] \,\&\, \ldots \,\&\, \rho[\![R_n]\!]$$

An inference is valid if and only if its antecedent logically implies its consequent, in all possible environments.

$$\Gamma \vdash R \;=_{df}\; \forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \rho[\![R]\!]$$

e.g. $(p \triangleq P \vdash p \underline{sat}\ R) = \left(\forall \rho.\ \rho(p) = \bigcup_{i \geq 0} a_i \Rightarrow \forall s\ s \in \rho[\![p]\!] \Rightarrow (\rho + ch(s))[\![R]\!]\right)$

An inference rule $\frac{A}{B}$ is valid if and only if $\rho[\![B]\!]$ can be validly deduced from the assumption $\rho[\![A]\!]$. This needs to be established for each inference rule of our system.

3.4 Proofs.

First we prove some simple lemmas about environments. They can be proved by induction on the structure of the formula R.

(a) If R^{x}_{e} is formed from R by replacing every free occurrence of x by a free occurrence of e, then (since e contains no channel names):

$$(\rho + ch(s))[\![R^{x}_{e}]\!] = (\rho + ch(s))[\rho[\![e]\!]/x]\,R.$$

(b) If $R_{<>}$ is formed from R by replacing all channel names by $<>$

$$(\rho + ch(<>))[\![R]\!] = \rho[\![R_{<>}]\!].$$

(c) If c is a channel name and e is an expression (containing no channel names)

$$(\rho + ch(s))[\![R^{c}_{e\hat{}c}]\!] = (\rho + ch((c.\rho[\![e]\!])\hat{}s))[\![R]\!]$$

and $$(\rho + ch(s))[\![R^{d[f]}_{e\hat{}d[f]}]\!] = (\rho + ch((d[\rho[\![f]\!].\rho[\![e]\!])\hat{}s))[\![R]\!]$$

(d) If the set of channel names in $\rho[\![X]\!]$ does not contain any of the channel names mentioned in R, then.

$$(\rho + ch(s))[\![R]\!] = (\rho + ch(s \setminus \rho[\![X]\!]))[\![R]\!]$$

(since $ch(s)(c) = ch(s \setminus C)(c)$ whenever $c \tilde{\in} C$.)

To prove consistency of the inference rules with the semantics of processes (interpreted as prefix closure), it is necessary to show that each of them can be proved as a theorem, using only the definitions explained in the earlier sections of this chapter, e.g. for a rule of the form

$$\frac{\Gamma \vdash A}{\Delta \vdash B}$$

We have to prove that the top inference implies the bottom one, i.e.

$$(\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \rho[\![A]\!]) \Rightarrow (\forall \rho.\ \rho[\![\Delta]\!] \Rightarrow \rho[\![B]\!]).$$

(1) Triviality. Suppose T contains no channel names free in Γ, and assume $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \rho[\![T]\!]$. Then

$$\begin{aligned}
\rho[\![\Gamma]\!] &\Rightarrow \forall s.\ (\rho + ch(s))[\![T]\!] && \text{(definition of } \rho[\![T]\!]) \\
&\Rightarrow \forall s.\ s \in \rho[\![P]\!] \Rightarrow (\rho + ch(s))[\![T]\!] \\
&\Rightarrow \rho[\![P \text{ sat } T]\!].
\end{aligned}$$

(2) Consequence. Assume $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow (\rho[\![P \text{ sat } R]\!] \ \&\ \rho[\![R \Rightarrow S]\!])$.

Since channel names mentioned in R or S are not free in Γ,

$$\begin{aligned}
\rho[\![\Gamma]\!] &\Rightarrow (\forall s.\ s \in \rho[\![P]\!] \Rightarrow (\rho + ch(s))[\![R]\!]) \ \&\ (\forall s.\ (\rho + ch(s))[\![R]\!] \Rightarrow (\rho + ch(s))[\![S]\!]) \\
&\Rightarrow \forall s.\ s \in \rho[\![P]\!] \Rightarrow (\rho + ch(s))[\![S]\!] \\
&\Rightarrow \rho[\![P \text{ sat } S]\!].
\end{aligned}$$

(3) Conjunction. Trivial.

(4) Emptiness. Assume $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \rho[\![R_{\langle\rangle}]\!]$. Then

$$\begin{aligned}
\rho[\![\Gamma]\!] &\Rightarrow \rho[\![R_{\langle\rangle}]\!] \\
&\Rightarrow (\rho + ch(\langle\rangle))[\![R]\!] && \text{(lemma (2))} \\
&\Rightarrow \forall s.\ s = \langle\rangle \Rightarrow (\rho + ch(s))[\![R]\!] \\
&\Rightarrow \forall s.\ s \in \rho[\![\text{STOP}]\!] \Rightarrow (\rho + ch(s))[\![R]\!] && \text{(def STOP)} \\
&\Rightarrow \rho[\![\text{STOP sat } R]\!].
\end{aligned}$$

(5) Output. Suppose $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow (\rho[\![R_{\langle\rangle}]\!] \ \&\ \rho[\![P \underline{sat} R^c_{e\hat{}c}]\!])$. Given ρ such that $\rho[\![\Gamma]\!]$, then $\rho[\![R_{\langle\rangle}]\!]$ and $\rho[\![P \underline{sat} R^c_{e\hat{}c}]\!]$. Thus

$$s \in \rho[\![c!e \to P]\!] \Rightarrow (s = \langle\rangle \vee s = \rho[\![c]\!].\rho[\![e]\!]\hat{}t) \quad \text{(for some } t \text{ in } \rho[\![P]\!])$$

In the first case,

$$s = \langle\rangle \Rightarrow ((\rho + ch(s))[\![R]\!] = \rho[\![R_{\langle\rangle}]\!]) \quad \text{(lemma (2))}$$
$$\Rightarrow (\rho + ch(s))[\![R]\!] \quad \text{(by } \rho[\![R_{\langle\rangle}]\!]\text{)}.$$

In the second case,

$$s = \rho[\![c]\!].\rho[\![e]\!]\hat{}t \Rightarrow ((\rho + ch(s))[\![R]\!] = (\rho + ch(\rho[\![c]\!].\rho[\![e]\!]\hat{}t))[\![R]\!])$$
$$\Rightarrow ((\rho + ch(s))[\![R]\!] = (\rho + ch(t))[\![R^c_{e\hat{}c}]\!]) \quad \text{(lemma (3))}$$
$$\Rightarrow (\rho + ch(s))[\![R]\!] \quad \text{(by } \rho[\![P \underline{sat} R^c_{e\hat{}c}]\!]\text{)}.$$

So $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \forall s.\ s \in \rho[\![c!e \to P]\!] \Rightarrow (\rho + ch(s))[\![R]\!]$, i.e.

$$\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \rho[\![(c!e \to P) \underline{sat} R]\!].$$

(6) Input. Assume $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow (\rho[\![R_{\langle\rangle}]\!] \ \&\ \rho[\![\forall v \in M.\ P^x_v \underline{sat} R^c_{v\hat{}c}]\!])$.

Given ρ such that $\rho[\![\Gamma]\!]$, then $\rho[\![R_{\langle\rangle}]\!]$ and $\rho[\![\forall v \in M.\ P^x_v \underline{sat} R^c_{v\hat{}c}]\!]$, i.e. $\forall u.\ u \in \rho[\![M]\!] \Rightarrow \rho[u/v][\![P^x_v \underline{sat} R^c_{v\hat{}c}]\!]$. Thus

$$s \in \rho[\![c?x:M \to P]\!] \Rightarrow s \in (\{\langle\rangle\} \cup \bigcup_{u \in \rho[\![M]\!]} (\rho[\![c]\!].u \to \rho[u/x][\![P]\!]))$$
$$\Rightarrow s = \langle\rangle \vee s = \rho[\![c]\!].u\hat{}t \quad \text{(for some } u \in \rho[\![M]\!] \text{ and } t \in \rho[u/x][\![P]\!]).$$

Let us only check the second case:

$$s = \rho[\![c]\!].u\hat{}t \Rightarrow (\rho + ch(s))[\![R]\!] = (\rho + ch(\rho[\![c]\!].u\hat{}t))[\![R]\!]$$
$$\Rightarrow (\rho + ch(s))[\![R]\!] = (\rho[u/v] + ch(\rho[u/v][\![c]\!].\rho[u/v][\![v]\!]\hat{}t))[\![R]\!]$$
(since v is not free in R and c)
$$\Rightarrow (\rho + ch(s))[\![R]\!] = (\rho[u/v] + ch(t))[\![R^c_{v\hat{}c}]\!] \quad \text{(lemma (3))}$$

Futhermore since v is not free in P, therefore $t \in \rho[u/x][\![P]\!]$ is equivalent to $t \in \rho[u/v][\![P^x_v]\!]$. Then $(\rho[u/v] + ch(t))[\![R^c_{v\hat{}c}]\!]$ follows from the assumption $\forall u.\ u \in \rho[\![M]\!] \Rightarrow \rho[u/v][\![P^x_v \underline{sat} R^c_{v\hat{}c}]\!]$. So $(\rho + ch(s))[\![R]\!]$. Hence

$\forall s.\ s \in \rho[\![c?x:M \to P]\!] \Rightarrow (\rho + ch(s))[\![R]\!]$, provided $\rho[\![\Gamma]\!]$.

DC

(7) Alternative. Trivial.

(8) Parallelism. Assume $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow (\rho[\![P \underline{sat}\ R]\!] \,\&\, \rho[\![Q \underline{sat}\ S]\!])$. Given ρ such that $\rho[\![\Gamma]\!]$, then $\rho[\![P \underline{sat}\ R]\!]$ and $\rho[\![Q \underline{sat}\ S]\!]$. Thus

$$s \in \rho[\![P \underset{X,Y}{\|} Q]\!] \Rightarrow s \in (\rho[\![P]\!]_{\rho[\![X]\!]} \wedge_{\rho[\![Y]\!]} \rho[\![Q]\!])$$

$$\Rightarrow s \setminus (\rho[\![Y]\!] - \rho[\![X]\!]) \in \rho[\![P]\!] \,\&\, s \setminus (\rho[\![X]\!] - \rho[\![Y]\!]) \in \rho[\![Q]\!]$$

$$\Rightarrow (\rho + ch(s \setminus \rho[\![Y]\!] - \rho[\![X]\!]))[\![R]\!] \,\&\, (\rho + ch(s \setminus \rho[\![X]\!] - \rho[\![Y]\!]))[\![S]\!]$$

(by $\rho[\![P \underline{sat}\ R]\!]$ and $\rho[\![Q \underline{sat}\ S]\!]$)

$$\Rightarrow (\rho + ch(s))[\![R]\!] \,\&\, (\rho + ch(s))[\![S]\!] \quad \text{(lemma (4))}$$

$$\Rightarrow (\rho + ch(s))[\![R \& S]\!].$$

So $\forall s.\ s \in \rho[\![P \underset{X,Y}{\|} Q]\!] \Rightarrow (\rho + ch(s))[\![R \& S]\!]$, provided $\rho[\![\Gamma]\!]$.

(9) Chan. Suppose $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \rho[\![P \underline{sat}\ R]\!]$. Given ρ such that $\rho[\![\Gamma]\!]$, then $\rho[\![P \underline{sat}\ R]\!]$. Thus

$$s \in \rho[\![(\underline{chan}\ L;\ P)]\!] \Rightarrow s \in (\rho[\![P]\!] \setminus \rho[\![L]\!])$$

$$\Rightarrow s = t \setminus \rho[\![L]\!] \quad \text{(for some } t \text{ in } \rho[\![P]\!])$$

$$\Rightarrow (\rho + ch(s))[\![R]\!] = (\rho + ch(t \setminus \rho[\![L]\!]))[\![R]\!]$$

$$\Rightarrow (\rho + ch(s))[\![R]\!] = (\rho + ch(t))[\![R]\!] \quad \text{(lemma (4))}$$

$$\Rightarrow (\rho + ch(s))[\![R]\!] \quad \text{(by } \rho[\![P \underline{sat}\ R]\!]\text{)}$$

Hence $\forall s.\ s \in \rho[\![(\underline{chan}\ L;\ P)]\!] \Rightarrow (\rho + ch(s))[\![R]\!]$, provided $\rho[\![\Gamma]\!]$.

(10) Recursion. We deal only with the simple case; treatment of mutual recursion is similar but much more tedious.

Suppose $\forall \rho.\ \rho[\![\Gamma]\!] \Rightarrow \rho[\![R_{\langle\rangle}]\!]$ and $\forall \rho'.(\rho'[\![\Gamma]\!] \,\&\, \rho'[\![p \text{ sat } R]\!]) \Rightarrow \rho'[\![P \text{ sat } R]\!]$.
Given ρ such that $\rho[\![\Gamma]\!]$ and $\rho[\![p \triangleq P]\!]$, let us prove $\forall s.\ s \in \rho[\![p]\!] \Rightarrow (\rho + ch(s))[\![R]\!]$.

Since $\rho[\![p \triangleq P]\!] \Rightarrow \rho(p) = \bigcup_{i \geq 0} a_i$ and $\rho[\![p]\!] = \rho(p)$, therefore $s \in \rho[\![p]\!] \Rightarrow s \in \bigcup_{i \geq 0} a_i$.

Consider first the base case,

$s \in a_0 \Rightarrow s \in \rho[\![\text{STOP}]\!]$

$\Rightarrow s = \langle\rangle$

$\Rightarrow (\rho + ch(s))[\![R]\!] = \rho[\![R_{\langle\rangle}]\!]$ (lemma (2))

$\Rightarrow (\rho + ch(s))[\![R]\!]$ (by first premise).

Note now that $(\rho[a_i/p] + ch(s))[\![R]\!] = (\rho + ch(s))[\![R]\!]$. This is because R contains no process name, and $\rho[a_i/p]$ differs from ρ only in ascribing a different value to the process name p. Similarly $\rho[a_i/p][\![\Gamma]\!] = \rho[\![\Gamma]\!]$, since p is not free in Γ.

Now assume for arbitrary i

$\forall s.\ s \in a_i \Rightarrow (\rho + ch(s))[\![R]\!]$,

then $\forall s.\ s \in \rho[a_i/p][\![p]\!] \Rightarrow (\rho + ch(s))[\![R]\!]$ ($s \in \rho[a_i/p][\![p]\!] = s \in a_i$),

i.e. $\rho[a_i/p][\![p \text{ sat } R]\!]$.

By second premise (let $\rho' = \rho[a_i/p]$),

$\rho[a_i/p][\![P \text{ sat } R]\!]$,

i.e. $\forall s.\ s \in \rho[a_i/p][\![p]\!] \Rightarrow (\rho[a_i/p] + ch(s))[\![R]\!]$,

thus $\forall s.\ s \in a_{i+1} \Rightarrow (\rho + ch(s))[\![R]\!]$ ($a_{i+1} = \rho[a_i/p][\![P]\!]$ and $(\rho + ch(s))[\![R]\!] = (\rho[a_i/p] + ch(s))[\![R]\!]$)

Hence $\forall s.\ s \in \bigcup_{i \geq 0} a_i \Rightarrow (\rho + ch(s))[\![R]\!]$,

i.e. $\forall s.\ s \in \rho[\![p]\!] \Rightarrow (\rho + ch(s))[\![R]\!]$.

4 Conclusion.

The worst defect of the proof system described in this paper is that it deals only with partial correctness; thus it permits a proof of the properties of every trace of the behaviour of a process P, but it cannot prove that P will actually behave in the desired way. For example P may deadlock before it has completed its appointed task, or indeed before doing anything whatsoever! This is because the process STOP satisfies any satisfiable invariant whatsoever. A similar complaint is made against the theory of partial correctness of sequential programs, in which a non-terminating loop satisfies every specification.

The worst defect of the prefix closure model of the behaviour of a process is that it takes an unrealistic approach to non-determinism. For example, consider a process Q which may non-deterministically decide on a path that leads to deadlock, or may decide to behave like the process P. In our model we have to define this as

Q = STOP | P ;

but unfortunately this is identically equal to P. The same identity holds if the deadlock could happen after a certain number of communications. Of course, it is possible to implement the union process P ∪ Q for arbitrary P or Q; but only by running both P and Q in parallel, up to the point where a communication occurs which is not possible for one of them, after which that one can be discarded. But this is not the kind of non-determinism that

arises naturally in the implementation of parallel processing networks, where the choice between alternatives occurs at the moment the first communication takes place, and may therefore be time dependent.

It is hoped that the adoption of a more realistic model of non-determinism will permit the formulation of proof rules for the total correctness of processes; but much further analysis will be required. The complexity of the definitions and proofs in this paper gives little hope for an easy solution.

References

1. C. A. R. Hoare, "A Model of Communicating Sequential Processes", C. U. P. (1980).
2. C. A. R. Hoare, "Communicating Sequential Processes", C. ACM, 21, 8 (Aug. 1978).
3. C. A. R. Hoare, "Procedures and Parameters: An Axiomatic Approach" Springer Verlag: 'Lecture Notes in Math.' vol. 188 (1971).
4. R. Milner, "Synthesis of Communicating Behaviour" Springer Verlag: 'Lecture Notes in Computer Science' vol. 64 (1978
5. J. Stoy, "Denotational Semantics", MIT Press (1977).
6. K. R. Apt, N. Francez, W. P. de Roever. A Proof System for Communicating Sequential Processes. TOPLAS 2, 3. 359-385 (July 1980)