PARTIAL CORRECTNESS

OF

COMMUNICATING PROCESSES

AND

PROTOCOLS

BY

ZHOU CHAO CHEN

AND

C.A.R. HOARE

Technical Monograph PRG-20

May 1981

Oxford University Computing Laboratory
Programming Research Group
45 Banbury Road
Oxford OX2 6PE
U.K.

This monograph contains two closely related papers. The first was presented at the Second International Conference on Distributed Computing Systems in Paris on 8th April, 1981. The second was presented at the INWG/NPL Workshop on Protocol testing - towards proofs? at Teddington on 28th May, 1981.

CONTENTS

Page

PARTIAL CORRECTNESS OF COMMUNICATING SEQUENTIAL PROCESSES

Zhou Chao Chen and C.A.R. Hoare

Programming Research Group, University of Oxford.
Institute of Computing Technology, Academia Sinica, Peking.

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: 4.22 5.24

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

Introduction (0)

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 methods, which provides a basis for proofs that programs expressed in a language will meet their specification has been extended to parallel programs [6]. This paper 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.

Processes and their description (1)

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 the channel "wire".

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

$$\xrightarrow{\text{input}} \boxed{\text{copier}} \xrightarrow{\text{wire}}$$

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) <>, i.e., the empty trace, describing its behaviour before it has input anything.

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

(iii) <input.27, wire.27, input.0, wire.0, input.3> 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:

<>, <wire.3, output.3>,
<wire.27, output.27, wire.0>, 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 traces of its possible communications with its neighbours. In the case of the copier process, this set will include (for example) all traces of the form <input.m> or <input.m, output.m>, 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.

Preliminaries (1.1)

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 constituant 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.,
NAT denotes the natural numbers {0,1,2, ...}
{0..3} denotes the finite range {0,1,2,3}
{ACK,NACK} denotes a pair of acknowledgment 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$\underline{\Delta}$P, where p is a process name 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]$\underline{\Delta}$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.

Process expressions (1.2)

It remains to specify the most important features 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]$\underline{\Delta}$Q, and Q' is formed from Q by replacing each occurrence of i by the value of e, provided that this is in M.

(4) If c is a channel name (possibly subscripted) and e is an expression, and P is a process expression, then "(c!e→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 →copier),
(col[i]!(3*i+j) →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→P)" is a process which first communicates on channel c <u>any</u> 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→(wire!x→copier))
(col[i-1]?y:NAT→(col[i]!(3*x+y) →mult[i]))

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

wire?x:NAT→output!x→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.

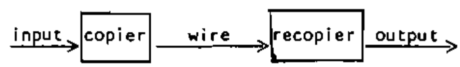
((wire!ACK→output!x→receiver)
|(wire!NACK→receiver))

Note. In future the inner brackets may be omitted, on the convention that → 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 vice versa. However any internal communication between P and Q uses one of the channels of X∩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". e.g.,

A network diagram



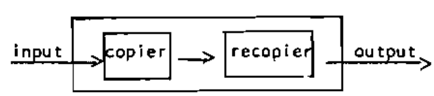
is denoted by the expression
$(copier_X||_Y recopier)$,

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 (<u>chan</u> 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 communication 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

(<u>chan</u> 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 <u>all</u> processes connected by a channel can simultaneously <u>input</u> 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.

Examples of process definitions (1.3)

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

copier $\underline{\Delta}$ (input?x:NAT → wire!x → copier).

A similar process is:

recopier $\underline{\Delta}$ (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 $\underline{\Delta}$ (input?y:M → q[y]).

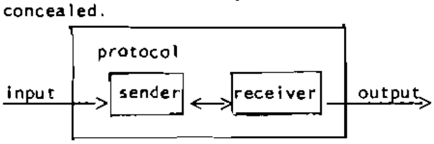
(3) The process q[x] (for any x in M) first transmits the value x along the channel named "wire"; it then inputs from the wire either an ACK signal or a NACK signal. In the first case, its subsequent behaviour is the same as that of the sender. In the other case, it transmits the message as often as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

q[x:M] $\underline{\Delta}$ (wire!x → (wire?y:{ACK} → sender
|wire?y:{NACK} → 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:

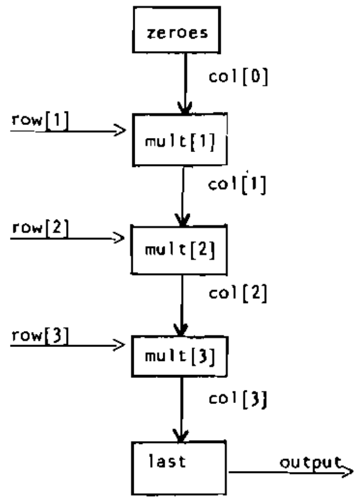
receiver $\underline{\Delta}$ (wire?z:M →
(wire!ACK → output!z → receiver
|wire!NACK → receiver))

(5) A communication 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 $\underline{\Delta}$ (chan wire; (sender||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] $\underline{\Delta}$ (row[i]?x:NAT → col[i-1]?y:NAT
→ col[i]!(v[i]*x+y) → mult[i])

The two other processes look after the boundary conditions.

zeroes $\underline{\Delta}$ (col[0]!0 → zeroes)
last $\underline{\Delta}$ (col[3]?y:NAT → output!y → last)

These processes can be assembled in a network.

network $\underline{\Delta}$ (zeroes||mult[1]||
mult[2]|| mult[3]||last)

Finally internal communication can be localised

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

Partial correctness of processes (2)

If P is a process expression and R is an assertion, we define "PsatR" 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 "PsatR" must be true for all values it can take.

Note. We do not allow process names to appear in assertions.

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≤t" to mean that the sequence t begins with s, i.e.

$$s \leq t =_{df} \exists u.(su = t)$$

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, $x\hat{\ }s$ 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

copier sat (#input ≤ #wire+1)

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

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

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.

Inference rules (2.1)

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 "Γ⊢Δ", 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) $$\frac{\Gamma \vdash T}{\Gamma \vdash P\ \underline{\text{sat}}\ T} \qquad (\text{triviality})$$

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: ⊢ wire ≤ wire.
Therefore ⊢ copier sat wire ≤ wire.

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

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 ≤ input,
then Γ ⊢ copier sat wire ≤ input,
wire ≤ input ⇒ $x\hat{\ }$wire ≤ $x\hat{\ }$input,
and therefore Γ ⊢ copier sat $x\hat{\ }$wire ≤ $x\hat{\ }$input.

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

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 $R_{\langle\rangle}$ be formed from R by replacing all channel names by the constant empty sequence $\langle\rangle$.

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

Example: ⊢ $\langle\rangle \leq \langle\rangle$.
Therefore ⊢ STOP sat wire ≤ input.
Similarly ⊢ STOP sat $((3\hat{\ }(4\hat{\ }c)) \leq \langle 3,4\rangle \& d \leq e)$

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

$$\frac{\Gamma \vdash R_{\langle\rangle},\ P\ \underline{\text{sat}}\ R^c_{e\hat{\ }c}}{\Gamma \vdash (c!e \rightarrow P)\ \underline{\text{sat}}\ R} \qquad (\text{output})$$

Example: $\vdash (3\hat{\ }\langle\rangle) \leq \langle 3,4\rangle \& \langle\rangle \leq \langle\rangle$,

$\text{STOP}\ \underline{\text{sat}}\ (3\hat{\ }(4\hat{\ }c)) \leq \langle 3,4\rangle \& d \leq e$,
therefore $\vdash (c!4 \rightarrow \text{STOP})\ \underline{\text{sat}}\ ((3\hat{\ }c) \leq \langle 3,4\rangle \& d \leq e)$,
similarly $\vdash (c!3 \rightarrow c!4 \rightarrow \text{STOP})\ \underline{\text{sat}}\ (c \leq \langle 3,4\rangle \& d \leq e)$.

Note. If c is a subscripted channel name from an array d[M], then $R^{d[f]}_{e\hat{\ }d[f]}$ is taken to be

$$R^{d}_{\lambda i:M.\ \underline{\text{if}}\ i = f\ \underline{\text{then}}\ e\hat{\ }d[f]\ \underline{\text{else}}\ d[i]},$$

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 → P) is like (c!x → P), except that it is prepared for communication of <u>any</u> 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.

$$\frac{\Gamma \vdash R_{<>}\ ,\quad \forall v{\in}M.\, P^{x}_{v}\ \underline{\text{sat}}\ R^{c}_{v\wedge c}}{\Gamma \vdash (c?x{:}M \rightarrow P)\ \underline{\text{sat}}\ R.} \qquad \text{(input)}$$

Example. Let Γ = copier <u>sat</u> (wire ≤ input).
Then $\Gamma \vdash$ <> ≤ v^<>, copier <u>sat</u> (v^wire≤v^input) (proved before).
∴ $\Gamma \vdash$ <>≤<>, ∀v∈M. (wire!v → copier) <u>sat</u> (wire≤v^input) (output, ∀-int)
∴ $\Gamma \vdash$ (input?x:M → wire!x → copier) <u>sat</u> (wire≤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.

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

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 $(P_x||_yQ)$ 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∩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.

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

Example: ⊢ copier <u>sat</u> wire ≤ input,
recopier <u>sat</u> output ≤ wire (assume)
Therefore ⊢ $(\text{copier}_x||_y\ \text{recopier})$ <u>sat</u> output≤wire & wire≤input (parallelism)
and so ⊢ $(\text{copier}_x||_y\text{recopier})$ <u>sat</u> (output ≤ input) (consequence)
(where X ={input,wire) and Y ={output,wire}).

(9) Let R be an assertion which does not mention any channel of the list L. Suppose P <u>sat</u> 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.

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

Example. ⊢ $(\text{copier}_x||_y\text{recopier})$ <u>sat</u> output ≤ input

(already shown)

therefore ⊢ (<u>chan</u> wire; $\text{copier}_x||_y\text{recopier}$) <u>sat</u> output≤ input.

(10) Consider a process name p, defined recursively by the equation p<u>Δ</u>P. We allow such definitions to appear in the list of assumptions of an inference. Suppose we wish to prove that "p<u>sat</u>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<u>sat</u>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<u>sat</u>R.
If p is not free in Γ.

$$\frac{\Gamma \vdash R_{<>}\ ;\quad \Gamma,\ p\,\underline{\text{sat}}\,R \vdash P\,\underline{\text{sat}}\,R}{\Gamma,\ p\,\underline{\Delta}\,P \vdash p\,\underline{\text{sat}}\,R} \qquad \text{(recursion)}$$

Note: The inference rules for recursion depend on <u>two</u> subsidiary inferences, here separated by semicolon.

Example: Let P stand for
(input?x:NAT → wire!x → copier).
⊢<>≤<> (theorem)
copier <u>sat</u> wire≤input ⊢
P<u>sat</u>wire≤input (already proved)
therefore copier<u>Δ</u>P ⊢ copier <u>sat</u>wire≤input. (recursion).

This rule extends to process array definitions: if q is not free in Γ,

$$\frac{\Gamma \vdash (\forall x{\in}M.\, S_{<>})\ ;\ \Gamma, (\forall x{\in}M.\, q[x]\,\underline{\text{sat}}\,S) \vdash (\forall x{\in}M.\, Q\,\underline{\text{sat}}\,S)}{\Gamma,\ q[x{:}M]\,\underline{\Delta}\,Q \vdash (\forall x{\in}M.\ q[x]\,\underline{\text{sat}}\,S)}$$

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

$$\frac{\Gamma \vdash R_{<>}, (\forall x{\in}M.\, S_{<>})\ ;\ \Gamma, p\,\underline{\text{sat}}\,R, (\forall x{\in}M.\, q[x]\,\underline{\text{sat}}\,S) \vdash P\,\underline{\text{sat}}\,R, (\forall x{\in}M.\, Q\,\underline{\text{sat}}\,S)}{\Gamma, p\,\underline{\Delta}\,P,\ q[x{:}M]\,\underline{\Delta}\,Q \vdash p\,\underline{\text{sat}}\,R, (\forall x{\in}M.\, q[x]\,\underline{\text{sat}}\,S)}$$

Examples (2.2.)

(1) Let $\Delta1$ be the list of definitions.

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

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 $<x,NACK>$, e.g.

$$f(<x,NACK,\ y,\ ACK>) = y$$

thus $f(<>) = <>$, $f(<x>) = <x>$, $f(x \frown ACK \frown wire) = x \frown f(wire)$, and $f(x \frown NACK \frown wire) = f(wire)$.

We want to prove that $\Delta1 \vdash sender\ \underline{sat}\ f(wire) \le input$.

Proof: Let A1 be the list of predicates
$sender\ \underline{sat}\ f(wire) \le input$,
$\forall x \in M.\ q\boxed{x}\underline{sat} f(wire) \le x \frown input$.

We shall prove the stronger lemma that $\Delta1 \vdash A1$ by rule (recursion).

It is easy to check the first subsidiary inference that $\vdash f(<>) \le <>$, $\forall x \in M.\ f(<>) \le x \frown <>$. The main part of the proof is displayed in table 1.

(2) Let $\Delta2$ be the definition.

$$receiver\ \underline{\Delta}(wire?x{:}M \rightarrow (wire!ACK \rightarrow output!x \rightarrow receiver \mid wire!NACK \rightarrow receiver))$$

We wish to prove that $\Delta2 \vdash receiver\underline{sat}output \le f(wire)$. The proof is left as an exercise.

(3) Let $\Delta3$ be the definition
$protocol\ \underline{\Delta}\ (\underline{chan} wire; sender_x \| _y receiver)$.

We wish to prove that $\Delta1,\ \Delta2,\ \Delta3 \vdash protocol\ \underline{sat}\ output \le input$.

(1) $sender\underline{sat}\ f(wire) \le input$ (already proved from $\Delta1$)

(2) $receiver\underline{sat}output \le f(wire)$ (already proved from $\Delta2$)

(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≤)

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

(6) $protocol\ \underline{sat}\ output \le input$
($\Delta3$, recursion (5), $<> \le <>$)

Prove the second subsidiary inference:

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

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

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

(1)	$sender\ \underline{sat}\ f(wire) \le input$	(assumption)
(2)	$\forall x \in M. q[x]\underline{sat}f(wire) \le x \frown input$	(assumption)
(3)	$f(<>) \le <>$	(def f)
(4)	$(input?x{:}M \rightarrow q[x])\underline{sat}f(wire) \le input$	(input (2),(3))
(5)	$x \in M \Rightarrow q[x]\underline{sat}f(wire) \le x \frown input$	($\forall$-elim (2))
(6)	$x \in M$	(assumption)
(7)	$q[x]\underline{sat}f(wire) \le x \frown input$	($\Rightarrow$-elim (5),(6))
(8)	$f(wire) \le input \Rightarrow f(x \frown ACK \frown wire) \le x \frown input$	(def f)
(9)	$f(wire) \le x \frown input \Rightarrow f(x \frown NACK \frown wire) \le x \frown input$	(def f)
(10)	$sender\ \underline{sat}\ f(x \frown ACK \frown wire) \le x \frown input$	(consequence (1),(8))
(11)	$\forall v \in \{ACK\}.sender\ \underline{sat}\ f(x \frown v \frown wire) \le x \frown input$	($\forall$-int (10))
(12)	$q[x]\ \underline{sat}\ f(x \frown NACK \frown wire) \le x \frown input$	(consequence (7),(9))
(13)	$\forall v \in \{NACK\}.\ q[x]\ \underline{sat}\ f(x \frown v \frown wire) \le x \frown input$	($\forall$-int (12))
(14)	$f(<x>) \le <x>$	(def f)
(15)	$(wire?y{:}\{ACK\} \rightarrow sender)\ \underline{sat}\ f(x \frown wire) \le x \frown input$	(input (11),(14))
(16)	$(wire?y{:}\{NACK\} \rightarrow q[x])\ \underline{sat}\ f(x \frown wire) \le x \frown input$	(input (13),(14))
(17)	$(wire?y{:}\{ACK\} \rightarrow sender \mid wire?y{:}\{NACK\} \rightarrow q[x])\ \underline{sat}\ f(x \frown wire) \le x \frown input$	(alternative (15),(16))
(18)	$f(<>) \le <x>$	(def f)
(19)	$(wire!x \rightarrow (wire?y{:}\{ACK\} \rightarrow sender \mid wire?y{:}\{NACK\} \rightarrow q[x]))\ \underline{sat}\ f(wire) \le x \frown input$	(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

Validity of the inference system (3)

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 space. 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.

Prefix closures (3.1.)

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

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

$$\begin{aligned} &\langle\rangle \setminus C = \langle\rangle, \\ &(c.m \frown s) \setminus C = c.m \frown (s \setminus C) \quad \text{if } c \tilde{\in} C \\ &\qquad\qquad\qquad = s \setminus C \quad \text{if } c \in C \\ &st \setminus C = (s \setminus C)(t \setminus C), \\ &u \setminus C = st \Rightarrow \exists vw.\ u = vw \ \&\ v \setminus C = s \ \&\ w \setminus C = t \end{aligned}$$

If P is a prefix closure, then we define

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

Theorem. $P \setminus C$ and P/C are prefix closures, and they are distributive in P.
Proofs are omitted; they are similar to the previous proof.

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

$$P\,{}_X\!\cap_Y Q = (P/(Y-X)) \cap (Q/(X-Y)).$$

Let s be a trace of this set. It follows that $s \setminus \bar{X} \in P$ and $s \setminus \bar{Y} \in Q$. 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. ${}_X\!\cap_Y$ is a distributive operator.

Proof. Trivial.

Denotational semantics of process expressions (3.2.)

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,

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

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, so that $\rho[\![P]\!]$ is the prefix closure denoted by P when the free variables of P take the values ascribed by ρ. 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) $\rho[\![P \underset{X\ Y}{||} Q]\!] = \rho[\![P]\!] \underset{\rho[\![X]\!]\ \rho[\![Y]\!]}{\cap} \rho[\![Q]\!]$

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

Semantics of inference rules (3.3.)

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.

$$(\rho+ch(s))[\![x]\!] = ch(s)(x) \text{ if } x \text{ is a channel name}$$
$$= ch(s)(c[\rho[\![e]\!]]) \text{ if } x \text{ is a subscripted channel name } c[e]$$
$$= \rho[\![x]\!] \text{ if } x \text{ contains no channel names}$$

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 sat R" states that all traces of the process P satisfy the predicate R, i.e.

$$\rho[\![P\ \underline{sat}\ R]\!] = \forall s.s \in \rho[\![P]\!] \Rightarrow (\rho+ch(s))[\![R]\!].$$
and $\rho[\![\forall x \in M.\ P\ \underline{sat}\ R]\!] = \forall v.v \in \rho[\![M]\!] \Rightarrow \rho[v/x][\![P\ \underline{sat}\ R]\!]$.

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 $\underline{p\Delta P}$. We define $\rho[\![\underline{p\Delta P}]\!]$ 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 $\underline{p\Delta P}$. 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)

$$\rho[\![\underline{p\Delta P}]\!] = (\rho(p) = \bigcup_{i \geq 0} a_i)$$

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

---

* usually written ρ and pronounced "rho".

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

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

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

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

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

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.

<u>Proofs</u> (3.4.)

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 $R_{\langle\rangle}$ is formed from R by replacing all channel names by $\langle\rangle$

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

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

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

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

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

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

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

(1) Triviality. Suppose

$\forall\rho.\ \rho[\![\Gamma]\!]\Rightarrow\rho[\![T]\!]$. Then

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

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

$\rho[\![\Gamma]\!]\Rightarrow(\forall s.\ s\in\rho[\![P]\!]\Rightarrow(\rho+ch(s))[\![R]\!])\&(\forall s.(\rho+ch(s))[\![R]\!]$
$\Rightarrow(\rho+ch(s))[\![S]\!])$
$\Rightarrow\forall s.\ s\in\rho[\![P]\!]\Rightarrow(\rho+ch(s))[\![S]\!]$
$\Rightarrow \rho[\![P\,\underline{sat}\,S]\!].$

(3) Conjunction. Trivial.

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

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

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

$\rho[\![R_{\langle\rangle}]\!]$ and $\rho[\![P\,\underline{sat}\,R^c_{e{}^\wedge c}]\!]$. Thus

$s\in\rho[\![c!e\rightarrow P]\!]\Rightarrow(s=\langle\rangle \vee s=\rho[\![c]\!].\rho[\![e]\!]{}^\wedge t)$
(for some t in $\rho[\![P]\!]$).

In the first case,

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

In the second case,

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

So $\forall\rho.\ \rho[\![\Gamma]\!]\Rightarrow\forall s.\ s\in\rho[\![c!e\rightarrow P]\!]\Rightarrow(\rho+ch(s))[\![R]\!]$, i.e.
$\forall\rho.\ \rho[\![\Gamma]\!]\Rightarrow\rho[\![(c!e\rightarrow P)\underline{sat}\,R]\!]$.

(6) Input. Assume $\forall\rho.\ \rho\llbracket\Gamma\rrbracket \Rightarrow (\rho\llbracket R_{<>}\rrbracket$
$\&\ \rho\llbracket\forall v\in M.\ P^x_v\ \underline{sat}\ R^c_{v\wedge c}\rrbracket)$.

Given ρ such that $\rho\llbracket\Gamma\rrbracket$,
then $\rho\llbracket R_{<>}\rrbracket$
and $\rho\llbracket\forall v\in M.\ P^x_v\ \underline{sat}\ R^c_{v\wedge c}\rrbracket$, i.e.

$\forall u.\ u\in\rho\llbracket M\rrbracket\Rightarrow\rho[^u/_v]\llbracket P^x_v\ \underline{sat}\ R^c_{v\wedge c}\rrbracket$. Thus

$$s\in\rho\llbracket c?x{:}M\rightarrow P\rrbracket\Rightarrow s\in(\{<>\}\ \bigcup_{u\in\rho\llbracket M\rrbracket}(\rho\llbracket c\rrbracket.u\rightarrow\rho[^u/_x]\llbracket P\rrbracket))$$
$$\Rightarrow s=<>\ \vee\ s=\rho\llbracket c\rrbracket.u\wedge t$$

(for some $u\in\rho\llbracket M\rrbracket$ and $t\in\rho[^u/_x]\llbracket P\rrbracket$).

Let us only check the second case:

$$s=\rho\llbracket c\rrbracket.u\wedge t\Rightarrow(\rho+ch(s)\llbracket R\rrbracket=(\rho+ch(\rho\llbracket c\rrbracket.u\wedge t))\llbracket R\rrbracket$$
$$\Rightarrow(\rho+ch(s)\llbracket R\rrbracket=(\rho[^u/_v]+ch(\rho[^u/_v]\llbracket c\rrbracket.\rho[^u/_v]\llbracket v\rrbracket\wedge t))\llbracket R\rrbracket$$

(since v is not free in R and c)

$$\Rightarrow(\rho+ch(s)\llbracket R\rrbracket=(\rho[^u/_v]+ch(t))\llbracket R^c_{v\wedge c}\rrbracket$$

(lemma (3))

Furthermore since v is not free in P,

therefore $t\in\rho[^u/_x]\llbracket P\rrbracket$ is equivalent to $t\in\rho[^u/_v]\llbracket P^x_v\rrbracket$. Then $(\rho[^u/_v]+ch(t))\llbracket R^c_{v\wedge c}\rrbracket$ follows from the asumption

$\forall u.\ u\in\rho\llbracket M\rrbracket\Rightarrow\rho[^u/_v]\llbracket P^x_v\ \underline{sat}\ R^c_{v\wedge c}\rrbracket$. So $(\rho+ch(s))\llbracket R\rrbracket$.

Hence

$\forall s.\ s\in\rho\llbracket c?x{:}M\rightarrow P\rrbracket\Rightarrow(\rho+ch(s))\llbracket R\rrbracket$, provided $\rho\llbracket\Gamma\rrbracket$.

(7) Alternative. Trivial.

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

$$s\in\rho\llbracket P_X\|_Y Q\rrbracket\Rightarrow s\in(\rho\llbracket P\rrbracket\ \underset{\rho\llbracket X\rrbracket\ \rho\llbracket Y\rrbracket}{\cap}\ \rho\llbracket Q\rrbracket)$$
$$\Rightarrow s\backslash(\rho\llbracket Y\rrbracket-\rho\llbracket X\rrbracket)\in\rho\llbracket P\rrbracket$$
$$\&\ s\backslash(\rho\llbracket X\rrbracket-\rho\llbracket Y\rrbracket)\in\rho\llbracket Q\rrbracket$$
$$\Rightarrow(\rho+ch(s\backslash(\rho\llbracket Y\rrbracket-\rho\llbracket X\rrbracket)))\llbracket R\rrbracket\ \&\ (\rho+ch(s\backslash(\rho\llbracket X\rrbracket-\rho\llbracket Y\rrbracket)))\llbracket S\rrbracket$$

(by $\rho\llbracket P\ \underline{sat}\ R\rrbracket$ and $\rho\llbracket Q\ \underline{sat}\ S\rrbracket$)

$$\Rightarrow(\rho+ch(s))\llbracket R\rrbracket\ \&\ (\rho+ch(s))\llbracket S\rrbracket$$

(lemma (4))

$$\Rightarrow(\rho+ch(s))\llbracket R\&S\rrbracket.$$

So $\forall s.\ s\in\rho\llbracket P_X\|_Y Q\rrbracket\Rightarrow(\rho+ch(s))\llbracket R\&S\rrbracket$, provided $\rho\llbracket\Gamma\rrbracket$.

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

$$s\in\rho\llbracket(\underline{chan}\,L;P)\rrbracket\Rightarrow s\in(\rho\llbracket P\rrbracket\backslash\rho\llbracket L\rrbracket)$$
$$\Rightarrow s\in t\backslash\rho\llbracket L\rrbracket$$

(for some t in $\rho\llbracket P\rrbracket$)

$$\Rightarrow(\rho+ch(s))\llbracket R\rrbracket=(\rho+ch(t\backslash\rho\llbracket L\rrbracket))\llbracket R\rrbracket$$
$$\Rightarrow(\rho+ch(s))\llbracket R\rrbracket=(\rho+ch(t))\llbracket R\rrbracket\quad\text{(lemma (4))}$$
$$\Rightarrow(\rho+ch(s))\llbracket R\rrbracket$$

(by $\rho\llbracket P\ \underline{sat}\ R\rrbracket$).

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

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

Suppose $\forall\rho.\ \rho\llbracket\Gamma\rrbracket\Rightarrow\rho\llbracket R_{<>}\rrbracket$ and $\forall\rho'.\ (\rho'\llbracket\Gamma\rrbracket\&\rho'\llbracket p\ \underline{sat}\ R\rrbracket)\Rightarrow\rho'\llbracket P\ \underline{sat}\ R\rrbracket$

Given ρ such that $\rho\llbracket\Gamma\rrbracket$ and $\rho\llbracket p\underline{\Delta}P\rrbracket$, let us prove $\forall s.\ s\in\rho\llbracket p\rrbracket\Rightarrow(\rho+ch(s))\llbracket R\rrbracket$.

Since $\rho\llbracket p\underline{\Delta}P\rrbracket\Rightarrow\rho(p)=\bigcup_{i\geq 0}a_i$ and $\rho\llbracket p\rrbracket=\rho(p)$, therefore

$s\in\rho\llbracket p\rrbracket\Rightarrow s\in\bigcup_{i\geq 0}a_i$.

Consider first the base case.

$$s\in a_0\Rightarrow s\in\rho\llbracket STOP\rrbracket$$
$$\Rightarrow s=<>$$
$$\Rightarrow(\rho+ch(s))\llbracket R\rrbracket=\rho\llbracket R_{<>}\rrbracket\quad\text{(lemma (2))}$$
$$\Rightarrow(\rho+ch(s))\llbracket R\rrbracket\quad\text{(by first premise).}$$

Note now that $(\rho[a^i/p]+ch(s))\llbracket R\rrbracket=(\rho+ch(s))\llbracket R\rrbracket$. 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 $\rho[a^i/p]\llbracket\Gamma\rrbracket=\rho\llbracket\Gamma\rrbracket$, since p is not free in Γ.

Now assume for arbitrary i

$$\forall s.\ s\in a_i\Rightarrow(\rho+ch(s))\llbracket R\rrbracket$$

then $\forall s.\ s\in\rho[a^i/p]\llbracket p\rrbracket\Rightarrow(\rho+ch(s)\llbracket R\rrbracket$

$(s\in\rho[a^i/p]\llbracket p\rrbracket=s\in a_i)$,

i.e. $\rho[a^i/p]\llbracket p\ \underline{sat}\ R\rrbracket$.

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

$$\rho[a^i/p]\llbracket P\ \underline{sat}\ R\rrbracket,$$

i.e. $\forall s.\ s\in\rho[a^i/p]\llbracket P\rrbracket\Rightarrow(\rho[a^i/p]+ch(s))\llbracket R\rrbracket)$

thus $\forall s.\ s\in a_{i+1}\Rightarrow(\rho+ch(s))\llbracket R\rrbracket$

$(a_{i+1}=\rho[a^i/p]\llbracket P\rrbracket$ and $(\rho+ch(s))\llbracket R\rrbracket=(\rho[a^i/p]+ch(s))\llbracket R\rrbracket$

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

i.e. $\forall s.\ s\in\rho\llbracket p\rrbracket\Rightarrow(\rho+ch(s))\llbracket R\rrbracket$.

Conclusion (4)

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 $P \cup Q$ 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 timedependent.

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"
In "On the Construction of Programs" C.U.P. pp 229-254 (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).

PARTIAL CORRECTNESS OF COMMUNICATION PROTOCOLS

Zhou Chao Chen and C.A.R. Hoare

Programming Research Group, University of Oxford.
Institute of Computing Technology, Academia Sinica, Peking

The previous paper introduced a notation for describing the behaviour and proving invariant properties of processes communicating over an arbitrary network of named channels. In this paper we confine attention to chains of linearly connected processes, in which each process can communicate only with its neighbour to the left or to the right. These chains can be used in the design of multi-level communications protocols, and an example of such is given in the final section.

CR Categories: 4.22 5.24

Key words and phrases: partial correctness, parallel programming, communications protocols, communicating processes.

1. Communication protocols.

From the most abstract point of view, a single-directional communication protocol can be specified as a process which accepts messages at the transmitting end (the left), and accurately reproduces them at the receiving end (the right). Its behaviour can be described as that of a process P communicating with its environment through channels named "left" and "right". The specification of its correctness states that the sequence of values transmitted to the right shall always be an initial subsequence of the sequence of values input from the left, i.e.

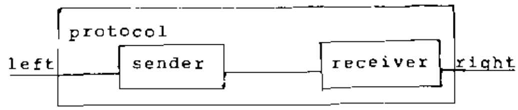
$$P \ \underline{\text{sat}} \ \text{right} \leq \text{left}.$$

A very simple process which satisfies this specification can be defined.

$$\text{copier} \triangleq (\text{left}?x{:}M \rightarrow \text{right}!x \rightarrow \text{copier})$$

where M is the set of message values that can be communicated.

In practice, of course, a communication protocol must be implemented as two processes, a sender and a receiver, connected by a transmission medium which physically separates them:



The sender copies messages from its left to the channel on its right, and the receiver copies messages from the channels on its left to the end recipient on its right.

In defining the sender and receiver (or any other processes connected in series with each other) it is very convenient to allow each process to use the name "left" to refer to the channel on its left, and the channel name "right" to refer to the channel name on its right. All processes defined in this paper will observe this convention. In order to connect such processes in series, we need to define a new composition operator, denoted by <>
Using this operator, we can give a formal definition of the picture:

$$\text{protocol} \triangleq (\text{sender} <> \text{receiver})$$

where sender = receiver = copier

The formal definition of this operation ($P<>Q$) must ensure that whenever the process P communicates on its right and the process Q communicates on its left, the effect is the same as if they were communicating on the <u>same</u> channel. Let us give this channel the temporary name "t". Now we define $P[t/\text{right}]$ as the process which behaves exactly like P, except that whenever P uses the name "right", $P[t/\text{right}]$ uses the name "t", more formally:

$$\underline{P}[\![P[t/\text{right}]]\!] = \{s|t/\text{right} \mid s \in \underline{P}[\![P]\!]\}$$

where $s[t/\text{right}]$ is formed from s by replacing every occurence of the channel name "right" by "t".

$Q[t/\text{left}]$ is defined similarly. The required communication between P and Q can now be achieved by composing them in parallel:

$$(P[t/\text{right}])_X \|_Y (Q[t/\text{left}]),$$

where $x = \{left, t\}$ and $y = \{t, right\}$. Then

the communications on the channel "t" must be concealed by declaring "t" as a local channel of the construction:

$$P<>Q =_{df} (\underline{chan}\ t; (P[t/right]_x \|_y Q[t/left]))$$

After such a complicated definition, it is comforting to check that if P and Q communicate only to the left and to the right, then $(P<>Q)$ has the same property. Thus $(P<>Q)$ can be successfully composed with another such process R, and the composition operator is associative.

$$(P<>Q)<>R = P<>(Q<>R)$$

Physically, this means it does not matter in what order the processes are connected. Syntactically, it means that brackets can be omitted without fear of ambiguity. The proof rule for this form of composition follows directly from its definition.

$$\frac{\Gamma \vdash P\ \underline{sat}\ R,\ Q\ \underline{sat}\ S}{\Gamma \vdash (P<>Q)\ \underline{sat}\ (R;S)}$$

where R and S are predicates of the channel names "left" and "right", and

$$(R;S) =_{df} \exists t \cdot R_t^{right}\ \&\ S_t^{left},$$

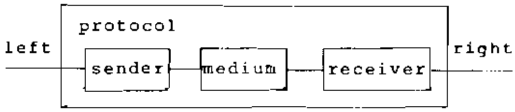
where t is a fresh variable, and R_t^{right} is formed from R by replacing all occurences of "right" by "t" and S_t^{left} is similarly formed from S.

A predicate R containing only two free variables "left" and "right" has an obvious correspondence with a relation

$$\{<left, right> | R\}$$

Under this correspondence, the operation $(R;S)$ is exactly the relational composition of R and S, and we can freely use its convenient properties, e.g., that it is associative and distributes through "or".

The design of the sender/receiver protocol given above was absurdly simple. For a practicable protocol, we need to take into account the unreliability of the transmission medium over which the messages are sent. The unreliable behaviour of the medium can also be modelled as a process, which communicates with the sending process (on its left) and the receiving process (on its right):



The formal definition of this series is just as expressive as the picture, and takes less space

$$protocol\ \underline{\Delta}\ (sender<>medium<>receiver)$$

The unreliability of a medium is best described by introducing an element of non-determinism into its behaviour. Let y range over elements of some set N, and let P_y be a process description for each value of y. Then $\prod_{y \in N} P_y$ describes a process that behaves like __any__ of the P_y, the choice between them being wholly arbitrary. In terms of sets of traces, this can be simply defined as the union of the traces of all the P_y as y ranges over N

$$\underline{P}[\![\prod_{y \in N} P_y]\!] = \bigcup_{v \in \underline{P}[\![N]\!]} \underline{P}[v/y][\![P_y]\!]$$

The corresponding proof rule is

$$\frac{\Gamma \vdash \forall y \in N.\ P_y\ \underline{sat}\ R}{\Gamma \vdash (\prod_{y \in N} P_y\ \underline{sat}\ R)}$$

As an example of an unreliable medium, consider one that may corrupt a message in passing. If x is a message value, let corruptions(x) be the set of possible message values which can result from such corruption. Of course, it is not excluded that the message may pass without corruption, i.e.

$$x \in corruptions\ (x).$$

(both the mathematician and engineer will regard this as such a special case that it is not worth mentioning separately). Now the behaviour of the medium can be defined

$$medium\ \underline{\Delta}\ (left?x{:}M \rightarrow \prod_{y \in corruptions(x)} (right!y \rightarrow medium))$$

Here, the selection of a particular corruption of x is nondeterministic. This medium satisfies the specification:

$$\#\ right \leq \#\ left$$

$\&\ \forall i < \#\ right.\ (right_i \in corruptions\ (left_i))$ where $\#\ c$ denotes the length of the sequence c.

The unreliability of such a medium can and should be mitigated by increased sophistication in the design of the sender and the receiver. In this case, it is the receiver that should try to reconstruct the correct value of a message from its corrupted version. Let "correction(y)" be a function that achieves this effect, i.e.

$$\forall y.\ y \in \text{corruptions}(x) \Rightarrow \text{correction}(y) = x$$

Then the "receiver" can be defined:

$$\text{receiver}_0 \ \underline{\Delta}\ (\text{left}?y{:}M \rightarrow \text{right}!\text{correction}(y) \rightarrow \text{receiver}_0).$$

This satisfies the specification

$$\#\ \text{right} \le \#\ \text{left}$$

$$\&\ \forall i < \#\ \text{right}.\ (\text{right}_i = \text{correction}\ (\text{left}_i))$$

We now wish to prove that the combination (medium <> receiver_0) is an error-free protocol, i.e. that

$$(\text{medium} <> \text{receiver}_0)\ \underline{\text{sat}}\ \text{right} \le \text{left}.$$

Using our proof rule for <>, we need to establish:

$$\#t \le \#\ \text{left}\ \&\ (\forall i < \#\ t.\ \ t_i \in \text{corruptions}\ (\text{left}_i))$$

$$\&\ \#\ \text{right} \le \#\ t\ \ \&\ \forall i < \#\ \text{right}.\ \ \text{right}_i = \text{correction}\ (t_i))$$

$$\Rightarrow \text{right} \le \text{left}$$

This follows immediately from the postulated properties of the corruptions and their corrections.

Unfortunately, it is not possible in general to find nontrivial corruption and correction relations for arbitrary messages; so it is necessary first to introduce some redundancy into the messages, and to strip off the redundancy afterwards. Let us introduce two functions "expand" and "contract" for this purpose, and stipulate that their composition is the identity function

$$(\text{expand};\ \text{contract}) = I.$$

Now we can define new senders and receivers

$$\text{sender}_1\ \underline{\Delta}\ (\text{left}?x{:}M \rightarrow \text{right}!\text{expand}(x) \rightarrow \text{sender}_1)$$

$$\text{receiver}_1\ \underline{\Delta}\ (\text{left}?x{:}M \rightarrow \text{right}!\text{contract}(x) \rightarrow \text{receiver}_1)$$

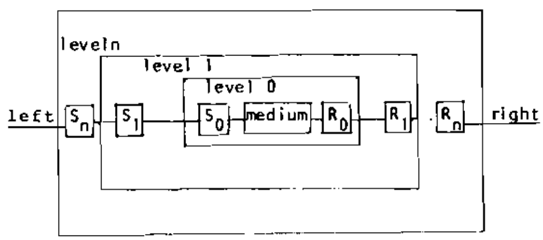
In order to achieve reliable transmission, we use the protocol_0 defined earlier as the medium over which the expanded messages are sent

$$\text{protocol}_1\ \underline{\Delta}\ \text{sender}_1 <> \text{protocol}_0 <> \text{receiver}_1.$$

That this is an error-free protocol can be readily proved by the proof rule of the composition operator

$$(\text{expand};I;\text{contract}) = (\text{expand};\text{contract}) = I$$

The technique of using a previously defined protocol as a transmission medium for a more elaborate protocol can be used to advantage in simplifying the design of elaborate protocols; indeed, it can be applied repeatedly at many <u>levels</u>; where the <u>lowest</u> level is the physical transmission medium, and the <u>highest</u> level is the protocol presented to the "end user". Each level has its own sender and receiver, and each of them treats the next lower level as the medium for transmission of its messages. Pictorially, the structure is like a set of nested boxes:



More formally, the levels can be defined

$$\text{level}_0\ \underline{\Delta}\ S_0 <> \text{medium} <> R_0$$

$$\text{level}_1\ \underline{\Delta}\ S_1 <> \text{level}_0 <> R_1$$

$$\vdots$$

$$\text{level}_n\ \underline{\Delta}\ S_n <> \text{level}_{n-1} <> R_n$$

$$\text{protocol}\ \underline{\Delta}\ \text{level}_n$$

But this conceptual structure for the protocol is quite different from its physical implementation, in which the senders at all levels are collected at one end of the transmission medium, and all the receivers at the other, as described in the definitions:

$$\text{sender}\ \underline{\Delta}\ (S_n <> \ldots <> S_1 <> S_0)$$

$$\text{receiver}\ \underline{\Delta}\ (R_0 <> R_1 <> \ldots <> R_n)$$

$$\text{protocol} \;\underline{\Delta}\; \text{sender} \ll\gg \text{medium} \ll\gg \text{receiver}$$

The associativity of the composition operator is vitally important to ensure that the physical and the logical groupings of the processes will exhibit identical behaviours.

The medium described above is a relatively well-behaved one. In practice, a transmission medium may lose messages, as well as corrupting them, or inserting spurious messages. For simplicity, we shall confine attention to a medium which simply loses messages

$$\text{lossy medium} \;\underline{\Delta}\; (\text{left}?x{:}M \rightarrow ((\text{right}!x \rightarrow \text{lossy medium}) \;\Pi\; \text{lossy medium}))$$

where the Π operator denotes non-deterministic choice between the two operands which it connects.

In order to counteract the unreliability of such a medium, it is essential for the receiver to be able to send back to the sender one of a range of signals acknowledging receipt of messages. Let A be the set of all such signals passing from right to left. It is reasonable to postulate that these can be distinguished from messages passing in the other direction, i.e.

$$A \cap M = \phi$$

The behaviour of a medium which transmits acknowledgements can be defined in the usual way as a process; and there is no guarantee that it will be immune to loss:

$$\text{copy back} = (\text{right}?a{:}A \rightarrow (\text{left}!a \rightarrow \text{copy back} \;\Pi\; \text{copy back}))$$

The overall behaviour of the transmission medium is a merging of the potential behaviours of the message medium and the acknowledgement medium.

$$\text{medium} = \text{lossy medium} \;|||\; \text{copyback}.$$

where $\underline{F}[\![P|||Q]\!] = \{s \mid \exists\, t,u.\, t \in \underline{P}[\![P]\!] \;\&\; u \in \underline{P}[\![Q]\!]$

$\&\; s$ is an interleaving of t and $u\}$.

The proof rule for interleaving operator is that, if $A \cap M = \phi$, then

$$\frac{\Gamma \vdash P \;\underline{\text{sat}}\; R1 \;\&\; \text{left}\restriction M = \text{right}\restriction M = \langle\rangle, \quad Q \;\underline{\text{sat}}\; R2 \;\&\; \text{left}\restriction A = \text{right}\restriction A = \langle\rangle}{\Gamma \vdash P|||Q \;\underline{\text{sat}}\; R1(\text{left}\restriction A, \text{right}\restriction A) \;\&\; R2(\text{left}\restriction M, \text{right}\restriction M),}$$

where $s\restriction C$ stands for the sequence obtained from s by cancelling the messages not in

C. An alternative definition of the medium without using $|||$ is:

$$\text{medium} \;\underline{\Delta}\; \text{left}?x{:}M \rightarrow \ell[x] \mid \text{right}?y{:}A \rightarrow r[y]$$

$$\ell[x{:}M] \;\underline{\Delta}\; \text{right}!x \rightarrow \text{medium} \mid \text{right}?y{:}A \rightarrow \ell r[x,y] \mid \text{medium}$$

$$r[y{:}A] \;\underline{\Delta}\; \text{left}!y \rightarrow \text{medium} \mid \text{left}?x{:}M \rightarrow \ell r[x,y] \mid \text{medium}$$

$$\ell r[x{:}M; y{:}A] \;\underline{\Delta}\; \text{right}!x \rightarrow r[y] \mid \text{left}!y \rightarrow \ell[x] \mid r[y] \mid \ell[x]$$

In order to counteract the losses on the medium defined above, we introduce a system of adding serial numbers to the messages and to the acknowledgement signals. Let n range over natural numbers, and let $s[n]$ be the behaviour of the sender before input of the nth message, and let $q[n,x]$ be its behaviour after input of the nth message with value x. In this state, it merely repeats output of the pair of values (n,x) until it receives the nth acknowledgement, all other acknowledgements being ignored. An acknowledgement is represented by a natural number, in this example $A = NN$.

$$\text{sender} \;\underline{\Delta}\; s[1]$$

$$s[n{:}NN] \;\underline{\Delta}\; (\text{left}?x{:}M \rightarrow q[n,x])$$

$$q[n,x] \;\underline{\Delta}\; (\text{right}!(n,x) \rightarrow q[n,x]) \mid (\text{right}?a{:}NN \rightarrow \text{if } a=n \;\underline{\text{then}}\; s[\text{succ}(n)] \;\underline{\text{else}}\; q[n,x])$$

Here, $\underline{\text{if}} \ldots \underline{\text{then}} \ldots \underline{\text{else}}$ has its usual meaning:

$$\underline{p}[\![\underline{\text{if}}\; B \;\underline{\text{then}}\; P \;\underline{\text{else}}\; Q]\!] = \underline{\text{if}}\; \underline{p}[\![B]\!] = \text{true} \;\underline{\text{then}}\; \underline{p}[\![P]\!] \;\underline{\text{else}}\; \underline{p}[\![Q]\!]$$

The corresponding proof rule is

$$\frac{\Gamma, B \vdash P \;\underline{\text{sat}}\; R; \quad \Gamma, \neg B \vdash Q \;\underline{\text{sat}}\; R}{\Gamma \vdash (\text{if } B \;\underline{\text{then}}\; P \;\underline{\text{else}}\; Q) \;\underline{\text{sat}}\; R}$$

Of course, in practice the retransmission of messages should not occur with too great rapidity; the process should spend a reasonable time waiting and listening for the acknowledgement. But such considerations of timing have been deliberately excluded from our mathematical theory, which is concerned only with those logical properties of the processes which are independent of timing.

The Receiver is similar to the sender. Its state after receipt of the nth message is $r[n]$. On receipt of the next message, the serial number is examined. If this is not equal to succ(n) the message is ignored. A message with a correct serial number is transmitted to the right, and its acknowledgement is sent back to the left. Acknowledgements for the previous message are repeated until a message with the next higher serial number is input

receiver $\underline{\Delta}$ r[0]

r[n:NN] $\underline{\Delta}$ (left?(a:NN, x:M) →

  <u>if</u> a=succ(n) <u>then</u> right!x→r[succ(n)]
                 <u>else</u> r[n]

  |left!n → r[n])

Here, the notation left?(a:NN,x:M) is used to input an ordered pair of values, the first of which is called "a" and the second "x".

Note that the spurious acknowledgements for the non-existent 0th message will be successfully ignored by the sender. More importantly, the set of acknowledgement signals can be reduced to merely two members A = {0,1}, with succ(0) = 1 and succ(1) = 0.

2. Weakest environment

In designing a chain of processes to meet some overall specification S, we may choose to design first the leftmost element of the chain to meet some specification Q.

Given Q and S, it is interesting to enquire what is the minimum specification R that must be met by the right part of the chain in order that their combination must meet the original specification, i.e.,

$$(Q;\ R) \Rightarrow S$$

The required specification is called the weakest <u>right</u> condition, and is defined:

$$S\ \underline{r}\ Q =_{df} \forall z. Q(z,\mathrm{left}) \Rightarrow S(z,\mathrm{right})$$

This definition has two important properties. Firstly $(S\ \underline{r}\ Q)$ itself (considered as a process) would be a suitable candidate to plug in on the right of Q in order that the combination should satisfy S.

Lemma 1

$$(Q;\ (S\ \underline{r}\ Q)) \Rightarrow S.$$

Proof

$$\begin{aligned} \mathrm{LHS} &= \exists t. Q(\mathrm{left},t)\ \&\ \forall z.Q(z,t) \Rightarrow S(z,\mathrm{right}) && \underline{\mathrm{def}};\ \text{and}\ \underline{r} \\ &\Rightarrow \exists t. Q(\mathrm{left},t)\ \&\ (Q(\mathrm{left},t) \Rightarrow S(\mathrm{left},\mathrm{right})) \\ &\Rightarrow S(\mathrm{left},\mathrm{right}) \end{aligned}$$

Secondly $(S\underline{r}Q)$ is both a necessary and sufficient condition which must be satisfied by any process if it is to serve its purpose in combination with Q:

Lemma 2

$$(Q;R) \Rightarrow S \quad \text{iff} \quad R \Rightarrow (S\underline{r}Q)$$

Proof

$$\begin{aligned} \mathrm{LHS} &= \forall \ell,r(\forall t.Q(\ell,t)\ \&\ R(t,r)) \Rightarrow S(\ell,r) && \underline{\mathrm{def}}; \\ &= \forall \ell,r,t.\ Q(\ell,t)\ \&\ R(t,r) \Rightarrow S(\ell,r) \\ &= \forall \ell,r,t. R(t,r) \Rightarrow (Q(\ell,t) \Rightarrow S(\ell,r)) \\ &= \forall t,r.\ R(t,r) \Rightarrow \forall \ell.Q(\ell,t) \Rightarrow S(\ell,r) \\ &= \mathrm{RHS} && \underline{\mathrm{def}}\ \underline{r} \end{aligned}$$

Theorem
$$\frac{\Gamma \vdash P1\ \underline{\mathrm{sat}}\ Q,\quad P2\ \underline{\mathrm{sat}}\ (S\underline{r}Q)}{\Gamma \vdash (P1 <> P2)\ \underline{\mathrm{sat}}\ S}$$

Of course, exactly similar reasoning applies if we wish to design first the rightmost member of a chain. We therefore define the weakest <u>left</u> condition:

$$R\ \underline{\ell}\ S =_{df} \forall z.\ R(\mathrm{right},z) \Rightarrow S(\mathrm{left},z)$$

Lemma 3 $\quad ((R\ \underline{\ell}\ S);\ R) \Rightarrow S.$

Lemma 4 $\quad ((Q;R) \Rightarrow S)\ \text{iff}\ (Q \Rightarrow (R\ \underline{\ell}\ S)).$

Theorem
$$\frac{\Gamma \vdash P1\ \underline{\mathrm{sat}}\ (R\underline{\ell}S)\ \&\ P2\ \underline{\mathrm{sat}}\ R}{\Gamma \vdash (P1 <> P2)\ \underline{\mathrm{sat}}\ S}$$

In designing a multi-level communication protocol, it is reasonable to design the higher levels first. Each level of the protocol has an overall specification S, and consists of a sender with specification Q and a receiver with specification R. It is interesting to enquire what is the weakest specification which must be met by the lower levels of the protocol in order that the design of the given level (Q,R) meet its specification S. We call this the weakest <u>inner</u> condition, and define

$$\mathrm{wic}(Q,S,R) =_{df} \forall z_1,z_2.Q(z_1,\mathrm{left})\ \&\ R(\mathrm{right},z_2) \Rightarrow S(z_1,z_2);$$

"wic" could also be defined in terms of $\underline{\ell}$ and $\underline{r}$, as shown in the following lemma:

Lemma 5
$$\mathrm{wic}(Q,S,R) = R\ \underline{\ell}(S\underline{r}Q) = (R\ \underline{\ell}\ S)\ \underline{r}\ Q.$$

The following lemmas give the desired properties of wic. They can be proved from the properties of $\underline{\ell}$ and $\underline{r}$.

Lemma 6 $\quad (Q;\ \mathrm{wic}(Q,S,R);\ R) \Rightarrow S).$

Lemma 7 $\quad (M \Rightarrow \mathrm{wic}(Q,S,R))\ \text{iff}\ ((Q;M;R) \Rightarrow S).$

Theorem

$$\frac{\Gamma \vdash P1 \ \underline{sat}\ Q,\ P3\ \underline{sat}\ R,\ P2\ \underline{sat}\ wic(Q,S,R)}{\Gamma \vdash (P1 <> P2 <> P3)\ \underline{sat}\ S}$$

In designing a protocol, it is logically impossible to guard against every conceivable error which can occur in the transmission medium: For example, nothing whatever can be done with a medium that delivers wholly random bits, or worse, one which, (like a more spiritual medium) delivers messages of a plausible but wholly fictitious transmitter. The best that can be done is to guard against most of the likely failure modes of the transmission medium. So it is useful to enquire of any given protocol what is the worst behaviour of the medium which it can tolerate, and still meet its overall specification. This is nothing other than the weakest inner condition of the whole protocol. The designer of the physical medium must ensure that the probability of violating this condition is negligibly small.

Now let us check if the previous protocol can tolerate the medium, which loses messages.

By the calculus given already we can prove that the processes "sender", "receiver" and "medium" satisfy the following specifications respectively.

Each time when "sender" receives a nth message x from its left side, it may transmit a variety of message sequences to its right side, which constitute the set $T_{x,n}$, where

$$T_{x,n} =_{df} (\{(n,x)\} \cup \{NN-\{n\}\})^{*} \hat{\ } \{n\},$$

where $A \hat{\ } B =_{df} \{s_1 \hat{\ } s_2 \mid s_1 \in A \ \&\ s_2 \in B\}$.

So the specification of "sender" can be defined as

$$T(left,right) =_{df} left \in M^{*} \ \&\ right \in T_{left},$$

where

$$T_{<>} = \{<>\} \text{ and } T_{x_1 \hat{\ } x_2 \hat{\ } \ldots \hat{\ } x_n} = \{s_1 \hat{\ } s_2 \hat{\ } \ldots \hat{\ } s_n \mid \forall i < n . s_i \in T_{x_i,i} \ \&\ \exists s . s \in T_{x_n,n} \ \&\ s_n \leq s\}$$

Similarly we describe the specification of "receiver" as follows.

$$R_{x,n} =_{df} (\{m\} \cup \{NN-\{n\} \times M\})^{*} \hat{\ } \{(n+1,x)\}$$

$$R(left,right) =_{df} left \in R_{right} \ \&\ right \in M^{*},$$

where $R_{<>} = \{s \mid \exists x,s' . s' \in R_{x,o} \ \&\ s \leq s'\}$

and

$$R_{x_1 \hat{\ } x_2 \hat{\ } \ldots \hat{\ } x_n} = \{s_1 \hat{\ } s_2 \hat{\ } \ldots \hat{\ } s_{n+1} \mid \forall i \leq n . s_i \in R_{x_i,i} \ \&\ \exists x,s . s \in R_{x,n+1} \ \&\ s_{n+1} \leq s\}$$

The overall specification for the protocol is left $\geq$ right.

Then $wic(T, left \geq right, R)$

$$= \forall z_1,z_2\ T(z_1,left) \ \&\ R(right,z_2) \Rightarrow z_1 \geq z_2$$

$$= \forall z_1,z_2 \in M^{*} . left \in T_{z_1} \ \&\ right \in R_{z_2} \Rightarrow z_1 \geq z_2.$$

The specification of "medium" - LOSS can roughly be described as : right↾NN×M can be obtained from left↾NN×M by cancelling some messages of NN×M, and left↾NN can be obtained from right↾NN by cancelling some messages of NN.

Now we can see $LOSS \Rightarrow wic(T, left \geq right, R)$. Because if $left \in T_{z_1}$ & $right \in R_{z_2}$ & LOSS (left,right), then "left" must be of form:

$$(x_1,1)^{+} \hat{\ } 1 \hat{\ } (x_2,1)^{+} \hat{\ } 2 \hat{\ } (x_3,3)^{+} \hat{\ } \ldots,$$

while "right" must be of form:

$$0^{*} \hat{\ } (x,1) \hat{\ } 1^{+} \hat{\ } (x_2,2) \hat{\ } 2^{+} \hat{\ } (x_3,3) \hat{\ } \ldots,$$

where u^{*} stands for any sequence of u and u^{+} for nonempty sequence of u.

The above definitions of the weakest conditions are given in terms of the overall specification and the specifications of first designed parts. Given process P, we know, the most precise specification of P, which can be defined in terms of channel predicate, is

$$P(s_1,s_2) =_{df} \exists s . s \in \mathcal{P}[\![P]\!] \ \&\ s \restriction left = s_1 \ \&\ s \restriction right = s_2.$$

So we define the weakest condition for given processes and overall specification:

$$S\ \underline{r}\ P =_{df} \forall z . P(z,left) \Rightarrow S(z,right),$$

$$P\ \underline{\ell}\ S =_{df} \forall z . P(right,z) \Rightarrow S(left,z),$$

and $wic(P1,S,P2) =_{df} \forall z_1,z_2\ P1(z_1,left) \ \&\ P2(right,z_2) \Rightarrow S(z_1,z_2),$

where P, P1 and P2 are processes and S is a channel predicate.

The following theorem shows that these definitions are reasonable.

Theorem.

(1) $\underline{P}[\![(P <> Q)\ \underline{sat}\ S]\!] = \underline{P}[\![Q\ \underline{sat}\ (S\ \underline{r}\ P)]\!]$

$$= \underline{P}[\![P\ \underline{sat}\ (Q\ \underline{\ell}\ S)]\!]$$

(2) $\underline{P}[\![(P1<>Q<>P2)\ \underline{sat}\ S]\!] =$

$\underline{P}[\![Q\ \underline{sat}\ wic(P1,S,P2)]\!]$

Based on these definitions we can develop a calculus for the proof of correctness of processes in terms of weakest conditions. Say, in this calculus we can get inference rules:

$\Gamma \vdash S\ \underline{r}(P1<>P2) \Leftrightarrow \Gamma \vdash (S\underline{r}P1)\underline{r}P2$

$\Gamma \vdash\ \ (P1<>P2)\underline{\dot{!}}S \Leftrightarrow \Gamma \vdash P1\ \underline{!}\ (P2\ \underline{!}\ S)$

$\Gamma \vdash wic(P1<>Q1,\ S,\ Q2<>P2) \Leftrightarrow$

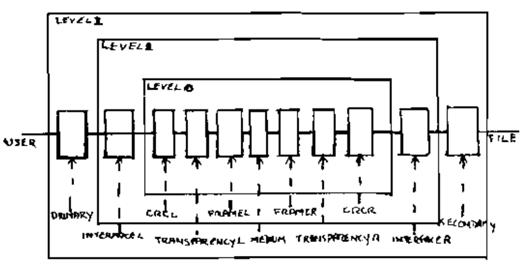
$\Leftrightarrow \Gamma \vdash wic(Q1, wic(P1,S,P2), Q2).$

The details of the calculus will not be presented in this paper.

3. An HDLC protocol.

In this section we present an HDLC protocol using the suggested approach, and simulate a medium with a burst of error. We then prove the partial correctness of the protocol in spite of these errors. Since the details of proofs are quite tedious, only a summary of them are given.

This is a point-to-point unbalanced system to collect files from a secondary station following the HDLC procedure. There are three levels in the protocol. The outermost level, level 2, is responsible for initiating the link, transmitting the data and disconnecting the link, according to the commands from higher levels, e.g. a user or file system. In level 2, the timeout retransmission and frame numbering are used to control error. This level works on messages, while the level 1 transforms messages into bit streams or vice versa like interface. The lowest expands and contracts bit streams for cyclic redundancy checks, transparency and framing. So the lowest level may be divided into three sublevels. The whole protocol can be pictured as the following diagram:



3.1. Level 2.

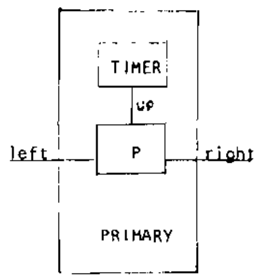
When PRIMARY receives an order "collect" from user, a file collection starts;

(1) PRIMARY initiates the link: sending SARM (Set Asychronous Response Mode) to SECONDARY, setting timer for retransmission of SARM, then waiting for the response UA (Unnumbered Acknowledgement) or DM (Disconnected Mode). DM means that SECONDARY has no data to be transmitted, so PRIMARY informs its user with "no", and this short transaction ends. UA means that SECONDARY wants to transmit a file, so PRIMARY informs user with "yes", and waits for data from SECONDARY.

(2) SECONDARY transmits data: a serial number Ns (modulo 8) is attached by SECONDARY to each item of data got from the file system; this data is then sent to PRIMARY with time-out transmission until RR (Receive Ready) is answered back. PRIMARY acknowledges receipt of data from SECONDARY with RR, and checks Ns to avoid duplicated data. If the serial number is in order, the data will be passed to user. If not, the data will be cancelled.

(3) Primary disconnects the link: At end of data collection "eof" is received by SECONDARY from file system. SECONDARY sends RD (Request Disconnect) to PRIMARY, and resends it until it receives DISC (Disconnect) back. When RD reaches PRIMARY it informs user with "eof", and answers SECONDARY with DISC.

The program PRIMARY can be written as follows:



P $\underline{\Delta}$ (left?collect $\to$ INITIAT)
  |(right?x:M $\to$ right!DISC $\to$ P),

where a?e stands for a?x:{e} and M for the set of all messages passing from SECONDARY to PRIMARY,

INITIAT $\underline{\Delta}$ right!SARM $\to$ up!set $\to$ WAIT

WAIT $\underline{\Delta}$ right?UA $\to$ up!reset $\to$ left!yes $\to$ RECEIVER
  |right?DM $\to$ up!reset $\to$ left!no $\to$ P
  |right?x:M-{UA,DM} $\to$ WAIT
  |up?timeout $\to$ INITIAT

RECEIVER $\underline{\Delta}$ R[0]

R[n:NN] $\underline{\Delta}$ right?(a:NN, x:DATA) $\to$ right!RR $\to$
  $\underline{\text{if}}$ a=n $\underline{\text{then}}$ left!x $\to$ R[n+1(mod8)]
    $\underline{\text{else}}$ R[n]

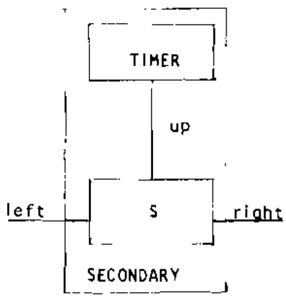
  |right?RD $\to$ left!eof $\to$ right!DISC $\to$ P
  |right?x: M-{(NN x DATA), RD} $\to$ R[n];

TIMER $\underline{\Delta}$ up? set $\to$ (up?reset $\to$ TIMER| up!timeout $\to$ TIMER)

PRIMARY $\underline{\Delta}$ ($\underline{\text{chan}}$ up; TIMER $_X||_Y$ P),

where X={up} and Y= {up,left,right}.

SECONDARY can be presented as follows:



$S \underline{\Delta}$ left?y:A → $\underline{\text{if}}$ y=SARM

$\underline{\text{then}}$ right!collect →

(right?yes → left!UA → SENDER

|right?no → left!DM → S)

$\underline{\text{else}}$ S,

where A stands for the set of all messages from PRIMARY to SECONDARY;

SENDER $\underline{\Delta}$ S[0];

S[n:NN] $\underline{\Delta}$ (right?x: DATA → Q[n,x])

|(right?eof → left!RD → up!set → WDISC)

Q[n:NN,x:DATA] $\underline{\Delta}$ left! (n,x) → up!set → WRR[n,x];

WRR[n:NN,x:DATA] $\underline{\Delta}$ left?RR → up!reset → S[n+1 (mod 8)]

|left?SARM → up!reset → left!UA→Q[n,x]

|left?y:A−{RR,SARM}→WRR[n,x]

|up?timeout → Q[n,x];

WDISC $\underline{\Delta}$ left?DISC → up!reset → S

|left?SARM → up!reset → S

|left?y: A−{DISC,SARM} → WDISC

|up?timeout → left!RD → up!set → WDISC;

TIMER $\underline{\Delta}$ up?set → (up?reset → TIMER| up!timeout→TIMER);

SECONDARY $\underline{\Delta}$ ($\underline{\text{chan}}$ up;TIMER$\|_{XY}$S),

where X ={up} and Y ={up,left,right}.

This protocol cannot guarantee that all the messages from the user can reach the system over a medium which may lose messages. This is because the loss of DM may cause the retransmission of SARM, and SECONDARY cannot recognise if this SARM is a new initializing signal or a retransmitted one. However, the data messages from file system to user are our main concern here, and this protocol can guarantee the correct data transmission over certain unreliable mediums as well.

So the overall specification of the protocol can be given as left$\upharpoonright$DATA≤right$\upharpoonright$DATA.

The specifications of PRIMARY and SECONDARY can be formulated in the following way:

Let the predicate S1 specify the sequences along the channel "right" of PRIMARY and let function f pick up the ordered data messages from the sequences. Then the specification of PRIMARY can be described as

$$\text{PRIM}(left,\text{right}) =_{df} S1(right) \& \text{left}\upharpoonright\text{DATA} \le f(\text{right}).$$

Let the predicate S2 specify the sequences along the channel "left" of SECONDARY. Then the specification of SECONDARY is

$$\text{SECD}(\text{left},\text{right}) =_{df} S2(\text{left}) \& f(\text{left}) \le right\upharpoonright\text{DATA}.$$

Thus wic(PRIM, left$\upharpoonright$DATA≤right$\upharpoonright$DATA, SECD)

$$= \forall z_1,z_2 \ . \ S1(\text{left}) \& z_1\upharpoonright\text{DATA} \le f(\text{left}) \& S2(\text{right}) \& f(\text{right}) \le z_2\upharpoonright\text{DATA} \Rightarrow z_1\upharpoonright\text{DATA} \le z_2\upharpoonright\text{DATA}$$

$$= \forall z_1,z_2 \ . \ S1(\text{left}) \& S2(\text{right}) \Rightarrow f(\text{left}) \le f(\text{right}).$$

This level is intended to work above the lower levels which may detect the errors caused by an unreliable medium. This intention can be checked as follows:

Let us define predicate LOSS(left,right) similarly to Section 2. i.e. cancelling some messages of A from "left$\upharpoonright$A" and some of M from "right$\upharpoonright$M" can form s_1 and s_2 such that s_1=right$\upharpoonright$A&left$\upharpoonright$M=s_2

Then we can prove

$$\text{LOSS} \Rightarrow \text{wic}(\text{PRIM},\text{left}\upharpoonright\text{DATA} \le \text{right}\upharpoonright\text{DATA},\text{SECD}).$$

i.e.

$$\text{LOSS}(\text{left},\text{right}) \& S1(\text{left}) \& S2(\text{right}) \Rightarrow f(\text{left}) \le f(\text{right})$$

3.2. Level 1.

This level realizes the transformation between a message and its binary code according to HDLC syntax. When receiving a message from level 2, level 1 transforms it into a bit stream with separators "start" and "end", then sends it to level 0. Conversely, when receiving a <start, bit-stream, end> from level 0, level 1 transforms it into the corresponding message, and passes it to level 2. If the received bit stream ends with the separator "error" (i.e. there is some error in this stream which has been detected by level 0) or no meaningful message corresponds to this bit stream, then this bit stream will be cancelled by this level.

For distinguishing between signals in different directions we use "start", "0", "1", "end", and "error" for signals from left to right, and "start'", "0'", "1'", "end'"and "error'"for signals from right to left.

The CSP processes INTERFACEL and INTERFACER of this level are not presented here, since they just do some routine coding and decoding.

Let decod be the function transforming the meaningful bit streams into messages according to HDLC syntax, and error streams, meaningless streams or incomplete streams into the empty sequence.

Then the specifications of INTERFACEL and INTERFACER are:

$$\mathrm{INTL}(\mathrm{left},\mathrm{right}) =_{df} \mathrm{left}\upharpoonright A \geq \mathrm{decod}(\mathrm{right}\upharpoonright A)$$
$$\quad \& \ \mathrm{left}\upharpoonright M \leq \mathrm{decod}(\mathrm{right}\upharpoonright M),$$

and
$$\mathrm{INTR}(\mathrm{left},\mathrm{right}) =_{df} \mathrm{decod}(\mathrm{left}\upharpoonright A) \geq \mathrm{right}\upharpoonright A$$
$$\quad \& \ \mathrm{decod}(\mathrm{left}\upharpoonright M) \leq \mathrm{right}\upharpoonright M.$$

In 3.1. we have shown that given PRIMARY and SECONDARY as the outer level, and $\mathrm{left}\upharpoonright \mathrm{DATA} \leq \mathrm{right}\upharpoonright \mathrm{DATA}$ as the overall specification, if the inner level satisfies the specification LOSS, then the whole protocol can satisfy the overall specification.

Now let us take LOSS as the overall specification of level 1 and INTERFACEL and INTERFACER as the outer level, and then look for an appropriate specification for the inner level (level 0).

Since wic(INTL,LOSS,INTR)

$$\begin{aligned}
&= \forall z_1, z_2.\,(z_1\upharpoonright A \geq \mathrm{decod}(\mathrm{left}\upharpoonright A) \\
&\qquad \& \ z_1\upharpoonright M \leq \mathrm{decod}(\mathrm{left}\upharpoonright M) \\
&\qquad \& \mathrm{decod}(\mathrm{right}\upharpoonright A) \geq z_2\upharpoonright A \\
&\qquad \& \mathrm{decod}(\mathrm{right}\upharpoonright M) \leq z_2\upharpoonright M) \\
&\qquad \Rightarrow \mathrm{LOSS}(z_1, z_2) \\
&= \mathrm{LOSS}(\mathrm{decod}(\mathrm{left}\upharpoonright A), \mathrm{decod}(\mathrm{right}\upharpoonright A)) \\
&\qquad \& \mathrm{LOSS}(\mathrm{decod}(\mathrm{left}\upharpoonright M), \mathrm{decod}(\mathrm{right}\upharpoonright M)) \\
&= \mathrm{LOSS}(\mathrm{decod}(\mathrm{left}), \mathrm{decod}(\mathrm{right})).
\end{aligned}$$

Let ERROR be a predicate to describe that there are some detected transmission errors. ERROR (left,right) holds if and only if there are sequences s_1 and s_2 obtained from "left$\upharpoonright$A " and "right$\upharpoonright$M" respectively by changing some bit streams to error streams, i.e. changing the end separator to "error", and stream body as well, such that $s_1 \geq \mathrm{right}\upharpoonright A$ and $\mathrm{left}\upharpoonright M \leq s_2$.

Then we can prove

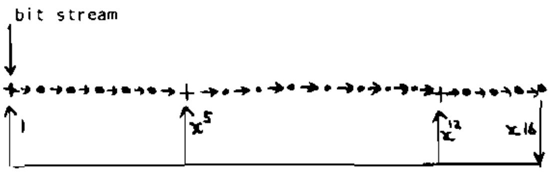
$$\mathrm{ERROR}(\mathrm{left},\mathrm{right}) \Rightarrow \mathrm{LOSS}(\mathrm{decod}(\mathrm{left}), \mathrm{decod}(\mathrm{right}))$$
$$\Rightarrow \mathrm{wic}(\mathrm{INTL},\mathrm{LOSS},\mathrm{INTR}).$$

Thus we will take ERROR as the specification of level 0; i.e. if level 0 can detect transmission errors, then the whole protocol works.

3.3. Level 0.

3.3.1. CRC sublevel.

Both CRC generation and check can be realized by shift register. The HDLC generating polynomial $p(x)$ is $x^{16}+x^{12}+x^5+1$, and the shift register for $p(x)$ is:



This shift register can be simulated in CSP as follows.

Let $p=0^3 10^6 10^4 1$, where a^n stand for n consecutive a's. Let tl(s) be the sequence obtained from s by cancelling its first bit - hd(s).

```
CRCGEN Δ left?start→right!start→SHIFT[0^16];
SHIFT[x:{0,1}*}]Δ left?y:{0,1}→right!y
                 → if hd(x)=1 then SHIFT[tl(x)^y⊕p]
                              else SHIFT[tl(x)^y]
              |left?end→CRC[x];
CRC[x:{0,1}*] Δ if x=<> then right!end→CRCGEN
                        else right!hd(x)→CRC[tl(x)].

CRCCHECK Δ right?start'→left!start'→SHIFTER[<>,<>];
SHIFTER[x:{0',1'}*,y:{0',1'}*]Δ right?z:{o',1'}
      →if length(x)≤15 then SHIFTER[x^z,y^z]
                       else left!hd(x)
      →if hd(y)=1' then SHIFTER[tl(x)^z,tl(y)^z⊕p]
                   → else SHIFTER[tl(x)^z, tl(y)^z]
     | right?end'→CHECK[y];
CHECK[y:{0',1'}*]Δ if y=0'^16 then left!end'→CRCCHECK
                               else left!error'→CRCCHECK;
CRCL Δ CRCGEN|||CRCCHECK
```

CRCR is similar to CRCL, but exchanging "left" and "right", and {start,0,1,end,error} and {start',0',1',end',error'}.

Let crc be the function on bitstreams defined as follows: if a bit stream with checksum is divisible by $p(x)$, then its corresponding value is the stream itself; if not divisible, then the value is the stream ended by separator "error" (or "error'"); if the stream is incomplete, then its value is the empty sequence.

Let crc' be the function defined in the same way as crc, except that the value of incomplete stream is the incomplete stream itself;

Let DIVISIBLE be a predicate. DIVISIBLE (s) iff all the complete streams in s are divisible by $p(x)$.

Then the specification of CRCL and CRCR are:

$$RL =_{df} left\upharpoonright A \geq crc(right\upharpoonright A) \ \& \ DIVISIBLE(right\upharpoonright A)$$
$$\& \ left\upharpoonright M \leq crc'(right\upharpoonright M)$$

and

$$RR =_{df} crc'(left\upharpoonright A) \geq right\upharpoonright A$$
$$\& \ crc(left\upharpoonright M) \leq right\upharpoonright M \ \& \ DIVISIBLE(left\upharpoonright M).$$

Now we define a predicate to describe a burst of errors of length less than 17 in a frame; then we can prove that it implies the weakest inner condition wic(RL, ERROR, RR).

BURST(left,right) holds iff by adding (modulo 2) bit streams of length less than 17 to the frames (complete or incomplete) of "left↾A" and "right↾M", we can obtain s_1 and s_2 such that $s_1 \geq right\upharpoonright A$ and $left\upharpoonright M \leq s_2$.

Since the burst errors less than 17 can be detected by CRC checksum of $p(x)$, we can prove

$$BURST \Rightarrow wic(RL, ERROR, RR)$$

3.3.2. Transparency sublevel.

This sublevel is responsible for inserting a redundant zero after five consecutive ones before transmitting frames, and removing the redundant zeros after receiving frames, for the sake of distinguishing the frame body from the frame flag (01^60).

```
INSERT Δ left?start→right!start→COUNT[0];
COUNT[x:NN] Δ if x=5 then right!0→COUNT[0]
                     else(left?0→right!0→COUNT[0]
                          |left?1→right!1→COUNT[x+1]
                          |left?end→right!end→INSERT)

REMOVEΔright?start'→left!start'→COUNT1[0];
COUNT1[x:NN]Δ right?1'→left!1'→COUNT1[x+1]
           |right?0'→if x≥5 then COUNT1[0]
                            else left!0'→COUNT1[0]
           |right?end'→left!end'→REMOVE;

TRANSPARENCY Δ INSERT|||REMOVE.
```

Similarly we can present TRANSPARENCYR.

Let redund be the function which cancels redundant zeros from bit streams. Then the specifications of this level can be given as

$$TRANPL(left,right) =_{df} left\upharpoonright A > redund(right\upharpoonright A)$$
$$\& \ left\upharpoonright M \leq redund(right\upharpoonright M)$$

and

$$TRANPR(left,right) =_{df} redund(left\upharpoonright A) \geq right\upharpoonright A$$
$$\& \ redund(left\upharpoonright M) \leq right\upharpoonright M$$

3.3.3 Frame sublevel.

This level is to transform the separators "start" and "end" into the HDLC frame flag (01^60) and vice versa.

```
FRAME Δ left?start→right!0→(right!1)^6→right!0→PASS;
PASS Δ left?x:{0,1}→right!x→PASS
      |left?end→right!0→(right!1)^6→right!0→FRAME;

DEFRAME Δ right?x:{0',1'}→if x=0'
                          then FLAG[0]
                          else OEFRAME;

FLAG[x:NN]Δ right?1'→FLAG[x+1]
          |right?0'→if x=6 then left!start'→BUF8[<>]
                           else DEFRAME ;

BUF8[x:{D',1'}*]Δ if x=0'1'^6 0'
                  then left!end'→DEFRAME
                  else (right?y:{0'1'}
                       → iflength(x)≤7 thenBUF8[x^y]
                  elseleft!hd(x)→BUF8[tl(x)^y]

FRAMEL Δ FRAME|||DEFRAME
```

FRAMER can be given simlarly.

Let fram be the function on bit streams which transforms the odd slag $o1^60$ into the separator "start" and the even one into "end", and cancels the unframed bit streams.

Then the specifications for this sublevel will be:

$$FL(left,right) =_{df} left\upharpoonright A \geq fram(right\upharpoonright A)$$
$$\& \ left\upharpoonright M \leq fram(right\upharpoonright M)$$

and

$$FR(left,right) =_{df} fram(left\upharpoonright A) \geq right\upharpoonright A$$
$$\& \ fram(left\upharpoonright M) \leq right\upharpoonright M$$

3.4. Medium.

Now we are simulating a medium of possible burst errors, the length of which is less than 17. At first let us check if the protocol can tolerate it.

Unfortunately, it is not true in the case that burst errors produce or destroy frame flags.

Suppose we have data $10^310^610^410^8$. Its CRC checksum is 0^{16}. So the framed bit stream for this data is

$$\underbrace{01^60}_{\text{flag}}\quad \underbrace{10^310^610^410^8}_{\text{data}}\quad \underbrace{0^{16}}_{\text{CRC}}\quad \underbrace{01^60}_{\text{flag}}$$

Thus if a burst of error of length 16 happens on the last 16 bits, and changes the bit stream to

$$\underbrace{01^60}_{\text{flag}}\quad \underbrace{10^310^610^41}_{\substack{\text{wrong}\\ \text{data}}}\quad \underbrace{0^80^8}_{\text{CRC}}\quad \underbrace{01^6\overbrace{01^8}^{\text{error}}}_{\text{flag}}$$

then a wrong, but undetectable data, $10^310^610^41$, reaches the destination.

So we simulate a medium, which may cause burst errors, but never produce or destroy frame flags in the following way. We plant a new level between CRC level and TRANPARENCY level; it consists of two processes: one may cause a burst of error for transmission from left to right, and the other one for right to left.

$$\begin{aligned}\mathrm{WIRE}\,\underline{\Delta}\,&\mathrm{left?start}\rightarrow\mathrm{right!start}\rightarrow\mathrm{WIRE}\\ &|\,\mathrm{left?y}:\{0.1\}\rightarrow(\mathrm{right!y}\rightarrow\mathrm{WIRE}\,|\,\mathrm{right!y}\oplus 1\rightarrow \mathrm{ERROR}[1])\\ &|\,\mathrm{left?end}\rightarrow\mathrm{right!end}\rightarrow\mathrm{WIRE}\ ;\end{aligned}$$

$$\begin{aligned}\mathrm{ERROR}[x:NN]\,\underline{\Delta}\ \mathrm{left?y}:\{0,1\}\rightarrow\ &\underline{if}\ x\le 15\\ &\underline{then}\ (\mathrm{right!0}\rightarrow\mathrm{ERROR}[x+1]\\ &\qquad|\,\mathrm{right!1}\rightarrow\mathrm{ERROR}[x+1])\\ &\underline{else}\ \mathrm{right!y}\rightarrow\mathrm{ERROR}[x]\\ |\,\mathrm{left?end}\rightarrow\mathrm{right!end}\rightarrow\mathrm{WIRE};&\end{aligned}$$

$\mathrm{PASS}\ \underline{\Delta}\ \mathrm{right?x}:M\rightarrow\mathrm{left!x}\rightarrow\mathrm{PASS};$

$\mathrm{MEDIUML}\ \underline{\Delta}\ \mathrm{WIRE}\,|||\,\mathrm{PASS}.$

MEDIUMR is similar to MEDIUML

The specification of them are:

$$\begin{aligned}\mathrm{ML(left,right)} =_{df}\ &\mathrm{BURST}(\mathrm{left}\upharpoonright A,\mathrm{right}\upharpoonright A)\\ \&\ &\mathrm{left}\upharpoonright M\le\mathrm{right}\upharpoonright M.\end{aligned}$$

and

$$\begin{aligned}\mathrm{MR(left,right)} =_{df}\ &\mathrm{left}\upharpoonright A\ge\mathrm{right}\upharpoonright A\\ \&\ &\mathrm{BURST}(\mathrm{left}\upharpoonright M,\mathrm{right}\upharpoonright M).\end{aligned}$$

Let
$$\begin{aligned}\mathrm{BUFF(left,right)} =_{df}\ &\mathrm{left}\upharpoonright A\ge\mathrm{right}\upharpoonright A\\ \&\ &\mathrm{left}\upharpoonright M\le\mathrm{right}\upharpoonright M.\end{aligned}$$

Then we can see

$\mathrm{BUFF}\Rightarrow\mathrm{wic(ML,BURST,MR)}.$

3.5. Partial Correctness of the Protocol.

Let us define the whole protocol as follows:

$$\begin{aligned}\mathrm{PROTOCOL}\,\underline{\Delta}\,&\mathrm{PRIMARY}\diamond\mathrm{INTERFACEL}\diamond\mathrm{CRCL}\diamond\mathrm{MEDIUML}\\ &\diamond\mathrm{TRANSPARENCYL}\diamond\mathrm{FRAMEL}\diamond\mathrm{FRAMER}\diamond\mathrm{TRANSPARENCYR}\\ &\diamond\mathrm{MEDIUMR}\diamond\mathrm{CRCR}\diamond\mathrm{INTERFACER}\diamond\mathrm{SECONDARY}.\end{aligned}$$

Since TRANSPARENCYL$\diamond$FRAMEL$\diamond$FRAMER$\diamond$TRANSPARENCYR <u>sat</u> BUFF can be proved from the specifications of the elements by the proof rule of composition, we have roughly shown that

$\mathrm{PROTOCOL}\ \underline{\mathrm{sat}}\ \mathrm{left}\upharpoonright\mathrm{DATA}\le\mathrm{right}\upharpoonright\mathrm{DATA}$

can be established by the theorem in Section 2.

Acknowledgement.

Thanks are due to Steve Brookes and Alastair Tocher for assistance in preparation of this paper.

PROGRAMMING RESEARCH GROUP TECHNICAL MONOGRAPHS

JUNE 1981

This is a series of technical monographs on topics in the field of computation. Copies may be obtained from the Programming Research Group, (Technical Monographs), 45 Banbury Road, Oxford, OX2 6PE, England.

PRG-1 *(out of print)*

PRG-2 Dana Scott
Outline of a Mathematical Theory of Computation

PRG-3 Dana Scott
The Lattice of Flow Diagrams

PRG-4 *(cancelled)*

PRG-5 Dana Scott
Data Types as Lattices

PRG-6 Dana Scott and Christopher Strachey
Toward a Mathematical Semantics for Computer Languages

PRG-7 Dana Scott
Continuous Lattices

PRG-8 Joseph Stoy and Christopher Strachey
OS6 - an Experimental Operating System for a Small Computer

PRG-9 Christopher Strachey and Joseph Stoy
The Text of OSPub

PRG-10 Christopher Strachey
The Varieties of Programming Language

PRG-11 Christopher Strachey and Christopher P. Wadsworth
Continuations: A Mathematical Semantics for Handling Full Jumps

PRG-12 Peter Mosses
The Mathematical Semantics of Algol 60

PRG-13 Robert Milne
The Formal Semantics of Computer Languages and their Implementations

PRG-14 Shan S. Kuo, Michael H. Linck and Sohrab Saadat
A Guide to Communicating Sequential Processes

PRG-15 Joseph Stoy
The Congruence of Two Programming Language Definitions

PRG-16 C. A. R. Hoare, S. D. Brookes and A. W. Roscoe
A Theory of Communicating Sequential Processes

PRG-17 Andrew P. Black
Report on the Programming Notation 3R

PRG-18 Elizabeth Fielding
The Specification of Abstract Mappings and their Implementation as B^+-trees

PRG-19 Dana Scott
Lectures on a Mathematical Theory of Computation

PRG-20 Zhou Chao Chen and C. A. R. Hoare
Partial Correctness of Communicating Processes and Protocols

PRG-21 Bernard Sufrin
Formal Specification of a Display Editor

PRG-22 C. A. R. Hoare
A Model for Communicating Sequential Processes

PRG-23 C. A. R. Hoare
A Calculus of Total Correctness for Communicating Processes

PRG-24 Bernard Sufrin
Reading Formal Specifications