SPECIFICATION AND CONSTRUCTION OF MACHINES

J.R. Abrial

Rapporteur: **Mr. B.C. Hampshere**

Abstract:

Specification and construction of sequential programs, although not entirely mastered, have now entered a more mature phase characterised by the recognition of a small set of well defined program constructs (sequencing, alternation, loop, and so on) together with the corresponding proof rules, and also by the definition of well defined and powerful specification and construction methods (pre-post-conditions, abstraction/implementation, combination of specifications, abstract data types, algebraic specifications, and so on).

The situation is certainly quite different for parallel processes. Although significant results have been obtained in recent years, there does not seem to exist a wide agreement about what the essential features of parallelism should be. (As an indication of this fact it is interesting to compare the various ways parallelism was handled in the four DOD coloured languages) Consequently one could hardly speak of well defined specification and construction methods, although some very spectacular results have been obtained here and there.

This talk is concerned with such questions as - Does it make sense to think of specifying processes which are supposed to run for ever? If yes, to specify what? Does there exist a way of "connecting" specifications in the same way as we connect processes, etc.?

The speaker will try to answer these kinds of questions and to study and illustrate a few examples.

SPECIFICATION AND CONSTRUCTION OF MACHINES

1. - WHAT IS A PROGRAM ?

In order to narrow the spectrum of possible answers to such a broad question, it might be advisable to choose a specific point of view. For instance, an interesting point of view might be that of the mathematician. In other words, by the question

What is a Program, from a Mathematical Point of View ?

we are not concerned by such activities as, say, the translation of programs from one Programming Language to the other or by the interpretation of programs by Machines ; rather, we are interested in those definitions of the program concept which are relevant to the main activity of the mathematician, that is, that of performing proofs.

The link between our intuitive understanding of programs and a possible answer to the previous question is given by the well-known mathematical concept of homogeneous binary relation. More precisely, a program might be said to establish a relationship between the values of its variables before and after its execution. For instance, let, by convention, the mathematical variable x (x') denote the value of the programming variable x before (after) the execution of the program

$$x := x + 1$$

If this is the case, then these two values are obviously related to each other by the following condition

$$x' = x + 1$$

It is traditional, in Mathematics, to write the form

$$x \; R \; x'$$

for expressing that x and x' are related by a certain relation R ; consequently we shall write

$$x\ (x := x + 1)\ x'$$

for expressing that x and x' are "related" by the program

$$x := x + 1$$

Conversely, any homogeneous binary relation will be said to <u>be</u> a "program". This apparently innocent statement enlarges considerably the usual notion of program : in particular such "programs" are no longer necessarily associated with the concept of computation. As another consequence, all usual mathematical operations on homogeneous binary relations become "automatically" operations on programs. For instance, here is a list of some of these operations together with their denotations.

Operation Name	Denotation	Notes
Identity	<u>skip</u>	
Composition	$P_1 ; P_2$	(1)
Guarding	$C \rightarrow P$	(1),(2)
Filtering	$P \rightarrow C$	(1),(2)
Alternation	$P_1 \, \Box \, P_2$	(1)
Conjunction	$P_1 \,\&\, P_2$	(1)
Simultaneity	P_1 , P_2	(3)
Hiding	<u>begin</u> y: s • P<u>end</u>	(4),(5)
Iterate	P^n	(1)
Closure	P^*	(1)

Notes : (1) The letters P, P_1 and P_2 denote programs working with a single variable x

(2) The letter C denotes a predicate of the single variable x

(3) The letter P_1 denotes a program working with a variable x ; the letter P_2 denotes a program working with a variable y

(4) The letter S denotes a set expression within which y is not free ; the letter P denotes a program working with both variables x and y

These operations can be given a mathematical definition in terms of the predicate

$$x\ P\ x'$$

as shown by the following diagram

Predicate	Definition	Notes
$x\ \underline{\text{skip}}\ x'$	$x = x'$	
$x\ (P_1\ ;\ P_2)\ x'$	$\exists z{:}S \bullet ((x\ P_1\ x')[z/x']\ \&\ (x\ P_2\ x')[z/x])$	(1),(2)
$x\ (C \rightarrow P)\ x'$	$C\ \&\ x\ P\ x'$	
$x\ (P \rightarrow C)\ x'$	$x\ P\ x'\ \&\ C[x'/x]$	(1)
$x\ (P_1 \square P_2)\ x'$	$x\ P_1\ x' \vee x\ P_2\ x'$	
$x\ (P_1 .\&\ P_2)\ x'$	$x\ P_1\ x'\ \&\ x\ P_2\ x'$	
$(x,y)(P_1\ ,\ P_2)(x',y')$	$x\ P_1\ x'\ \&\ y\ P_2\ y'$	
$x\ \underline{\text{begin}}\ y{:}T \bullet P\ \underline{\text{end}}\ x'$	$\exists y,y'{:}T \bullet (x,y)\ P\ (x',y')$	
$x\ P^n\ x'$	$x\ R^n\ x'$	(3),(4)
$x\ P^*\ x'$	$\exists n{:}\mathbb{N} \bullet x\ P^n\ x'$	

Notes : (1) Postfixing a predicate with the form "[expression/x]" corresponds to the substitution of "expression" for all free occurences of the letter "x" in the predicate in question

(2) The values of the variable x are supposed to lie within a certain set S

(3) The letter R denotes the relation "$\{x,x':S \mid x\ P\ x'\}$"

(4) The iterate R^n of an homogeneous binary relation R is a supposedly known concept.

All these definitions might be generalized to programs working with several variables.

The traditional assignment operator ":=" leads to the following predicate

$$x\ (x := \text{expression})\ x'$$

which is equivalent to

$$x' = \text{expression}$$

such an assignment might be generalized by writing any expression on the left hand side of the operator. Consequently, the predicate

$$x\ (\text{expression1} := \text{expression2})\ x'$$

is equivalent to

$$\text{expression1}[x'/x] = \text{expression2}$$

It might also be generalized by replacing the "=" sign in the assignment operator ":=" by any relational symbol. For instance, the predicate

$$x\ (x :> x)\ x'$$

is equivalent to

$$x' > x$$

Likewise, the predicate

$$x\ (x :\in \text{set-expression})\ x'$$

is equivalent to

$$x \in \text{set-expression}$$

All these definitions might lead to the proof of very simple theorems. For instance, let PROG be the following program

$$(x + 1)^2 \leq m \quad \rightarrow \quad x := x + 1$$

and let Q_n be the following predicate

$$x' = x + n \quad \& \quad \forall p : 1 \cdot\cdot n \cdot (x + p)^2 \leq m$$

The following predicates are obviously true

$$x \ \text{PROG}^0 \ x' \Leftrightarrow Q_0$$
$$x \ \text{PROG}^1 \ x' \Leftrightarrow Q_1$$

Now suppose

$$x \ \text{PROG}^n \ x' \Leftrightarrow Q_n$$

Consequently

$$x \ \text{PROG}^{n+1} \ x' \quad \Leftrightarrow \quad \exists z : \mathbb{N} \cdot (Q_n[z/x'] \ \& \ Q_1[z/x])$$

yielding, after straightforward calculations

$$x \ \text{PROG}^{n+1} \ x' \quad \Leftrightarrow \quad Q_{n+1}$$

As a consequence, we have, by induction, for all n

$$x \ \text{PROG}^n \ x' \Leftrightarrow Q_n$$

and also

$$(x \ \text{PROG}^n \ x')[0/x] \quad \Leftrightarrow \quad (x' = n \quad \& \quad n^2 \leq m)$$

It is traditional to reason about programs by using various approaches. For instance, we might ask whether a program terminates, or whether a program is less-defined than another one, or whether a program transforms a certain predicate into another one. All such facts are denoted by using the following forms

Termination	$\underline{wp}\ P$
Less-definedness	$P_1 \sqsubseteq P_2$
Predicate Transformation	$\{C_1\}\ P\ \{C_2\}$

As for previous forms, these forms might be given a mathematical definition in terms of the predicate

$$x\ P\ x'$$

as follows

$\underline{wp}\ P$	$\exists x':S \bullet x\ P\ x'$
$P_1 \sqsubseteq P_2$	$\forall x':S \bullet (x\ P_1\ x' \Longrightarrow x\ P_2\ x')$
$\{C_1\}\ P\ \{C_2\}$	$C_1 \Longrightarrow \forall x':S \bullet (x\ P\ x' \Longrightarrow C_2[x'/x])$

A number of well known elementary theorems might be proved concerning these predicates. For instance, it is very simple to prove

$$\underline{wp}\ (C \rightarrow P) \iff (C\ \&\ \underline{wp}\ P)$$

Consequently, we have

$$\underline{wp}\ \text{PROG} \iff (x + 1)^2 \leq m$$

Certain classical program constructs might be defined in terms of the above definitions. For instance, the loop of a program P, denoted after E.W. Dijkstra

$$\underline{\text{do}}\ P\ \underline{\text{od}}$$

is defined as

$$P^* \rightarrow \neg\ \underline{\text{wp}}\ P$$

Consequently, the predicate

$$x\ (\underline{\text{do}}\ P\ \underline{\text{od}})\ x'$$

is equivalent to

$$\exists x : \mathbb{N} \bullet x\ P^n\ x' \quad \& \quad (\neg \underline{\text{wp}}\ P)[x'/x]$$

For instance, the predicate

$$x\ (\underline{\text{do}}\ \text{PROG}\ \underline{\text{od}})\ x'$$

is equivalent to

$$\exists n : \mathbb{N} \bullet Q_n \quad \& \quad m < (x' + 1)^2$$

yielding, for x = 0

$$\exists n : \mathbb{N} \bullet x' = n \quad \& \quad n^2 \leq m < (x' + 1)^2$$

that is

$$x'^2 \leq m < (x' + 1)^2$$

We have then just proved that the program

$$\underline{\text{do}}\ (x+1)^2 \leq m \rightarrow x := x + 1\ \underline{\text{od}}$$

results, when "started" with $x = 0$, in the integer square root of the parameter m.

2. - WHAT IS A MACHINE ?

Over the past few years, it has become fashionable to distribute computer processing among several inter-connected machines. An interesting challenge, raised by the building of such structures, is the discovery of a "good" mathematical model that could help prove their properties in a systematic way.

In order to do so, one of the first questions that comes to mind is obviously the following

What is a Machine ?

To such a very general question, an equally general answer can only be given ; in fact, observing the run of a machine through its "visible" registers, and recording <u>what</u> happens and also <u>when</u> it happens, results in the production, after the machine has stopped, of a complete record of its "visible" behavior : in other words, the machine has been a mere producer of its past.

More precisely, we shall consider the <u>history</u> of <u>events</u> which occur on each visible register (henceforth called channel) of a running machine. Such histories will be represented by finite functions, the domains of which are subsets of the positive Natural Numbers. In fact, these numbers simply denote some measurements of the time (in some arbitrary unit) <u>relative</u> to the starting date of the machine.

For instance, we might observe that the following history records the past of a machine with a single channel "in".

$$in = \{1 \rightarrow a\ ,\ 3 \rightarrow b\ ,\ 4 \rightarrow c\ ,\ 7 \rightarrow d\}$$

Hence, at time 1, "event a" has occured, then, later, at time 3, "event b" has occured, and so on.

Another machine, called A, has "produced" the following pasts on its channels "in" and "out"

$$in = \{1 \rightarrow a\ ,\ 3 \rightarrow b\ ,\ 4 \rightarrow c\ ,\ 7 \rightarrow d\}$$

$$out = \{2 \rightarrow a\ ,\ 5 \rightarrow c\ ,\ 8 \rightarrow d\ ,\ 9 \rightarrow d\}$$

On these records, we can see that each event occuring on channel "out", at, say, time t, has already occured on channel "in" ; more precisely, this event was the last one to occur, before t, on channel "in".

The following two histories are produced by a machine called B

$$in = \{1 \rightarrow a \;,\; 3 \rightarrow b \;,\; 4 \rightarrow c \;,\; 7 \rightarrow d\}$$

$$out = \{2 \rightarrow a \;,\; 4 \rightarrow b \;,\; 8 \rightarrow c \;,\; 9 \rightarrow d\}$$

On these records, we can observe that all events occuring on channel "in", later occur on channel "out" in the same time order. Also, machine B, unlike machine A, may "produce" two events at the same time on distinct channels.

Our next example shows the following record for a machine called C

$$in = \{1 \rightarrow a \;,\; 3 \rightarrow b \;,\; 4 \rightarrow c \;,\; 7 \rightarrow d\}$$

$$out = \{2 \rightarrow a \;,\; 6 \rightarrow c \;,\; 7 \rightarrow b \;,\; 9 \rightarrow d\}$$

This machine looks like our previous machine B in that all events occuring on channel "in" later occur on channel "out" ; however, the time order is not preserved any more.

Our final small example at this stage is given by the machine D the channel histories of which are

$$in = \{1 \rightarrow a \;,\; 3 \rightarrow b \;,\; 6 \rightarrow c \;,\; 8 \rightarrow d\}$$

$$out = \{2 \rightarrow a \;,\; 5 \rightarrow b \;,\; 7 \rightarrow c \;,\; 9 \rightarrow d\}$$

As we can see, this machine is a straightforward copying machine.

3. - SPECIFYING MACHINES

In the previous section, we have seen how the external behavior of a machine can be described, once this machine has stopped, by the histories of its channels. More generally, a machine can be specified by exhibiting some characteristic properties of its channel histories. In this section, we shall show how such specifications might be written.

In order to do this, we shall first define a number of elementary functions on histories and then use these functions to formally specify the very simple examples described in the previous section.

We need to define the set of all histories the events of which belong to a given set S. It is denoted

$$\mathrm{hist}(S)$$

and is defined as

$$\{h:\mathbb{N}_1 \nrightarrow S \mid \mathrm{finite}(h)\}$$

We write

$$\langle\ \rangle_S$$

for the empty history built on S, so that the set of non-empty histories

$$\mathrm{hist1}(S)$$

is defined as

$$\mathrm{hist}(S) - \{\langle\ \rangle_S\}$$

The two following functions define the domain and cardinality of an history

$$\mathrm{dom} : \mathrm{hist}(S) \rightarrow \mathcal{P}(S)$$

$$\# : \mathrm{hist}(S) \rightarrow \mathbb{N}$$

Two other functions transform an history into corresponding finite sequences of events and times, both ordered in ascending value of time

$$\mathrm{trace} : \mathrm{hist}(S) \rightarrow \mathrm{seq}(S)$$

$$\mathrm{when} : \mathrm{hist}(S) \rightarrow \mathrm{seq}(\mathbb{N})$$

Finally, the two following functions give the "value" and the "index" of the last event that has occured since a given time

$$\text{value} : (\text{hist}_1(S) \times \mathbb{N}) \nrightarrow S$$

$$\text{index} : (\text{hist}(S) \times \mathbb{N}) \longrightarrow \mathbb{N}$$

For instance, on the following history h

$$\{2 \to a\ ,\ 3 \to b\ ,\ 4 \to c\ ,\ 7 \to d\}$$

we have

$$\text{dom}(h) = \{2\ ,\ 3\ ,\ 4\ ,\ 7\}$$

$$h\,\# = 4$$

$$\text{trace}(h) = \langle a\ ,\ b\ ,\ c\ ,\ d\rangle$$

$$\text{when}(h) = \langle 2\ ,\ 3\ ,\ 4\ ,\ 7\rangle$$

$$\text{value}(h,5) = c$$

$$\text{index}(h,5) = 3$$

$$\text{index}(h,1) = 0$$

Equipped with these tools, we can write formal statements corresponding to our previous elementary example machines. For instance, our machine A might be specified by the three following predicates, the conjunction of which is named "MEMORY_S"

$$\text{in} = \langle\ \rangle_S \implies \text{out} = \langle\ \rangle_S$$

$$\text{dom}(\text{in}) \cap \text{dom}(\text{out}) = \emptyset_{\mathbb{N}}$$

$$\forall t{:}\text{dom}(\text{out}) \bullet \text{out}(t) = \text{value}(\text{in},t)$$

Likewise, the two following predicates the conjunction of which is named "BUFFER_S", specify our machine B.

$$\text{trace}(\text{in}) = \text{trace}(\text{out})$$

$$\forall i{:}1 \bullet\bullet \text{out}\,\# \bullet \text{when}(\text{in})(i) < \text{when}(\text{out})(i)$$

The machine C can be characterized by the very existence of a one-to-one mapping f from "dom(in)" on to "dom(out)" such that, for all time t in "dom(in)", the following predicate is true

$$in(t) = out(f(t)) \;\; \& \;\; t < f(t)$$

For instance, for the following observation of C

$$in = \{1 \to a \; , \; 3 \to b \; , \; 4 \to c \; , \; 7 \to d\}$$

$$out = \{2 \to a \; , \; 6 \to c \; , \; 7 \to b \; , \; 9 \to d\}$$

we have the following mapping f

$$f = \{1 \to 2 \; , \; 3 \to 7 \; , \; 4 \to 6 \; , \; 7 \to 9\}$$

Consequently the predicate "BAG_S", which characterizes C, is

$$\exists f : dom(in) \rightarrowtail\!\!\!\twoheadrightarrow dom(out) \cdot \forall t : dom(in) \cdot (in(t) = out(f(t) \;\; \& \;\; t < f(t))$$

Finally, the machine D might be characterized as being a MEMORY and at the same time a BUFFER ; consequently its specification is given by the following predicate named "$COPYER_S$"

$$MEMORY_S \;\; \& \;\; BUFFER_S$$

Given a Machine M and a predicate P (with a free channel variable c) which represents a certain specification, we shall denote the fact

M is a P

by writing down the following form

$$M \xrightarrow{c:S} P$$

Note : Such a form might be generalized in an obvious way to more channels.

For instance, our example machines A, B, C and D are respectively a MEMORY, a BUFFER, a BAG and a COPYER, that is

$$A \xrightarrow{\text{in,out:S}} \{\text{MEMORY}_S\}$$

$$B \xrightarrow{\text{in,out:S}} \{\text{BUFFER}_S\}$$

$$C \xrightarrow{\text{in,out:S}} \{\text{BAG}_S\}$$

$$D \xrightarrow{\text{in,out:S}} \{\text{COPYER}_S\}$$

Now, if a predicate Q follows from a predicate P and if a machine M "is a P", then it also "is a Q". This remark can be formally written in the form of following proof rule

$$\frac{\begin{array}{c} M \xrightarrow{c:S} \{P\} \\ P \vdash Q \end{array}}{M \xrightarrow{c:S} \{Q\}}$$

Note : For the moment such a proof rule is only stated ; we do not yet have the possibility of proving it rigorously.

For instance, the above proof rule leads to the following obvious theorems

$$B \xrightarrow{\text{in,out:S}} \{\text{BAG}_S\}$$

$$D \xrightarrow{\text{in,out:S}} \{\text{MEMORY}_S\}$$

$$D \xrightarrow{\text{in,out:S}} \{\text{BUFFER}_S\}$$

stating that B, which is a BUFFER, is also a BAG and that D, which is a COPYER, is also a MEMORY and a BUFFER.

4. CONNECTING MACHINES

We have just seen how to assert that a machine M "is" a certain predicate P (i.e. follows the specification described by a certain predicate P). This fact is denoted

$$M \xrightarrow{c:S} \{P\}$$

We introduce now a form expressing the connection of two machines M_1 and M_2 through a common channel c. Such a connection denoted

$$M_1 \xrightarrow{c:S} M_2$$

results in a new machine, the visible channels of which are those of M_1 as well as those of M_2 apart from the common channel c which is hidden.

Now, suppose that a machine M_1 (supposedly working with channel c and channel d) "is a P_1", that is

$$M_1 \xrightarrow{c:S\ ;\ d:T} \{P_1\}$$

Likewise, suppose that a machine M_2 (supposedly working with channel c and channel e) "is a P_2", that is

$$M_2 \xrightarrow{c:S\ ;\ e:U} \{P_2\}$$

We would like to find a predicate P_3, within which the letter c is not free, such that the new machine

$$M_1 \xrightarrow{c:S} M_2$$

"be a P_3", that is

$$(M_1 \xrightarrow{c:S} M_2) \xrightarrow{d:T\ ;\ e:U} \{P_3\}$$

Intuitively, the new machine is "almost" a P_1, likewise it is "almost" a P_2, consequently it is "almost" a P_1 & P_2. In the three cases we have written "almost" because the corresponding predicates still contain the letter c. Now, if P_3, within which c is not free, follows from P_1 & P_2, that is, if

$$P_1 \ \&\ P_2 \vdash P_3$$

then, obviously, the new machine "is a P_3". This informal reasonning can be formally stated in the form of the following proof rule

$$\frac{\begin{array}{c} M_1 \xrightarrow{c:S\ ;\ d:T} \{P_1\} \\ M_2 \xrightarrow{c:S\ ;\ e:U} \{P_2\} \\ P_1\ \&\ P_2 \vdash P_3 \end{array}}{(M_1 \xrightarrow{c:S} M_2) \xrightarrow{d:T\ ;\ e:U} \{P_3\}}$$

Note : Again, this proof rule has not yet been formally proved.

As a trivial example, we shall now connect two buffers and prove that the resulting machines is also a buffer. In the previous section, we introduced the following predicate $BUFFER_S$

$$trace(in) = trace(out)$$

$$\forall i:1\cdot\cdot out\,\#\cdot\ when(in)(i) < when(out)(i)$$

Now, let B_1 (working with channels "in" and "c") and B_2 (working with channels "c" and "out") be two buffers, that is

$$B_1 \xrightarrow{in,c:S} \{BUFFER_S[c/out]\}$$
$$B_2 \xrightarrow{c,out:S} \{BUFFER_S[c/in]\}$$

As the following theorem is obvious

$$BUFFER_S[c/out]\ \ \&\ \ BUFFER_S[c/in]\ \vdash\ BUFFER_S$$

then we have, as expected

$$(B_1 \xrightarrow{c:S} B_2) \xrightarrow{in,out:S} \{BUFFER_S\}$$

For instance, the following observation of B_1 and B_2 is compatible with their connection through channel c

$$in = \{1 \to a\ ,\ 3 \to b\ ,\ 5 \to c\}$$
$$c = \{2 \to a\ ,\ 4 \to b\ ,\ 6 \to c\}$$
$$out = \{3 \to a\ ,\ 5 \to b\ ,\ 7 \to c\}$$

An interesting machine, that we shall call E, is one that merges or distributes non-deterministically one of its channels c into two others, c_1 and c_2. It can be specified by the predicate MD_S (for

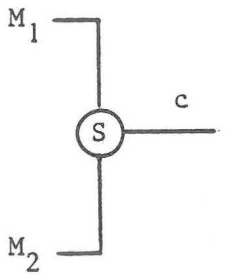
Merger-Distributor), which is the conjunction of the two following predicates

$$\mathrm{dom}(c_1) \cap \mathrm{dom}(c_2) = \emptyset_S$$

$$c = c_1 \cup c_2$$

Note : This machine can be generalized in an obvious way so as to merge or distribute one channel into more than two channels.

Now, given two machines M_1 and M_2 with a common channel c, then the form

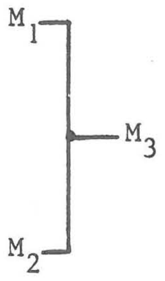


in an abbreviation for

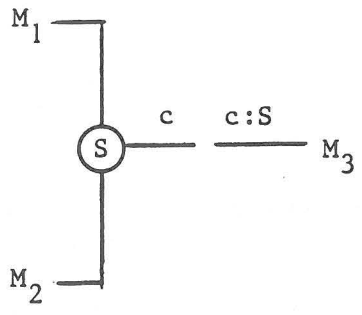
$$M_1[c_1/c] \xrightarrow{c_1:S} E \xrightarrow{c_2:S} M_2[c_2/c\ ]$$

Note : "$M_1[c_1/c]$" and "$M_2[c_2/c]$" denote machines M_1 and M_2 with a corresponding renaming of channel c.

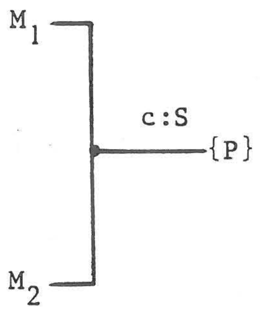
Likewise the form



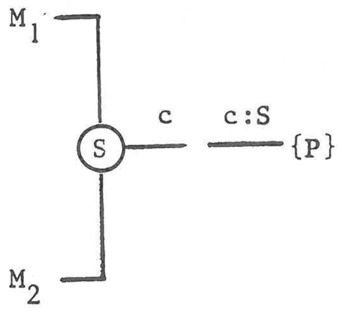
is an abbreviation for



and the form



is an abbreviation for



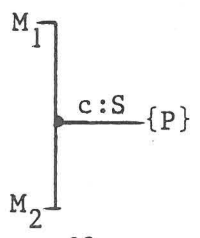
Note : All these forms might be generalized to more machines and to more channels.

The following obvious proof rule follows from these definitions

$$M_1 \;\underline{\quad c : S \quad}\; \{P_1\}$$

$$M_2 \;\underline{\quad c : S \quad}\; \{P_2\}$$

$$L \vdash P$$

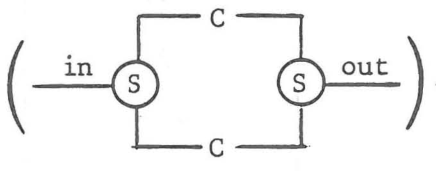


where the predicate L, called the linkage condition, is

$$P_1[c_1/c] \quad \& \quad P_2[c_2/c] \quad \& \quad c = c_1 \cup c_2 \quad \& \quad \mathrm{dom}(c_1) \cap \mathrm{dom}(c_2) = \emptyset_S$$

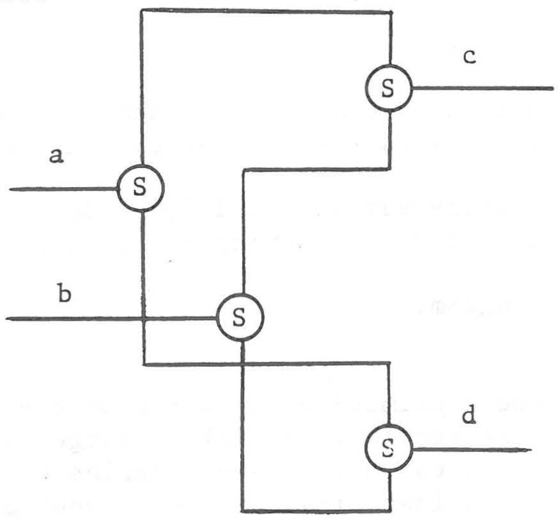
Using this proof rule, it is very easy to prove that the connection of two BAGs in this way, results in another BAG, that is

$$C \xrightarrow{in,\ out\ :\ S} \{BAG_S\}$$



$$\xrightarrow{in,\ out\ :\ S} \{BAG_S\}$$

Our last example at this stage, is a channel switch which can be defined as follows



for which one might easily prove

$$a \cup b = c \cup d$$

$$\mathrm{dom}(a) \cap \mathrm{dom}(b) = \mathrm{dom}(c) \cap \mathrm{dom}(d)$$

in other words there is no loss nor duplication of information.

5. - CONSTRUCTING MACHINES

So far, we have only considered machines from an external point of view ; in this section, on the contrary, we shall study the inside of machines, that is develop primitives able to express their internal behaviors.

Curiously enough, the number of such primitives can be reduced to the bare minimum ; in fact, we only need to express the writing on or the reading from a channel, the time atomicity of an action and, finally, the starting of a machine. The following diagram shows the denotation of these operations.

Operation	Denotation	Notes
Reading	$\underline{\text{read}}_S\ c$	(1)
Writing	$c\,!\ := \text{exp}$	(2)
Atomicity	$< P >$	(3)
Starting	$\boxed{P}$	(3)

Notes :

(1) c denotes a history variable and S denote a set expression compatible with the carrier set of the history variable c.

(2) c denotes a history variable and "exp" denote an element expression compatible with the carrier set of the history variable c.

(3) P denotes a program.

Before defining these primitives in terms of the program features defined in the first section, we shall enlarge our previous set of tools dealing with histories. We shall define three functions yielding respectively the last time, the last event and the past of a given history. These functions are the following

$$\downarrow \ : \ \text{hist}(S) \rightarrow \mathbb{N}$$

$$? \ : \ \text{hist1}(S) \rightarrow S$$

$$- \ : \ \text{hist1}(S) \rightarrow \text{hist}(S)$$

For instance, on the following history h

$$\{1 \rightarrow a,\ 3 \rightarrow b,\ 7 \rightarrow c\}$$

we have

$$h\downarrow = 7$$

$$h? = c$$

$$h^{-} = \{1 \rightarrow a, 3 \rightarrow b\}$$

and also, by convention

$$< >_S \downarrow = 0$$

We are now ready to define our primitives. We shall suppose that a variable t, denoting time, is locally available.

Primitive	Definition	Notes
$\underline{\text{read}}_S\, c$	$c\downarrow < t \rightarrow c :\in \{c \cup \{t \rightarrow x\} \mid x : S\}$	(1)
$c! := exp$	$c\downarrow < t \rightarrow c := c \cup \{t \rightarrow exp\}$	
$<P>$	$P \, , \, t :> t$	(2)
$\boxed{P}$	$\underline{\text{begin}}\ t : \mathbb{N}1 \bullet P\ \underline{\text{end}}$	(3)

Notes :

(1) c is not free in S.

(2) The program P should not modify the program variable t.

(3) The program P can only work with the program variable t, and with <u>history variables</u>.

As a consequence we have the following theorems

$$c(\underline{\text{read}}_S c)c' \Leftrightarrow (c\downarrow < t \quad \& \quad c'\downarrow = t \quad \& \quad c'? \in S \quad \& \quad c'^{-} = c)$$

$$c(c! := exp)c' \Leftrightarrow (c\downarrow < t \quad \& \quad c'\downarrow = t \quad \& \quad c'? = exp \quad \& \quad c'^{-} = c)$$

$$(c,t)(\langle P\rangle)(c',t') \Leftrightarrow (c \; P \; c' \quad \& \quad t' > t)$$

$$c(\boxed{P})\,c' \Leftrightarrow \exists t,t' : \mathbb{N}1 \bullet (c,t)\; P\; (c',t')$$

The connection primitive already introduced and denoted

$$M_1 \xrightarrow{c \;:\; S} M_2$$

might now be formally defined as follows

$$\underline{\text{begin}}\; c : \text{hist}(S) \bullet c = \langle\;\rangle_S \rightarrow (M_1 \;\&\; M_2)\; \underline{\text{end}}$$

Note : M_1 and M_2 denotes programs working with <u>history variables only</u>.

Suppose that machine M_1 works with channels c and d, and that machine M_2 works with channels c and e, then it is very simple to prove

$$(d,e)(M_1 \xrightarrow{c:S} M_2)(d',e') \Leftrightarrow \exists c,c' : \text{hist}(S) \bullet (c = \langle\;\rangle_S \quad \& \quad (c,d)M_1(c',d') \quad \& \quad (c,e)M_2(c',e$$

It only remains for us to define the predicate

$$M \xrightarrow{c \;:\; S} \{P\}$$

This can be done as follows

$$c = \langle\;\rangle_S \Longrightarrow \forall c' : \text{hist}(S) \bullet (c \; M \; c' \Longrightarrow P[c'/c])$$

In other words, it is exactly

$$\{c = \langle\;\rangle_S\} \;\; M \;\; \{P\}$$

From these definitions, it is very simple to prove the already stated proof rules concerning machine connections.

We are also ready to construct a memory, a buffer or a copyer. Let R, W_1 and W_2 be the following programs

$$< \underline{\text{read}}_S \ \text{in} >$$

$$\text{in} \neq < >_S \ \rightarrow \ < \text{out}! := \text{in}?>$$

$$\text{out}\# < \ \text{in}\# \ \rightarrow \ < \text{out}! \ := \text{trace}(\text{in}) \ (\text{out}\# + 1)>$$

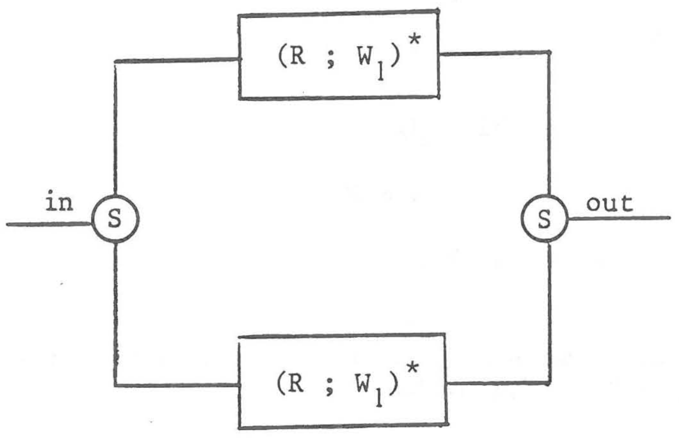
Then it is very simple to prove

$$\boxed{(R \;\;\square\;\; W_1)^*}\;\underline{\;\text{in, out : S}\;}\;\{\text{MEMORY}_S\}$$

$$\boxed{(R \;\;\square\;\; W_2)^* \;\; ; \;\; \underline{\text{do}} \;\; W_2 \;\; \underline{\text{od}}}\;\underline{\;\text{in, out : S}\;}\;\{\text{BUFFER}_S\}$$

$$\boxed{(R \;\; ; \;\; W_1)^*}\;\underline{\;\text{in, out : S}\;}\;\{\text{COPYER}_S\}$$

As a copying machine is also a BAG (since it is a BUFFER) then we can construct a BAG as follows



And finally our Merger/Distributor can be built from the program

$$c_1 \ \gg_S \ c_2$$

which is

$$<\underline{\text{read}}_S c_1 \ \ ; \ \ c_2! \ := \ c_1?>$$

so that machine E is

$$\boxed{\begin{array}{l}(c_1 \gg_S c \;\square\; c_2 \gg_S c)^* \\ \square \\ (c \gg_S c_1 \;\square\; c \gg_S c_2)^*\end{array}}$$

6. - AN EXAMPLE

In this section, we shall study a more elaborate - although still very simple - example ; it is a variant of the AB transmission protocol described in (14).

Let us first consider a machine called SND (for sender) which works with three channels "in", "p", and "k" :

. Channel "in" conveys "messages" belonging to a certain set M

. Channel "p" conveys "packets" each of which is made of of a serial number and a message

. Channel "k" conveys "acknowledgments" under the form of serial numbers.

Consequently we have

$$\begin{array}{rcl} in & : & hist(M) \\ p & : & hist(\mathbb{N} \times M) \\ k & : & hist(\mathbb{N}) \end{array}$$

Given a packet "(n,m)", we shall use two projection functions called "message" and "number" and such that

$$\begin{array}{lcl} message(n,m) & = & m \\ number(n,m) & = & n \end{array}$$

The machine SND <u>may</u> read channel "in" only when it receives an acknowledgment, that is

(S1) $\quad dom(in) \subset dom(k)$

Moreover, this acknowledgment must be equal to the number of messages read so far on channel "in", that is

(S2) $\forall t : \mathrm{dom}(in) \bullet k(t) = \mathrm{index}(in, t-1)$

The machine SND may send packets made of the last message read if any, together with the corresponding serial number, that is

(S3) $\forall t : \mathrm{dom}(p) \bullet \mathrm{index}(in, t) \neq 0$

(S4) $\forall t : \mathrm{dom}(p) \bullet p(t) = (\mathrm{index}(in,t), \mathrm{value}(in,t))$

Moreover SND must send the last message read if any, that is

(S5) $in\downarrow \ \leq \ p\downarrow$

Finally, the reception of an acknowledgment and the sending of a packet excludes each other, that is

(S6) $\mathrm{dom}(k) \cap \mathrm{dom}(p) = \emptyset_{\mathbb{N}}$

The following diagram summarizes our specification of SND

(S1)	$\mathrm{dom}(in) \subset \mathrm{dom}(k)$
(S2)	$\forall t : \mathrm{dom}(in) \bullet k(t) = \mathrm{index}(in, t-1)$
(S3)	$\forall t : \mathrm{dom}(p) \bullet \mathrm{index}(in,t) \neq 0$
(S4)	$\forall t : \mathrm{dom}(p) \bullet p(t) = (\mathrm{index}(in,t), \mathrm{value}(in,t))$
(S5)	$in\downarrow \ \leq \ p\downarrow$
(S6)	$\mathrm{dom}(k) \cap \mathrm{dom}(p) = \emptyset_{\mathbb{N}}$

We then consider a second machine, called RCV, also working with three channels "out", "p" and "k", of the following type

$$\begin{aligned} out &: hist(M) \\ p &: hist(\mathbb{N} \times M) \\ k &: hist(\mathbb{N}) \end{aligned}$$

The machine RCV may write on channel "out" only when it receives a packet, that is

(R1) $$dom(out) \subset dom(p)$$

However, RCV <u>must</u> do so exactly when the received packet conveys a number the value of which is equal to the number of messages already written on channel "out" plus one, that is

(R2) $$\forall t : dom(p) \bullet (number(p(t)) = index(out,t-1)+1 \Leftrightarrow t \in dom(out))$$

Then the message written on "out" is that of the received packet, that is

(R3) $$\forall t : dom(out) \bullet out(t) = message(p(t))$$

The machine RCV <u>may</u> send acknowledgments which are numbers equal to the number of messages written on channel "out" (this number might then be 0), that is

(R4) $$\forall t : dom(k) \bullet k(t) = index(out,t)$$

Finally, the sending of an acknowledgment and the reception of a packet exclude each other, that is

(R5) $$dom(k) \cap dom(p) = \emptyset_{\mathbb{N}}$$

The following diagram summarizes our specification of RCV.

$$
\boxed{\begin{array}{ll}
(R1) & dom(out) \subset dom(p) \\
(R2) & \forall t : dom(p) \bullet (number(p(t)) = index(out,t-1) + 1 \Leftrightarrow t \in dom(out)) \\
(R3) & \forall t : dom(out) \bullet out(t) = message(p(t)) \\
(R4) & \forall t : dom(k) \bullet k(t) = index(out,t) \\
(R5) & dom(k) \cap dom(p) = \emptyset_{\mathbb{N}}
\end{array}}
$$

We shall connect then two machines thus forming a new machine called SR defined as follows

$$SND \; \frac{k : \mathbb{N} \; ; \; p : \mathbb{N} \times M}{} \; RCV$$

We would like to prove that this machine is a COPYER. In fact, it is very simple to prove that SR is indeed a MEMORY ; the predicate

$$in = < >_M \Longrightarrow out = < >_M$$

follows from (S3) and (R1), then

$$dom(in) \cap dom(out) = \emptyset_{\mathbb{N}}$$

follows from (S6), (S1) and (R1), and finally

$$\forall t : dom(out) \bullet out(t) = value(in,t)$$

follows from (R3) and (S4), consequently we have

$$SR \; \frac{in, \; out : M}{} \; \{MEMORY_M\}$$

We shall now prove that SR is a BUFFER. In fact, (R2), (R3) and (S4) lead to

(1) $\forall t : dom(out) \bullet (index(out,t),out(t)) = (index(in,t),value(in,t))$

yielding

(2) $$\text{trace(out)} \subset \text{trace(in)}$$

However, from (S2) and (R4), we obtain

$$\forall t : \text{dom(in)} \bullet \text{index(in,t)} = \text{index(out,t)} + 1$$

yielding

(3) $$\text{in\#} = \text{index(out, in}\downarrow) + 1$$

therefore, after (1), we obtain

(4) $$\text{out\#} \leq \text{in\#} \leq \text{out\#} + 1$$

Now, suppose

(H1) $$\text{in\#} = \text{out\#} + 1$$

From (R1), we obtain

(5) $$\text{out}\downarrow \leq p\downarrow.$$

suppose moreover

(H2) $$\text{out}\downarrow < p\downarrow$$

Consequently

(6) $$\text{index(out, } p\downarrow - 1) = \text{out\#}$$

Therefore, after (S5), (S4) and (H1), we obtain

$$\text{number } (p\,?) = \text{in\#} = \text{out\#} + 1 = \text{index(out, } p\downarrow - 1) + 1$$

consequently, after (R2)

$$p\downarrow \in \mathrm{dom}(out)$$

this contradicting (H2), therefore, after (5)

$$out\downarrow = p\downarrow$$

and consequently, after (S4), (S5) and (R2)

$$number(p?) = in\# = out\#$$

thus contrading (H1), therefore after (4)

$$in\# = out\#$$

Consequently, after (2), we obtain

(7) $$trace(in) = trace(out)$$

We already proved

$$\mathrm{dom}(in) \cap \mathrm{dom}(out) = \emptyset_{\mathbb{N}}$$

therefore, after (1)

$$\forall i : 1 \mathrel{..} out\# \bullet when(in)(i) < when(out)(i)$$

which leads, after (7), to

$$SR \xrightarrow{in,\ out\ :\ M} \{BUFFER_M\}$$

It remains for us to construct both machines SND and RCV ; this can be done respectively as follows

```
(< $\underline{read}_{\mathbb{N}}$ k  ;  ($\underline{skip}$ □ k? = in#  →  $\underline{read}_M$ in)>
□
 < in ≠ $< >_M$  →  p! := (in#, in?)>)$^*$ ;
(< in ≠ $< >_M$  →  p! := (in#, in?)>  □  in = $< >_M$  →  $\underline{skip}$)
```

```
(< $\underline{read}$ $p_{\mathbb{N} \times M}$      ;
   (number(p?) = out# + 1  →  out! := message (p?)
   □
    number(p?) ≠ out# + 1  →  $\underline{skip}$)>
□
 < k! := out# >)$^*$
```

It is not difficult to prove that these machines are built according to the previous specifications, and also that they are able to transmitt sequences (of message) of any length. For instance, suppose that "k", "in", "p" and "out" all are empty histories and that "k'", "in'", "p'" and "out'" are histories such that

$$k' = \bigsqcup_{i:1..n} \{2i - 1 \rightarrow i - 1\}$$

$$dom(in') = dom(k')$$

$$p' = \bigsqcup_{i:1..n} \{2i \rightarrow (i, in(2i - 1))\}$$

$$out' = \bigsqcup_{i:1..n} \{2i \quad in(2i - 1)\}$$

then for each natural number n we have obviously

$$(in,k,p)\ SND\ (in',k',p') \quad \& \quad (out,k,p)\ RCV\ (out',k',p')$$

7. - DISTRIBUTING A PROGRAM

As a final example we shall show how one might distribute a program among several machines. The following algorithm, attributed to E.W. Dijkstra, transforms two finite and disjoint sets of Natural Numbers S_1 and S_2 into two other sets S_1' and S_2' such that

$$S_1' \cup S_2' = S_1 \cup S_2$$
$$S_1' \cap S_2' = \emptyset_{\mathbb{N}}$$
$$\max(S_1') < \min(S_2')$$

In order to obtain this result one repeatedly exchanges the maximum of S_1 and the minimum of S_2 until the desired condition is met.

Let the predicate C and the programs P_1 and P_2 respectively be

$$\min(S_2) < \max(S_1)$$
$$S_1 := (S_1 - \{\max(S_1)\}) \cup \{\min(S_2)\}$$
$$S_2 := (S_2 \cup \{\max(S_1)\}) - \{\min(S_2)\}$$

and let the program P and the predicate A respectively be

$$C \rightarrow (P_1 , P_2)$$
$$S_1 \in F_1(\mathbb{N}) \quad \& \quad S_2 \in F_1(\mathbb{N}) \quad \&$$
$$S_1 \cap S_2 = \emptyset_{\mathbb{N}} \quad \& \quad S_1 \cup S_2 = X$$

Note : The form $F_1(\mathbb{N})$ denotes the set of non empty and finite subsets of the set of Natural Numbers $\mathbb{N}$.

It is then very easy to prove the following theorems

$$\{A\} \quad P \quad \{A\}$$
$$\{A\} \quad \underline{\text{do}} \quad P \quad \underline{\text{od}} \quad \{A \quad \& \quad \max(S_1) < \min(S_2)\}$$

We shall now distribute the program

$$\underline{\text{do}} \quad P \quad \underline{\text{od}}$$

by "executing" both programs P_1 and P_2 on separate communicating machines. However, before doing this, we shall define the two following functions f and g, the arguments of which are respectively a set (of numbers) and two sequences (of numbers) <u>of the same size</u>.

$$f\ ,\ g\ :\ (\mathbb{P}(\mathbb{N})\ \times\ \text{seq}(\mathbb{N})\ \times\ \text{seq}(\mathbb{N}))\ \nrightarrow\ \mathbb{P}(\mathbb{N})$$

We shall define these functions recursively as follows

$$f(S\ ,\ <>_{\mathbb{N}}\ ,\ <>_{\mathbb{N}})\ =\ S$$

$$g(S\ ,\ <>_{\mathbb{N}}\ \ <>_{\mathbb{N}}\ =\ S$$

$$f(S,\ s_1 \curvearrowleft x,\ s_2 \curvearrowleft y)\ =\ f((S - \{x\})\ \cup \{y\}\ ,\ s_1\ ,\ s_2)$$

$$g(S,\ s_1 \curvearrowleft x,\ s_2 \curvearrowleft y)\ =\ g((S\ \cup \{x\})\ -\ \{y\}\ ,\ s_1\ ,\ s_2)$$

Note : The operator $\curvearrowleft$ denotes the "pushing" of an element at the end of a sequence.

Given two history variables M and m and two non empty finite sets of numbers S_1 and S_2

$$M\ ,\ m\ :\ \text{hist}(\mathbb{N})$$

$$S_1\ ,\ S_2\ :\ \mathbb{F}_1(\mathbb{N})$$

we define both sets S_3 and S_4 respectively as follows

$$f(S_1\ ,\ \text{trace}(M)\ ,\ \text{trace}(m))$$

$$g(S_2\ ,\ \text{trace}(M)\ ,\ \text{trace}(m))$$

Let P_3 and P_4 be the two following programs

$$<M!\ :=\ \max(S_3)\ ;\ \underline{\text{read}}_{\mathbb{N}}\ m>$$

$$<\ \underline{\text{read}}_{\mathbb{N}}\ M\ ;\ m!\ :=\ \min(S_4\ \cup\ \{M?\})>$$

Both programs M_1 and M_2 are now defined as follows

$$P_3 \;;\; \underline{\text{do}}\ m? \neq M? \rightarrow P_3\ \underline{\text{od}} \;;\; \langle \ell! := S_3 \rangle$$

$$P4^* \;;\; \langle u! := S_4 \rangle$$

The final machine M_3 is

$$\boxed{M_1} \xrightarrow{m\ ,\ M\ :\ \mathbb{N}} \boxed{M_2}$$

and, of course, we would like to prove the following

$$A \vdash M_3 \xrightarrow{\ell, n\ :\ \mathbb{F}_1(\mathbb{N})} \left\{ \begin{array}{l} A[\ell?/S_1\ ,\ u?/S_2] \\ \\ \max(\ell?) < \min(u?) \end{array} \right\}$$

Let M_t and m_t be the histories M and m <u>restricted to</u> the internal

$$1 \cdots t$$

and let S_{3_t} and S_{4_t} be respectively

$$f(S_1\ ,\ \text{trace}(M_t)\ ,\ \text{trace}(m_t))$$

$$g(S_2\ ,\ \text{trace}(M_t)\ ,\ \text{trace}(m_t))$$

and let finally C_1 and C_2 be respectively the conjunction of the following predicates

C1 $\quad \text{dom}(m) = \text{dom}(M)$

$\forall t : \text{dom}(m) \bullet$

$\quad (M(t) = \max(S_{3_{t-1}}) \quad \&$

$\quad t = m\downarrow \Rightarrow m(t) = M(t))$

$\ell? = S_{3_{m\downarrow}}$

C_2

$$\begin{aligned} & dom(m) = dom(M) \\ & \forall t : dom(M) \bullet \\ & \quad m(t) = max(S_{4_{t-1}} \cup \{M(t)\}) \\ & u? = S_{4_{M\downarrow}} \end{aligned}$$

It is then an easy matter to prove

$$\boxed{M_1} \quad \frac{m , M: \mathbb{N} ; \ell: \mathbb{F}_1(\mathbb{N})}{} \{C_1\}$$

$$\boxed{M_2} \quad \frac{m , M: \mathbb{N} ; u: \mathbb{F}_1(\mathbb{N})}{} \{C_2\}$$

If A_t denotes

$$A[S_{3_t}/S_1 \;,\; S_{4_t}/S_2]$$

one might also easily prove

$$C_1 \;\&\; C_2 \vdash \forall t:dom(m) \bullet A_{t-1} \Rightarrow A_t$$

yielding

$$C_1 \;\&\; C_2 \vdash A_o \Rightarrow A_{m\downarrow}$$

that is

$$A \;;\; C_1 \;\&\; C_2 \vdash A_{m\downarrow}$$

However, as

$$m? = M?$$

and also

$$m? = min(S_{4_{m\downarrow-1}} \cup \{M?\})$$

consequently, we have

$$M? \leq \min(S_{4_{m\downarrow -1}})$$

that is

$$\max(S_{3_{m\downarrow -1}}) \leq \min(S_{4_{m\downarrow -1}})$$

however, as the following holds

$$A_{m\downarrow -1}$$

we obtain

$$\max(S_{3_{m\downarrow -1}}) < \min(S_{4_{m\downarrow -1}})$$

Noticing that

$$S_{3_{m\downarrow -1}} = S_{3_{m\downarrow}}$$

$$S_{4_{m\downarrow -1}} = S_{4_{M\downarrow}}$$

we obtain the final result

$$A \vdash M_3 \; \frac{\ell\ ,\ u: \mathbb{F}_1(\mathbb{N})}{} \; \left(\begin{array}{l} A[\ell?/S_1\ ,\ u?/S_2] \\ \\ \max(\ell?) \ < \ \min(u?) \end{array} \right)$$

by applying the "connection" proof rule.

Acknowledgments

Most of the ideas in this paper are based on the works of C.A.R. HOARE, C.B. JONES and E.W. DIJKSTRA.

References

(1) Dijkstra : A Discipline of Programming, by E.W. Dijkstra, Prentice-Hall.

(2) Hoare : An axiomatic Basis for Computer Programming, by C.A.R. Hoare, CACM.

(3) Hoare : Communicating Sequential Programming, by C.A.R. Hoare, CACM, Vol. 21, n° 8, pp. 666-677.

(4) Hoare : A Non-deterministic Model for Communicating Sequential Processes, by C.A.R. Hoare, S.D. Brookes and A.W. Roscoe.

(5) Hoare : A Calculus of Total Correctness for Communicating Processes, by C.A.R. Hoare, Oxford University, PRG Monograph 23.

(6) Hoare : Specifications, Programs and Implementations, by C.A.R. Hoare, Technical Monograph PRG-29, Oxford University, June 1982.

(7) Jones : Software Development : A Rigorous Approach, by C.B. Jones, Prentice-Hall Int. 1980.

(8) Jones : Development Methods for Computer Programs Including a Notion of Interference, by C.B. Jones, Thesis, Oxford University, 1981.

(9) Kahn : Coroutines and Network of Parallel Processes, by G. Kahn and D.B. Mac Queen in IFIP 77 Proc.

(10) Milner : An Algebraic Definition of Simulation between Programs, by R. Milner, Stanford, Memo AIM-142, Report n° CS-205.

(11) Milner : A Calculus of Communication Systems, by R. Milner, Springer Verlag, Lecture Notes in Computer Science, Vol. 92.

(12) Zhou : Partial Correctness of Communicating Sequential Processes, by Zhou Chao Chen and C.A.R. Hoare, Procs of Int. Conf. on Distributed Processes, Paris 1981.

(13) Zhou: Partial Correctness of Communication Protocols, by Zhou Chao Chen and C.A.R. Hoare, NPL Workshop 1981.

(14) Melliar-Smith : From state to temporal logic : specification methods for protocol standards by R.L. Schwartz and P.M. Melliar-Smith, IEE Transactions on communications, Dec. 1982.