An Axiomatic Proof Technique for
Networks of Communicating Processes*

K. M. Chandy
J. Misra

TR-98 May 1979

* Work partially supported by NSF Grant MCS77-09812.

* The reproduction of this report was supported by Hitachi Corporation.

1. Networks of Processes

This paper suggests methods for proving the correctness of networks of processes which communicate exclusively through messages.

1.1 Fundamental definitions

A network is a collection of processes which communicate exclusively through messages. A process is either a network, or a program in the conventional sense, with special primitives for message transmission.

For purposes of exposition we suggest the following primitives based on Hoare [4] for message transmission. We emphasize that these primitives are suggested merely for ease of exposition; our proof techniques are not restricted to these primitives. (For instance, input commands appearing in guards as in Hoare [4] are amenable to this form of analysis.) A process h may have two types of statements for message transmission.

Input statements have the form

x:=?,

and output statements have the form

?:=y,

where x and y are variables local to h. x is called an input variable and y an output variable of h.

Each output (input) variable of a process is bound to at most one input (output) variable of another process.

The binding is external to the processes. Let x be an output variable of process h_1 and let x be bound to y, an input variable of process h_2. Then h_1 will wait at an output statement

?:=x;

until control in h_2 reaches a corresponding input statement

y:=?;

A message transmission will take place some arbitrary but bounded time after h_1 reaches the output statement and h_2 reaches the corresponding input statement. h_1 and h_2 will both complete executions of their input,output statements simultaneously when the message transmission is over; at this point y in h_2 has the value of x in h_1. Of course the value of x in h_1 is unchanged by the message transmission.

Formally a binding is a pair of tuples of the form $((h_1,x),(h_2,y))$ where h_1,h_2 are processes, $h_1 \neq h_2$, x is an output variable of h_1 and y an input variable of h_2.

A network N is (H,B,I,O) where

H is a set of processes $\{h_1,h_2,\ldots,h_m\}$,

B is a set of bindings $\{\ldots((h_i,x),(h_j,y))\ldots\}$ where every input (output) variable of a process in H is bound to at most one output (input) variable of a process in H,

I is a set of tuples $\{\ldots(h_i,x)\ldots\}$ where x is an input variable of h_i which is not bound and,

O is a set of tuples $\{\ldots(h_j,y)\ldots\}$ where y is an output variable of h_j which is not bound.

If (h_i,x) is in I then variable x of h_i is assigned values by a process external to N. Similarly, if (h_j,y) is in O then variable y of h_j assigns values to a variable in a process external to N.

A process $h = (T,I,O)$ is either (1) a network $T = (H,B,I,O)$ or (2) a program T with input variables I and output variables O.

We associate a labelled directed graph $G = (V,E,I,O)$ with a network $N = (H,B,I,O)$ such that there is a one to one correspondence between vertices of G and processes in N and between labelled edges in G and bindings in N: an edge corresponding to $((h_i,x), (h_j,y))$ is directed from the vertex corresponding to h_i to the vertex corresponding to h_j and its tail is labelled x and the head y. In addition to E, we associate an input edge of G directed to the vertex corresponding to h_i and labelled x, at its head, for every $(h_i,x) \in I$; <u>output edges</u> are defined similarly. We will refer to B either as the set of labelled edges or as the set of bindings, and also use other graph notation. No ambiguity should arise.

We associate a sequence of messages S_e with every edge $e = ((h_i,x),(h_j,y))$ which is the sequence of messages sent from h_i by the execution of all statements of the form

?:=x.

Note that S_e is also the sequence of messages received by h_j by executing all statements of the form

y:=?.

We define S_e, where e belongs to I or O in the obvious way. Thus the effect on S_e of the corresponding input or output statement may be defined as follows, where || denotes string concatanation:

$\{S_e = S_e^o\}$?:=x $\{S_e = S_e^o \; || \; x\}$

$\{S_e = S_e^o\}$ y:=? $\{S_e = S_e^o \; || \; y\}$

2. Axioms

2.1 Intuition

Our proofs are hierarchic: to prove properties about a network $N = (H,B,I,O)$ we first prove properties about processes in H and then deduce network properties from process properties, which are stated as assertions over message sequences.

An *assertion over a network* $N = (H,B,I,O)$ is defined to be a boolean function on sequences S_e (where edge e is in B,I or O) and on free variables and constants.

Generally, it is impossible to deduce that an assertion on a network N holds true at all times from a proof of a single process h in H because some other process may falsify the assertion. We must therefore show that no process in H falsifies the assertion. The concept of *not-falsifying* is central to network proofs; hence we coin the phrase "a process h *preserves* an assertion P" to denote that h does not falsify P *at any point in its execution*. Preservation is the concept of invariance for sequential programs generalized to networks. If all processes preserve P and P is true initially, then P is true at all times.

Generally we cannot prove properties about the inputs to a process h from the description of h alone; hence to prove that h preserves an assertion P we have to make assumptions about the inputs to h. A simple (and apparently reasonable) assumption to use in proofs of h is that P is true after every input of h; however this is an unreasonable requirement of the sender process because if P were false prior to the message transmission it may be impossible for the sender to reestablish P. Therefore we will focus on proofs which show for a process h that h preserves P <u>provided</u> that inputs to h preserve P. The axiom (below) shows how to combine such process proofs into a network proof.

2.2 Definitions and axioms

We say that "edge e preserves assertion P," denoted by P[e], if no message transmission along e falsifies P, i.e. P[e] states that if P were tru immediately prior to the transmission of a message along e then P is true immediately after the transmission.

Note: If S_e is not named in P then P[e].

<u>Notation</u>: Let $h_j = (T_j, I_j, O_j)$ be a process in a network N = (H,B,I,O) and let P be an assertion over N. $P|h_j|P$ denotes that given $[\forall e \in I_j, P[e]]$ there exists a proof of h_j that $[\forall e \in I_j \cup O_j, P[e]]$. P|N|P denotes that given $[\forall e \in I, P[e]]$ there exists a proof of N that $[\forall e \in (B \cup O \cup I), P[e]]$.

Note: P|h|P, (P|N|P) mean that if all inputs to h (N) preserve P, then execution of h(N) never falsifies P at any point in its execution.

Note: If h is a program, then P|h|P may be demonstrated by a proof of h showing that P holds at all program points given that (1) P holds initially and (2) the following additional axiom may be used in the proof of h,

$$\{P\} \quad m \quad \{P\}$$

where m is an input statement. Proof techniques as in Hoare [3] and Owicki and Gries [5] may be used to prove h.

If h is a network, the following axiom may be used to demonstrate P|h|P.

<u>Axiom</u>: (Invariance of Hierarchical Composition)

Let N = (H,B,I,O), and let P be an assertion over N.

$[\forall h \in H, P|h|P] \rightarrow P|N|P.$

Note: $P|N|P$ denotes that there exists a proof that every edge in N preserves P given only P[e], for every e in I. The left hand side of the axiom ($\forall h \in H, P|h|P$) denotes that P is preserved by all processes in H (i.e. by every edge in N) provided P[e] holds for all e which are input to processes h in H, i.e. for all e in $B \cup I$. Both the left and right hand sides give preconditions for P to be preserved by all edges in N; the axiom states that if "P preserved by all edges in I <u>and in B</u>" is a precondition, then so is "P preserved by all edges in I."

<u>Intuition</u>: A formal operational model and proofs of the soundness of the axiom with respect to the formal model are found in [2]. Here we only observe that the validity of the axiom follows by applying induction on the chronological sequence of messages transmitted along edges in N.

Note: Another way of viewing $P|h|P$ is as follows:

Consider process h connected to another process $\bar{h}$ called its complement, which represents the rest of the network. $\bar{h}$ feeds input to, and accepts output from h. $P|h|P$ denotes that if $\bar{h}$ guarantees P is true after every input to h then h guarantees that P is true after every output of h (i.e. input to $\bar{h}$). Note that $P|h|P$ is a property of h alone. The axiom states that if the processes in the network preserve P when running in isolation (i.e. with their complements) then the network preserves P.

<u>Definition</u>: Let P and Q be assertions over a network N = (H,B,I,O), and let h be a process in H.

$P|h|Q$ ($P|N|Q$) denotes that h (N) does not falsify either P or Q at any point in its execution given that all inputs to h (N) preserve P. Formally,

$$P|h|Q \equiv \frac{P[e], \text{ all } e \text{ input to } h}{P[e] \text{ and } Q[e], \text{ all } e \text{ incident on } h}$$

$P|N|Q$ is defined similarly.

Note: One way of proving $P|h|Q$ is to show $P|h|P$ and $Q|h|Q$ and that if every input to h preserves P then every input to h also preserves Q.

If h is a program $P|h|Q$ may be demonstrated from a proof of h by showing that P and Q hold at all program points, given that P and Q hold initially, and the axiom {P} m {P} can be used where m is an input statement.

The following theorem which is one way of demonstrating $P|N|Q$, follows directly from axiom 1.

Theorem: Let N = (H,B,I,O) and let P and Q be assertions over N.

$P|h|Q$ for all h in H $\rightarrow$ $P|N|Q$

The technique proposed here leads naturally to hierarchical and modular network design and verification. In order to design a network N with specification $P|N|Q$, where P,Q only name sequences external to or input or output of N, we will first postulate a network structure (H,B,I,O) and a network invariant R where,

1. P is preserved on input implies R is preserved on input and,
2. R implies P and Q and,
3. $R|h|R$ for all h in H -- this implies $R|N|R$.

Hierarchical design proceeds by refining h similarly. There is an obvious similarity with hierarchical, modular design of sequential programs: P corresponds to the input assertion, P and Q to the output assertion and R to the loop invariant.

The proof technique presented here is inadequate for proving termination or absence of deadlock; such techniques are found in [1].

3. An Example

3.1 A network to compute the factorial of a sequence of numbers

We design a network which receives a sequence of nonnegative integers along an input edge and sends a sequence of numbers which are factorials of corresponding numbers in the input sequence along an output edge. We assume that every input number is less than n. We employ three kinds of processes.

1. Buffer process: This process has one input and one output edge. It maintains a queue of bounded or unbounded size -- the maximum size of the queue is unimportant for proofs. The incoming messages are appended to the rear of the queue. For output, messages from the front of the queue are removed and sent. Since this process is generally well understood, we do not elaborate on it.
2. Input process: It has one input edge with associated variable x and two output edges with associated variables r and y.

   Input process:

   ```
   loop
     x:=?;
     if x≠0 then  y:=x-1; ?:=y endif;
     r:=x; ?:=r
   endloop
   ```
3. Output process: It has two input edges with associated variables u, v and one output edge with associated variable w.

   Output process:

   ```
   loop
     u:=?;
     if u=0 then w:=1 else v:=?; w:=u*v endif;
     ?:=w
   endloop
   ```

A combined process, CP, is a network constructed from 3 buffer processes, one input process and one output process, as shown in Figure 1: ba,bd,bu stand for buffer across, buffer down and buffer up.

The network N consists of n combined processes $CP_1,\ldots,CP_n$. For simplicity, an edge incident on a buffer within the combined process has the same labels at head and tail.

Any combined process CP_i, $i>1$, receives its inputs through variable x_i, delegates responsibility to CP_{i+1} by an output along t_i to compute the factorial of the next lower number, receives the response from CP_{i+1} via z_i and produces its own output along w_i. The operations are however asynchronous in that many inputs may be read before any output is produced. CP_n is a process that receives zeroes and outputs 1's.

Notation: In the following, S,S_1,S_2 denote sequences of integers.

$0\le S<i$ :: every element of S is less than i and greater than or equal to 0.

$S_1 \sqsubseteq S_2$ :: S_1 is an initial segment of S_2; possibly $S_1 = S_2$.

red(S) :: the sequence obtained from S by removing all zeroes and reducing the remaining elements by 1.

S! :: the sequence obtained from S by taking factorial of each element.

Let Sx_i denote the sequence associated with variable x of CP_i; Similarly, St_i, Sz_i, Sw_i.

Note: $St_i = Sx_{i+1}$, $i<n$.

$Sw_i = Sz_{i-1}$, $i>1$.

Initial condition on N: all sequences are initially null.

3.2 Proof of the network

Given the input specification,

$$0 \leq Sx_1 < n$$

it is required to prove the output specification,

$$Sw_1 \sqsubseteq Sx_1!\ .$$

The methods given in this paper are inadequate for proving statements of the form, "N will not deadlock" or "the output corresponding to every input will eventually be produced"; see [1] for proof techniques for such properties. In the following proof, we will often make use of the following observations to simplify the proofs. These observations follow directly from definitions.

Observations:

1. $[P \rightarrow Q$ <u>and</u> $P|h|Q] \rightarrow [Q|h|Q]$
2. true$|h|$P, if P is an assertion over sequences not incident on h.
3. $P_1|h|Q_1$ <u>and</u> $P_2|h|Q_2 \rightarrow (P_1$ <u>and</u> $P_2)|h|(Q_1$ <u>and</u> $Q_2)$.
4. $(S_1 \sqsubseteq S_2)|h|(S_1 \sqsubseteq S_2)$, where S_1 is either an input to h or is not incident on h.

In order to use axiom 1, we postulate the following invariant.

P:: $0 \leq Sx_1 < n$ <u>and</u> $Sx_{i+1} \sqsubseteq red(Sx_i)$, $1 \leq i < n$ <u>and</u> $Sw_i \sqsubseteq Sx_i!$, $1 \leq i \leq n$.

Note:

1. If the input to N preserves the input specification, it preserves P.
2. P implies the output specification.

We next show $P|CP_i|P$, for every $1 \leq i \leq n$, from which $P|N|P$ may be deduced using axiom 1. Note that the internal structure of CP_i is of no consequence at this stage since its externally observable behavior is captured by $P|CP_i|P$.

3.3 Proof of CP_i

From P, we can deduce the following.

Q_i, $1 \leq i < n$ :: $0 \leq Sx_i$ and $St_i \sqsubseteq red(Sx_i)$ and $Sz_i \sqsubseteq St_i!$ and $Sw_i \sqsubseteq Sx_i!$

Q_n :: $0 \leq Sx_n < 1$ and $Sw_n \sqsubseteq Sx_n!$

Using observations 1, 2, 4 of 4.2, is follows that it is sufficient to demonstrate $Q_i|CP_i|Q_i$, $1 \leq i \leq n$ in order to prove $P|CP_i|P$, $1 \leq i \leq n$. The proof of $Q_n|CP_n|Q_n$ is straightforward and hence is omitted here.

Notation: We drop the subscript i, in the following discussion.

In order to show $Q|CP|Q$, we again have to postulate an invariant R which relates the various internal sequences of CP.

R :: $0 \leq Sx$ and $Sz \sqsubseteq St!$ and $Sr \sqsubseteq Sx$ and $Sy \sqsubseteq red(Sx)$ and $St \sqsubseteq Sy$ and $Su \sqsubseteq Sr$ and $Sv \sqsubseteq Sz$ and $Sw \sqsubseteq Su!$.

Note: 1. If Q is preserved on input to CP, R is preserved on input to CP. Also $R \rightarrow Q$.

Hence $Q|CP|Q$ follows from $R|CP|R$ which follows from $R|h|R$ for every process h in CP.

It is easy to prove the following facts from proofs of individual programs; from observations in 4.2 it then follows that each process in CP preserves R.

$(true|input|Sr \sqsubseteq Sx$ and $Sy \sqsubseteq red(Sx))$, $(true|ba|Su \sqsubseteq Sr)$,

$(true|bd|St \sqsubseteq Sy)$ and $(true|bu|Sv \sqsubseteq Sz)$. We show, $R|output|R$ formally in the next section.

3.4 Proof of output

Using the observations of (4.2), it is sufficient to show,

(Su $\sqsubseteq$ Sx and $0 \le$ Su and Sv $\sqsubseteq$ red(Sx)!) |output| Sw $\sqsubseteq$ Su!

Let, R_1 :: Su $\sqsubseteq$ Sx and $0 \le$ Su and Sv $\sqsubseteq$ red(Sx)!.

R_2 :: Sw $\sqsubseteq$ Su!.

R_3 :: Sw $\sqsubseteq$ Su! and u = tail(Su). {tail is the last element of a nonnull string}

R_4 :: |red(Su)| = |Sv|. {|Sv| denotes the length of Sv}

R_5 :: |red(Su)| = |Sv| + 1.

It is required to show that R_1|output|R_2. Note that $R_3 \rightarrow R_2$.

Annotated proof:

{R_1 and R_2 and R_4} [Note: R_4 follows from initial conditions]

loop {R_1 and R_2 and R_4}

u:=?;

{R_1 and R_3 and (R_4 or R_5)} [R_1 is preserved by input]

if u=0 then {R_1 and R_3 and R_4 and u=0}

w:=1 {R_1 and R_3 and R_4 and w=u!}

else {R_1 and R_3 and R_5 and u≠0}

v:=?;

{R_1 and R_3 and R_4 and u≠0 and v = tail(Sv)} [R_1 is preserved by input]

{v=(u-1)!}

w:=u*v

{R_1 and R_3 and R_4, w=u!}

endif;

{R_1 and R_3 and R_4, w=u!}

?:=w

{R_1 and R_2 and R_4}

endloop

4. Acknowledgement

The impetus for this work came from the simple, yet powerful language proposed by Hoare [4]. The concept of non-interference due to Owicki and Gries [5] is implicit in this work. We are grateful to Professor D. Gries for discussions.

References

1. K.M. Chandy and J. Misra, Deadlock absence proofs for networks of communicating processes, Computer Sciences Technical Report, University of Texas at Austin, 1979.

2. K.M. Chandy and J. Misra, A formal model of networks of communicating processes, Computer Sciences Technical Report, University of Texas at Austin, 1979.

3. C.A.R. Hoare, An axiomatic basis for computer programming, CACM October 1969.

4. C.A.R. Hoare, Communicating sequential processes, CACM, August 1978.

5. S.S. Owicki and D. Gries, An axiomatic proof technique for parallel programs, Acta Informatica, June 1976.

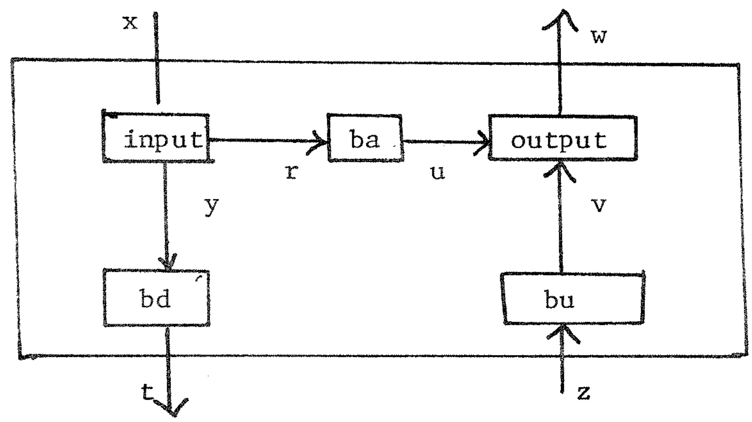

Figure 1: Internal Structure of a CP

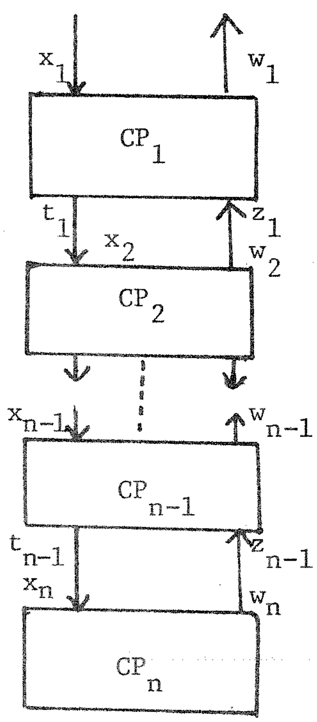

Figure 2: The Structure of the Network