PROVING SAFETY AND LIVENESS OF COMMUNICATING PROCESSES WITH EXAMPLES

J. Misra, K. M. Chandy and Todd Smith
Computer Sciences Department, University of Texas, Austin 78712

ABSTRACT

A method is proposed for reasoning about safety and liveness properties of message passing networks. The method is hierarchical and is based upon combining the specifications of component processes to obtain the specification of a network. The inference rules for safety properties use induction on the number of messages transmitted; liveness proofs use techniques similar to termination proofs in sequential programs. We illustrate the method with two examples: concatenations of buffers to construct larger buffers and a special case of Stenning protocol for message communication over noisy channels.

Key Words and Phrases: communicating processes, message-passing systems, proofs of process networks, safety, liveness.

CR-Categories: C.2.2, C.2.4, D.1.3, F.3.1, F.3.2

1. INTRODUCTION

This paper presents a method for reasoning about safety and liveness properties of networks of processes in which communication is through messages only. The key features of this method are:

(1) Modular Specification: We present a scheme for specifying processes in a modular fashion. The specification relies exclusively on a process's interaction with its environment and is independent of process implementation.

(2) Hierarchy: We present inference rules by which a specification for a network is derived from specifications of component processes. Thus the proof of a network is not concerned with implementations of component processes.

(3) Compatibility With Sequential Programming Proof Techniques: We have extended well known sequential programming proof constructs such as pre-condition, post-condition and the ideas of termination proof to distributed systems. Those familiar with the Floyd-Hoare proof technique for sequential programming should find our method to be straightforward.

The organization of this paper is as follows. We describe a model of computation in section 2. We discuss the proof technique in section 3. Section 4 contains the example of concatenations of buffers to construct larger buffers. We prove a special case of the Stenning protocol for message communication over noisy channels, in section 5.

Apt, DeRoever, Francez [1] and Levin, Gries [4] propose alternate proof techniques. Both these works depend upon analysis of code fragments of two communicating processes to ensure that only desirable communications take place. Pioneering work using temporal logic in proving liveness properties is due to Owicki and Lamport [7]. Hailpern [2] proposes proof techniques using temporal logic for general concurrent programs which include both shared memory as well as message passing systems. A proof of Stenning protocol appears in Hailpern, Owicki [3].

2. MODEL OF A NETWORK

Our reasoning technique is applicable to a variety of network models and protocols. However we confine our discussion to an extremely simple network model. In this section our goal is to define a model,

*This work was supported by the Air Force Office of Scientific Research under grant AFOSR 81-0205 and the University Research Institute at the University of Texas.

not a programming language; hence syntactic issues will be treated informally.

A process is either a sequential process or a network of processes. A sequential process is a sequential program with commands for message transmission. It may have input ports through which messages are received and output ports through which messages are sent. An output port of one process may be connected to the input port of another process by a directed channel. A port is connected to one chanel and a channel is always connected to one input port and one output port. All connections of ports and channels are static.

A sequential process h can execute a send command which has the form:

send m via p

where m is a local variable and p is an output port of h. Process h continues execution of its program following execution of the send command. Execution of this command results in a message m being sent along the channel to which output port p is connected. Messages sent along a channel arrive at their destination in the order sent and after an arbitrary but finite delay.

A sequential process h can execute a receive command which has the form:

receive m via p

where m is a local variable and p an input port of h. Execution of this command results in the first message (if any) which has arrived at the input port p being removed, and its value assigned to m. If there is no such message, h waits until a message arrives at the port. A process can also test whether there is a message at an input port; for instance it may execute a statement of the form: if there is a message at input port p then s1 else s2.

A network is also a process with input and output ports. A network consists of one or more component processes whose ports are connected by channels. Any port of a component process, which is not connected by a channel to another component process port, is a port of the network.

Example: A Sequential Process: Merge2

This process receives monotone increasing sequences along its two input ports in[1] and in[2] and produces the merged monotone increasing output sequence along its single output port out. Its sequential program is given below.

```
Process Merge2 (input port in[1], in[2];
                output port out)
  receive $x_1$ via in[1];
  receive $x_2$ via in[2];
  while true do {loop forever}
  if $x_1 < x_2$ then
     begin send $x_1$ via out;
           receive $x_1$ via in[1]
     end
     else if $x_2 < x_1$ then
     begin send $x_2$ via out;
           receive $x_2$ via in[2]
     end
     else {$x_1 = x_2$}
     begin send $x_1$ via out;
           receive $x_1$ via in[1];
           receive $x_2$ via in[2]
     end
```

Example: A Network: merge3

merge3 receives monotone increasing sequences along 3 input ports in[1], in[2] and in[3]; it outputs the monotone increasing merged sequence along its single output port out. merge3 can be implemented as a network of two component merge2 processes.

3. PROOFS OF PROCESSES

We use some ideas from sequential program proofs in proofs of message-passing systems. In an annotated proof of a sequential program, each statement s has a precondition pre(s) and a postcondition post(s). The proof shows that if assertion pre(s) holds prior to execution of s, post(s) holds following execution of s assuming execution of s terminates. We shall use the precondition/postcondition concept for describing process safety properties. Proofs of liveness (or termination) in sequential programs are based on demonstrating the existence of a metric such that the execution of each statement causes the metric to decrease in value. We will use a similar technique in process proofs. However, processes can wait indefinitely for messages, something that conventional sequential programs do not do; to handle this we introduce a new concept called activity which is the condition under which a process will definitely send or receive a message. Other liveness properties are derived from the basic property of activity and from safety.

3.1 Trace

A trace of a process h is a sequence of tuples $<(port_1,v_1), (port_2,v_2),\ldots,(port_n,v_n)>$, where in some computation the ith message sent or received by h is through $port_i$ and has value v_i. If $port_i$ is an output (input) port then h sent (received) v_i through $port_i$. Thus the trace is a chronological sequence of all interactions that a process has with its environment in a particular computation.

An assertion r holds at all points of a trace T: $<(port_1,v_1),\ldots(port_n,v_n)\ldots>$, if r holds for all initial prefix traces

$<(port_1,v_1)\ldots(port_i,v_i)>$, $i \geq 0$, of T. Note that r must then hold for the null trace, i.e. the trace which has no element. The trace T' : <T;(port,v)> which has T as the initial prefix trace and one more element, is called an extension of T.

The sequence of messages transmitted or received by a process h via $port_i$ will be denoted by $h.port_i$ (or $port_i$ when we are discussing process h). Let Z, Z1 and Z2 be sequences of messages. Then |Z| is the length of Z and Z1 $\underline{\alpha}$ Z2 denotes that Z1 is an initial subsequence of Z2. Note that Z $\underline{\alpha}$ Z, for all Z.

3.2 Specification of a Process

We use three propositions r, s and q to specify a process h, and the specification will be denoted by $r|\frac{h}{q}|s$; r is called the precondition, s the postcondition and q the activity condition. r and s are assertions on traces of h while q is an assertion on the trace of h and the empty/non-empty status of the channels connected to its ports.

$r|\frac{h}{q}|s$ means that

(1) s holds for the null trace,

(2) if r holds at all points of a trace T of h then s holds at all points of any trace T' of h, where T' is an extension of T,

(3) if r holds at all points of a trace T of h and q holds for T then there exists a trace T' of h which is an extension of T.

The second condition does not state that the trace T will be extended to T'; it merely states that if the trace is extended then s holds for the extended trace. The third condition is a sufficient condition under which the trace of h will definitely be extended. Since all process speeds are assumed to be non-zero and finite, the phrase "trace of h definitely will be extended" means that no process can have its trace extended indefinitely without the trace of h being extended.

The proof $r|\frac{h}{q}|s$ for a sequential process, requires one sequential program proof. A proof method appears in [5], when q is absent; it has been applied in a number of examples in [6]. We have not included the proof method in this paper. In next section, we show how the specifications of a network can be proven from specifications of component processes.

3.3 Theorem of Hierarchy

The theorem of hierarchy gives the conditions under which we can deduce $R|\frac{H}{Q}|S$, for a network H, given $r_i|\frac{h_i}{q_i}|s_i$, for all processes h_i in H. We first present an axiom - the communication Axiom C - which captures the essence of the proposed communication protocol. The only assumption made about the communication protocol in the theorem of hierarchy is the communication axiom C; therefore changes in the protocol only affect C and not the theorem of hierarchy directly.

We give C for the model of section 2. If there is a channel linking the output port p_1 of process h_1 with input port p_2 of h_2 then the sequence of messages received by h_2 through p_2 must be an initial subsequence of the messages sent by h_1 through p_1. Formally,

$$h_2.p_2 \;\underline{\alpha}\; h_1.p$$

Let the port P of the network H be the same as the port p of the component process h; then since renaming of a port does not alter the message sequence through it,

$$\underline{H.P} = \underline{h.p}$$

Combining these we have the communication axiom,

C :: If there is a channel linking output port p_1 of h_1 with input port p_2 of h_2, then $h_2.p_2 \;\underline{\alpha}\; h_1.p_1$. If port P of H is the same as port p of h, then H.P = h.p .

Given $r_i|\frac{h_i}{q_i}|s_i$, for all processes h_i, i=1,2,... in a network H, we give conditions under which $R|\frac{H}{Q}|S$ holds. Let,

$$s = C \;\frac{\text{and}}{i}\; s_i,\quad r = \frac{\text{and}}{i}\; r_i,\quad q = \frac{\text{or}}{i}\; q_i$$

3.3.1 Statement of the Theorem of Hierarchy

If, (i) $r_i|\frac{h_i}{q_i}|s_i$, i=1,2,...

(ii) s and R ⇒ r, {harmony}

(iii) s ⇒ S, {abstraction}

(iv) s and Q ⇒ q {progress}

(v) s and Q ⇒ ($\sum_i$ trace length of h_i) $\leq$ F (trace length of H), for some function F {boundedness}

then $R|\frac{H}{Q}|S$.

3.3.2 Explanation

Conditions (ii) and (iii) deal with safety and (iv) and (v) with liveness. Condition (ii), called the harmony condition says that all preconditions assumed by the component processes are implied by the precondition of the network H and the postconditions of the component processes. Condition (iii), called the abstraction condition, says that the network's postcondition must be derivable from the postconditions of component processes. Condition (iv), called the progress condition, states that the network can be active only if some component process is active. Condition (v), called the boundedness conditions, states that processes cannot send or receive messages indefinitely without the network communicating as well[+]. The essence of the safety rules is: each time the trace of some process h_i is extended, process h_i guarantees s_i (and hence s is maintained) and harmony guarantees r for the extended trace.

4. AN EXAMPLE: CONCATENATION OF BOUNDED BUFFERS

4.1 Operational Description of a Bounded Buffer

A bounded buffer process of size b is shown schematically in Figure 1. This process can hold at most b, b>0, items of data. It is interposed between a producer and a consumer. The process sends requests for data via ro to the producer if it has room for data (not all buffer spaces are full) and if it has no outstanding request to the producer. It receives data from the producer through di. It receives requests from the consumer for data via ri if it has some data (the buffer spaces are not all empty) and if it has already serviced all consumer requests; it subsequently sends data through do in such a case. The goal of this example is to show formally that concatenation of N buffers of sizes $b_1, b_2, \ldots, b_N$ is equivalent to a single buffer of size $\sum_{i=1}^{N} b_i$.

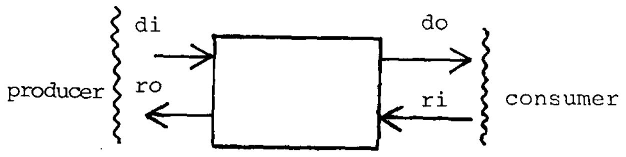

Figure 1: Bounded buffer of size b.

4.2 Specification of Bounded Buffer of Size b

The buffer process of size b can be specified by the assertions r, s and q. We present each of the assertions in a formal notation and then explain in English. In the following "a is empty," where a is a port of some process h, denotes that the channel connected to a is empty.

r :: true

s :: $|\underline{do}| \le |\underline{ri}| \le |\underline{do}| + 1$ (s1);

{The data to and requests from the consumer alternate}

$|\underline{di}| \le |\underline{ro}| \le |\underline{di}| + 1$ (s2);

{The requests to and data from the producer alternate}

$|\underline{ri}| \le |\underline{di}|$ (s3)

{no buffer underflow, i.e. no request from the consumer is accepted unless there is data}

$|\underline{ro}| \le |\underline{do}| + b$ (s4);

{no buffer overflow}

$\underline{do}\ \alpha\ \underline{di}$ (s5);

{buffer transmits the received data in sequence}

q :: $(|\underline{do}| < |\underline{di}|$ and

$(|\underline{do}| < |\underline{ri}|$ or ri is not empty))

{buffer is not empty and all requests sent by the consumer have not been processed; data will be sent to consumer}

or $(|\underline{di}| < |\underline{do}| + b$ and

$(|\underline{ro}| = |\underline{di}|$ or di is not empty))

{buffer is not full and producer has responded to all requests for data; request will be sent to producer}

The problem is to show that concatenation of any N buffers of sizes $b_1, b_2 \ldots b_N$ has the same specification as a buffer of size $\sum_{i=1}^{N} b_i$. We show that the concatenation of two buffers of sizes b_1, b_2 has the same specification as a single buffer of size $b_1 + b_2$. The proof follows for N > 2 in a straightforward manner.

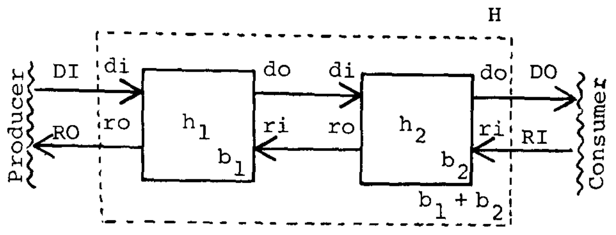

Figure 2. Concatenation of two buffers of sizes b_1, b_2.

4.3 Proof of Bounded Buffer Concatenation

4.3.1 Harmony

Trivial, since r is true.

[+]Hoare terms this "absence of infinite chatter."

4.3.2 Abstraction

(s1) $|\underline{DO}| \le |\underline{RI}| \le |\underline{DO}| + 1$:

follows from,

$|\underline{h_2.do}| \le |\underline{h_2.ri}| \le |\underline{h_2.do}| + 1$ (s_1 for h_2)

and the communication axiom C.

(s2) Proof similar to (s1).

(s3) $|\underline{RI}| \le |\underline{DI}|$:

$|\underline{RI}| = |\underline{h_2.ri}| \le |\underline{h_2.di}|$ (C, s3 for h2)

$|\underline{h_2.di}| \le |\underline{h_1.do}|$ (C)

$|\underline{h_1.do}| \le |\underline{h_1.di}| = |\underline{DI}|$

(s1,s3 for h_1,C)

(s4) $|\underline{RO}| \le |\underline{DO}| + b_1 + b_2$:

$|\underline{RO}| = |\underline{h_1.ro}| \le |\underline{h_1.ri}| + b_1$

(C and s4, s1 for h_1)

$|\underline{h_1.ri}| \le |\underline{h_2.ro}|$ (C)

$|\underline{h_2.ro}| \le |\underline{h_2.do}| + b_2$

$= |\underline{DO}| + b_2$

(s4 for h_2, C)

(s5) Similar to proof of (s1).

4.3.3 Progress

We will show that if h_1 is not active (q_1 is false), h_2 is not active (q_2 is false) and s holds then H is not active (Q is false). The negation of q_1 can be written as a conjunction of two propositions, (i) and (ii):

(i) the buffer in h_1 is empty ($|\underline{h_1.do}| = |\underline{h_1.di}|$) or h_1 is waiting for requests from its consumer h_2 ($|\underline{h_1.do}| = |\underline{h_1.ri}|$ and channel $h_1.ri$ is empty), and

(ii) the buffer in h_1 is full ($|\underline{h_1.di}| = |\underline{h_1.do}| + b_1$) or h_1 is waiting for response from the producer $|\underline{h_1.di}| < |\underline{h_1.ro}|$ and channel $h_1.di$ is empty).

A similar set of propositions correspond to $\sim q_2$ and $\sim Q$.

It is straightforward to conclude from $\sim q_1$ and $\sim q_2$ that all buffers in h_1 are empty or all buffers in h_2 are full. We now show that the corresponding conditions (i) and (ii) hold for H in this case. Condition (i) for H is: all buffers in H are empty or H is waiting for requests from the consumer. If all buffers in H are not empty, then from the observation in the first line of this paragraph, all buffers in h_2 cannot be empty, and therefore from $\sim q_2$, h_2 is waiting for requests from the consumer. Condition (ii) can be proven symmetrically.

4.3.4 Boundedness

We can show using (s1),(s2),(s3),(s4) for h_1 and h_2, that the trace lengths of h_1 and h_2 is no more than twice the trace length of H.

5. STENNING PROTOCOL WITH WINDOW SIZE 1 [3,6]

Stenning protocol can be used to send messages from a producer to a consumer over noisy channels. We consider a special case of the Stenning protocol in this paper - the transmitter sends a new message only after it receives an acknowledgement from the receiver for the previous message; if it receives no acknowledgement within a specified time period, it retransmits the message. The full Stenning protocol allows the transmitter to send more than one message without having received acknowledgements. Conceptually, the proof of full Stenning protocol is only slightly more difficult than the one presented here; a proof of safety for the general case using methods of this paper appears in [6].

This example illustrates the use of the theorem of hierarchy on a problem in which (1) the communication axiom C described earlier is no longer valid, since a channel can lose, duplicate and permute messages and (2) time-out is an essential feature of the protocol.

5.1 Description of Stenning Protocol

The communication network is shown within dotted lines in Figure 3. For simplicity of description, each channel has a name which is identical to the port names at both ends.

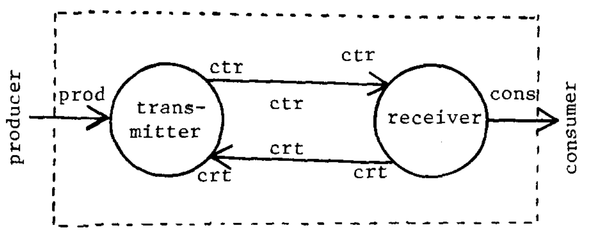

Figure 3. A network to implement Stenning Protocol.

The channels linking the transmitter and receiver can lose, duplicate or permute messages sent along them. The transmitter receives a message from the producer and transmits it along channel ctr after appending an idenfifying sequence number. It continues to retransmit the message after some time unless it receives an acknowledgement (ack) for that message along crt. Upon receiving an ack for the last message sent, transmitter receives the next data item from the producer. The receiver, upon receiving a data item along ctr, checks to see if it is the last data item it has transmitted to the consumer - in this case it sends an ack along crt - or if it is the next item to be transmitted to the consumer (this is determined by the sequence number appended to every data item) - in this case, it sends the data item to the consumer and an ack along crt.

If a channel loses all messages or never delivers some particular message even if it is transmitted many many times, we cannot guarantee eventual delivery of a message. Therefore we postulate the following communication axioms for every channel a, {a.read(v)/a.sent(v) denotes the number of times message v has been received/sent along channel a},

(C1) $a.read(v) > 0 \Rightarrow a.sent(v) > 0$, for all v;
{every message received must have been sent}

(C2) there exist monotone nondecreasing functions f_1, f_2 such that

$$f_1(a.read(v)) \le a.sent(v) \le f_2(a.read(v)) \text{ for all v.}$$

{every message sent often enough will be received often enough and no message is duplicated infinitely often. This means in particular that a sender process cannot be infinitely faster than the receiver process}

<u>Notation</u>: To simplify notation, we assume that every message is a tuple consisting of a sequence number (a positive integer) and a data item. Thus the messages sent by the producer to the transmitter, by the transmitter to the receiver, by the receiver to the consumer and the acks sent by the receiver to the transmitter are all tuples of the same form.

5.2 <u>Specifications of Component Processes</u>

5.2.1 <u>Specification of the transmitter</u>

Let $<(c_1,v_1)\ldots(c_i,v_i)\ldots(c_L,v_L)>$ be the trace.

r :: jth item received along port prod, has sequence number j

s :: (1) c_i = prod, c_j = prod, $i < j \Rightarrow \exists k$, $i<k<j$, $(c_k,v_k) = (crt,v_i)$

{A message is received along prod only if ack to all earlier messages have been received}

(2) c_j = ctr $\Rightarrow \exists i$, $i<j$, (c_i,v_i) = (prod,v_j) <u>and</u> k, $i<k<j$, $(c_k,v_k) \neq$ (crt,v_j)

{A message is transmitted along ctr only if it has been received along prod and no ack for it has been received}

q :: $\forall i(c_i,v_i) \neq$ (crt,<u>prod</u>(N)),

{The trace will definitely be extended if an ack for the N-th message has not been received}

<u>Note</u>: It follows that the last message received from the producer will be retransmitted indefinitely often unless an ack for it is received. The trace will be extended as long as ack for the N-th message has not been received.

5.2.2 <u>Specification of the receiver</u>

{$<(c_1,v_1),\ldots,(c_i,v_i),\ldots,(c_L,v_L)>$ denotes the trace.}

r :: <u>true</u> {no assumptions made about the input data}

s :: (1) c_j = cons $\Rightarrow$ (c_{j-1},v_{j-1}) = (ctr,v_j)

{Only the last message received along ctr can be sent along cons}

(2) c_j = crt $\Rightarrow$ [c_{j-1} = cons <u>or</u> c_{j-1} = ctr] <u>and</u> [v_j = last(cons) = last(ctr)],

where last(Z) denotes the last message sent or received along port Z.

{An ack is sent only if the last(ctr) and last(cons) match. Furthermore at most one ack is sent after receiving a message.}

(3) The jth message sent along cons has sequence number j.

q :: c_L = ctr <u>and</u> [v_L = last(cons) <u>or</u> v_L = last(cons) $\oplus$ 1],

where last(cons) $\oplus$ 1 denotes a message with sequence number 1 higher than last(cons).

{The receiver will extend its trace if it receives along ctr, last(cons) or last(cons) $\oplus$ 1; in the former case, it sends an ack along crt and in the latter case, it also sends a message to the consumer.}

5.2.3 Desired network proof

{$<(c_1,v_1)..(c_L,v_L)>$ is the network's trace.}

R :: The jth message received along prod has sequence number j.

S :: c_{i+1} = prod $\Rightarrow$ c_i = cons

c_{i+1} = cons $\Rightarrow$ (c_i, v_i) = (prod, v_{i+1})

{Messages from the producer and to the consumer alternate.}

Q :: $|\underline{\text{cons}}| < N$

{Network's trace will be extended, i.e. a message will be received from the producer or sent to the consumer, if all N messages have not been sent to the consumer.}

5.3 Proof of the Stenning Communication Protocol

5.3.1 Harmony

{s <u>and</u> R $\Rightarrow$ r}

Trivial, since R $\Rightarrow$ $r_{\text{transmitter}}$ and r_{receiver} = <u>true</u>.

5.3.2 Abstraction

Lemma 1: Given s, every message sent along cons must have been received along prod.

Proof: Every message sent along cons must have been received by the receiver along ctr {from s_{receiver}}. Every message received along ctr must have been sent along ctr {from channel axiom C1}. Every message sent along ctr must have been received along prod {from $s_{\text{transmitter}}$}.

The lemma follows.

Lemma 2: Given s, the transmitter receives an ack v only if v has been sent along cons.

Proof: Follows from s_{receiver} and chan- channel axiom C1, applied to channel crt.

Proof of abstraction hypothesis: From lemma 1 and the fact that the jth message sent along cons has sequence number j, it follows that the sequence of messages sent along cons is the same as the sequence received along prod. Therefore it remains to show that the network's operation alternates between receiving from prod and sending to cons. If c_i = prod and c_j = prod, i < j, in the network trace, then from $s_{\text{transmitter}}$, the transmitter must have received v_i along crt prior to receiving v_j along prod. From lemma 2, there exists k, i<k<j such that (c_k, v_k) = (cons, v_i). It is straightforward to show that between every two message sends along cons, there must be a message receipt along prod.

5.3.3. Progress

{s and Q $\Rightarrow$ q}

Q says that $|\underline{\text{cons}}| < N$. From s, jth data item sent along cons has sequence number j. Therefore, no data item with sequence number N has been sent along cons, if Q holds. From lemma 2, transmitter could not have received an ack for <u>prod</u>(N). Therefore $q_{\text{transmitter}}$ holds.

5.3.4 Boundedness

{s and Q $\Rightarrow$ ($\sum_i$ trace length of h_i) $\leq$

F(trace length of H),

for some function F}

We show boundedness from s alone. We will in fact show a bound on the number of times that any message v is transmitted along the channels crt and ctr. In any computation of the network, consider the point at which the transmitter last sent message v along ctr. From $s_{\text{transmitter}}$, transmitter has received 0 acks for v along crt at that point. From channel axiom C2, receiver has sent no more than $f_2(0)$ acks for v. Since v is the last message being sent by the transmitter, from s_{receiver}, the receiver sends an ack every time it receives v and hence the receiver could not have received v more than $f_2(0)$ times. Therefore from C2 transmitter could have sent v at most $f_2(f_2(0))$ times. A message is received a bounded number of times if it is sent a bounded number of times (from C2). The result follows.

6. CONCLUSION

The goal of this paper has been to extend the ideas of sequential program proving to proofs of message communicating systems. Ideas of pre- and post conditions and boundedness seem to have natural analogs in message passing systems. It is hoped that the full power of sequential program proving methods can be applied to these systems; to do so we need to develop a convenient notation for descriptions of traces and operations on them.

7. REFERENCES

[1] Apt, K.R., N. Francez and W.P. de Roever, "A Proof System for Communicating Sequential Processes," <u>TOPLAS</u>, Vol. 2, July 1980.

[2] Hailpern, B.T., "Verifying Concurrent Processes Using Temporal Logic," Ph.D Thesis, Computer Systems Lab, Stanford University, August 1980.

[3] Hailpern, B. and S. Owicki, "Modular Verification of Computer Communication Protocols," Research Report RC8726, IBM Watson Research Center, March 1981.

[4] Levin, G.M. and D. Gries, "A Proof System for Communicating Sequential Processes," Acta Informatica 15, Springer-Verlag, 1981.

[5] Misra, J. and K.M. Chandy, "Proofs of Networks of Processes," IEEE-TSE, Vol. SE-7, No. 4, July 1981.

[6] Ossefort, M., "A Unified Approach to Formal Verification of Network Safety Properties," Ph.D thesis, Computer Sciences Department, University of Texas, 1982.

[7] Owicki, S. and L. Lamport, "Proving Liveness Properties of Concurrent Programs," Computer Systems Laboratory, Stanford University, 1980.