Derivation of a Distributed Algorithm for Finding Paths in Directed Networks*

Robert McCurley
Fred B. Schneider

TR 83-586
December 1983

Department of Computer Science
Cornell University
Ithaca, New York 14853

*This work is supported in part by NSF Grant MCS-81030605. Schneider is also supported by an IBM Faculty Development Award.

Derivation of a Distributed Algorithm for Finding Paths in Directed Networks*

Robert McCurley
Fred B. Schneider

TR 83-586
December 1983

Department of Computer Science
Cornell University
Ithaca, New York 14853

*This work is supported in part by NSF Grant MCS-81030605. Schneider is also supported by an IBM Faculty Development Award.

Derivation of a Distributed Algorithm for Finding Paths in Directed Networks*

Robert McCurley
Fred B. Schneider

Department of Computer Science
Cornell University
Ithaca, New York 14853

December 14, 1983

ABSTRACT

A distributed algorithm is developed that can be used to compute the topology of a network, given that each site starts with information about sites it is adjacent to, the network is strongly connected, and communication channels are unidirectional. The program is derived and proved correct using assertional reasoning.

*This work is supported in part by NSF Grant MCS-8103605. Schneider is also supported by an IBM Faculty Development Award.

1. Introduction

Computer-communication networks implemented by radio channels present some interesting problems. Due to local terrain and antenna placement, sites might be able to send messages directly to sites from which they cannot receive messages directly. We call such a network a *directed network*. If there is a path between every pair of sites in a directed network, then every site can communicate with all other sites. To do this, sites must be aware of the topology of the network so that messages can be forwarded over appropriate routes.

An algorithm to compute and disseminate the topology of a directed network is derived in this paper. The algorithm is actually more general, since it can be used to disseminate the union of the information known to each site to all sites in the network. While the algorithm is itself interesting, our major concern in this paper is with its derivation. Techniques usually associated with developing sequential programs [Dijkstra 76][Gries 81] are used in developing our distributed program.

Section 2 gives some definitions pertaining to directed networks. In section 3, the algorithm is derived and proved correct. Section 4 contains some conclusions.

2. Directed Networks

A directed network is modeled by a set of sites P and a set of *links* $L \subseteq P \times P$. L models the communication structure of the network; link $(i,j) \in L$ iff site j can receive directly messages sent by site i. Communication between all pairs of sites is possible only if the graph (P, L) is strongly connected. In the following, we consider only strongly connected directed networks.

Each link in a directed network is assumed to implement a *virtual circuit* [Tannenbaum 81] with the following properties:

VC1: Every message sent is delivered.

VC2: Messages on a virtual circuit are delivered in the order sent.

A virtual circuit behaves like a FIFO queue—messages received are removed from the front of the queue and messages sent are appended to the rear. Implementation of a communications service satisfying VC1 and VC2 is presumed to be done by low-level software and will not be considered here. (An acknowledgment-retry protocol cannot be used because a channel for the acknowledgment message may not exist, but other techniques, such as repeated transmission of messages, or use of error-correcting codes, could be used.)

In a directed network, i is a *predecessor* of j and j a *successor* of i if $(i,j) \in L$. The set of predecessors of a site i is denoted[1] by $pred_i$; the set of successors by $succ_i$.

For any site i we assume the

Local Topology Assumption: $pred_i$ is the only information about L that is initially available to site i.

This assumption reflects the fact that, in a network implemented by radio channels, a site initially knows about only those sites from which it can directly receive.

Communication between a site and its successors takes place using a **broadcast** statement, which corresponds to a radio broadcast. To send a message m to all successors, a site i executes the statement

broadcast m.

The effect of this is to append m to the end of the message queue associated with each

[1]Assertions, values and variables associated with a site are subscripted with the name of that site.

link (i,j), $j \in succ_i$, after some unpredictable but finite delay.

To receive a message from a particular predecessor site k, i executes a **receive** statement

receive x **from** k.

This removes the first message from the queue associated with link (k,i) and assigns it to x; if the queue is empty, site i is delayed until a message from site k has been delivered.

3. An Algorithm for Directed Networks

We begin by defining the following functions on sites:

$|j,k|$ $\equiv$ the length of the shortest directed path from site j to site k.

$$diam(k) \equiv \max_{j \in P} |j,k|.$$

Because the network is strongly connected, $|j,k|$ and $diam(k)$ are total.

Now, consider a directed network in which a set W_i is stored at each site i, where[2]

W-Assumption:
$(\forall i: i \in P: W_i \neq \emptyset) \ \wedge \ (\forall i,j: i,j \in P: i \neq j \Rightarrow W_i \cap W_j = \emptyset)$.

For each site i, define[3]

$$Q_i^t \equiv (\cup j: |j,i| = t: W_j) - (\cup j: |j,i| < t: W_j) \qquad \text{for } 0 \leq t.$$

Thus, Q_i^t contains those values that appear in set W_j at each site j that is connected to i by a directed path of length t and no directed path with length less than t.

---

[2]It is not difficult to make arbitrary sets satisfy the W-Assumption: The set $W_k \times \{k\}$ is used in place of W_k, for all sites k.

[3]The symbol – denotes set difference.

We now derive an algorithm that establishes[4] R_i: $S_i = (\cup j: j \in P: W_j)$ at each site i, or equivalently

$$R_i\colon\ S_i = (\cup j: 0 \leq j \leq diam(i): Q_i^j).$$

Such an algorithm can be used to compute and distribute the topology of a directed network by using $W_i = pred_i \times \{i\}$.

3.1. The Loop at Site i

Site i uses a loop to establish R_i. The loop is developed from a loop invariant, which is obtained by generalizing R_i. R_i can be weakened by replacing the constant $diam(i)$ by an integer variable c_i to obtain the loop invariant:

$$P0_i\colon\ S_i = (\cup j: 0 \leq j \leq c_i: Q_i^j) \ \wedge \ 0 \leq c_i.$$

Replacing a constant by a variable is one of the standard techniques described in [Gries 81] for obtaining a loop invariant from a result assertion. $P0_i$ asserts that S_i contains values in sets W_k for all sites k connected to i by a directed path of length at most c_i.

Our first task is to make $P0_i$ true initially. The multiple assignment

$$c_i,\ S_i := 0,\ W_i$$

suffices for this because $true \Rightarrow wp(\text{"}c_i,\ S_i := 0,\ W_i\text{"}, P0_i)$.

Our next task is choose a guard β_i for the loop. β_i must satisfy $\neg \beta_i \wedge P0_i \Rightarrow R_i$, and an obvious choice for the guard is $c_i \neq diam(i)$. Unfortunately, due to the Local Topology Assumption, this guard is not computable at site i because $diam(i)$ may not be known there. We return to this problem later. For the meantime, we use $c_i \neq diam(i)$.

---

[4] It is convenient in assertions to denote $\bigcup_{Q(j)} X_j$ by $(\cup j: Q(j): X_j)$.

Finally, we develop the body of the loop. Execution of the loop body, in a state in which β_i and $P0_i$ are true must reestablish the loop invariant and make progress towards termination. Progress can be made by increasing c_i (see $P0_i$). In order to reestablish $P0_i$, values must be added to S_i. Since $wp(\text{“}c_i, S_i := c_i + 1, S_i \cup Q_i^{c_i+1}\text{”}, P0_i)$ is

$$S_i \cup Q_i^{c_i+1} = (\cup j: 0 \leq j \leq c_i + 1: Q_i^j) \quad \wedge \quad 0 \leq c_i + 1,$$

which is implied by $P0_i \wedge c_i \neq diam(i)$, the assignment

$$c_i, S_i := c_i + 1, S_i \cup Q_i^{c_i+1} \tag{3.1.0}$$

suffices.

It remains to compute $Q_i^{c_i+1}$ from values local to site i. By definition, $x \in Q_i^{c_i+1}$ if and only if (i) $x \in Q_k^{c_i}$ for some $k \in pred_i$, and (ii) there is no $k' \in pred_i$ for which $x \in Q_{k'}^{c'} \wedge c' < c_i$. That is, $x \in Q_i^{c_i+1}$ if there is a path $(p, \ldots, k, i)$ of length $c_i + 1$ where $x \in W_p$ and there is no path from p to i (through k or any other predecessor of i) with length less than c_i. Thus,

$$\begin{aligned} Q_i^{c_i+1} &= (\cup k: k \in pred_i: Q_k^{c_i}) - (\cup k': k' \in pred_i: (\cup j: 0 \leq j < c_i: Q_{k'}^j)) \\ &= (\cup k: k \in pred_i: Q_k^{c_i}) - (\cup j: 0 \leq j \leq c_i: Q_i^j) \\ &= (\cup k: k \in pred_i: Q_k^{c_i}) - S_i. \end{aligned}$$

(The last step follows from $P0_i$.) Substituting this expression for $Q_i^{c_i+1}$ into (3.1.0), the assignment statement for the loop body becomes

$$c_i, S_i := c_i + 1, S_i \cup ((\cup k: k \in pred_i: Q_k^{c_i}) - S_i)$$

which simplifies to

$$c_i, S_i := c_i + 1, S_i \cup (\cup k: k \in pred_i: Q_k^{c_i}). \tag{3.1.1}$$

This assignment can be executed at site i only if sets $Q_k^{c_i}$ for all $k \in pred_i$ are known to i. We therefore assume the existence of a routine $Acquire_i$, defined by

$$\{true\}$$
$$Acquire_i$$
$$\{(\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}.$$

Then, the program we have developed thus far is obtained by rewriting (3.1.1) using $V_i[k]$ in place of $Q_k^{c_i}$:

$$\begin{array}{l} c_i, S_i := 0, W_i; \\ \{P0_i\} \\ \textbf{do}\ \ c_i \neq diam(i) \rightarrow \{c_i \neq diam(i) \wedge P0_i\} \\ \qquad\qquad\qquad\quad Acquire_i; \\ \qquad\qquad\qquad\quad \{P0_i \wedge (\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\} \\ \qquad\qquad\qquad\quad c_i, S_i := c_i + 1, S_i \cup (\cup k: k \in pred_i: V_i[k]) \\ \qquad\qquad\qquad\quad \{P0_i\} \\ \textbf{od} \\ \{P0_i \wedge c_i = diam(i)\} \\ \{R_i\} \end{array}$$

3.2. A Computable Guard

From the definition of Q_i^t we have

$$(\forall t: diam(i) < t: Q_i^t = \emptyset) \tag{3.2.0}$$

and

$$\begin{array}{lll} Q_i^t = \emptyset \ \Rightarrow & t > diam(i) & \vee \\ & (\forall j: |j,i| = t: W_j = \emptyset) & \vee \\ & (\cup j: |j,i| = t: W_j) \subseteq (\cup j: |j,i| < t: W_j). & \end{array} \tag{3.2.1}$$

By the W-Assumption, the W_i are nonempty and disjoint, so that (3.2.1) simplifies to $Q_i^t = \emptyset \Rightarrow t > diam(i)$, which combined with (3.2.0) yields

$$t > diam(i) \Leftrightarrow Q_i^t = \emptyset. \tag{3.2.2}$$

Computation of $Q_i^{c_i}$ by site i is accomplished by maintaining a set-valued variable

T_i defined by

$$P1_i:\ T_i = (\cup j: 0 \leq j < c_i: Q_i^{c_i})$$

By making $P1_i$ an invariant of the loop, $Q_i^{c_i}$ can be computed during each iteration by evaluating $S_i - T_i$ because

$$P0_i \wedge P1_i \Rightarrow (Q_i^{c_i} = S_i - T_i). \tag{3.2.3}$$

Finally, $P0_i \wedge c_i > diam(i) \Rightarrow R_i$. Thus, we are free to choose $c_i \leq diam(i)$ as β_i the guard for the loop. This is convenient, because, according to (3.2.2), $\beta_i \Leftrightarrow Q_i^{c_i} \neq \emptyset$. And, from (3.2.3) we get $\beta_i \Leftrightarrow S_i - T_i \neq \emptyset$. This last formulation has the additional virtue that it is entirely in terms of variables maintained at site i.

Adding statements to the program to establish and maintain T_i and changing the guard as just suggested yields the following program at site i.

$c_i, S_i, T_i := 0, W_i, \emptyset;$
$\{P0_i \wedge P1_i\}$
do $S_i - T_i \neq \emptyset \rightarrow$ $\{S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i\}$
$Acquire_i;$
$\{P0_i \wedge P1_i \wedge (\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$
$c_i, S_i, T_i := c_i + 1, S_i \cup (\cup k: k \in pred_i: V_i[k]), S_i$
$\{P0_i \wedge P1_i\}$
od
$\{P0_i \wedge c_i > diam(i)\}$
$\{R_i\}$

3.3. Implementing Acquire$_i$

During iteration t of the loop, the sets Q_k^t for all $k \in pred_i$ are obtained by $Acquire_i$ routine. $P1_k$ is an invariant of the loop at site k, so k computes $Q_k^{c_k}$ during each iteration simply by evaluating $S_k - T_k$ and broadcasts the value to its successors.

The loop body at site k is executed $diam(k)+1$ times because c_k is initially 0, it is increased by one each iteration, and the body is no longer executed when $c_k > diam(k)$. Thus, if[5] the loop terminates, $diam(k)+1$ values are sent by k to each site in $succ_k$.

Now consider a site i, $i \in succ_k$. Because messages sent along link (k,i) are delivered in the order sent (due to VC1 and VC2), the successive values received on that link are $Q_k^0, Q_k^1, \ldots, Q_k^{diam(k)}$. Therefore, we can implement $Acquire_i$ by

broadcast $S_i - T_i$;
cobegin $\underset{k \in pred_i}{//}$ **receive** $V_i[k]$ **from** k $\{V_i[k] = Q_k^{c_i}\}$ **coend**
$\{(\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$

Unfortunately, this introduces the possibility of infinite blocking. The **cobegin** terminates only if every **receive** terminates, and a **receive** terminates only if there is a message available for receipt. $Acquire_i$ is executed once per loop iteration, i.e. $diam(i)+1$ times. Therefore, at least $diam(i)+1$ messages must be sent on link (k,i) for each $k \in pred_i$ to prevent infinite blocking at site i.

From the definition of $diam$ and the fact that i is a successor of k, we obtain

$$diam(k)+1 \geq diam(i). \tag{3.3.0}$$

Consequently, if k makes a **broadcast** after completing $diam(k)+1$ iterations then the total number of messages sent by k is $diam(k)+2 \geq diam(i)+1$ (by (3.3.0)) and infinite blocking at i is avoided.

Inserting the code for $Acquire_i$ into the program and adding a **broadcast** after the loop yields:

[5] "If" because absence of deadlock and loop termination have not yet been shown.

$c_i, S_i, T_i := 0, W_i, \emptyset;$
$\{P0_i \wedge P1_i\}$
do $S_i - T_i \neq \emptyset \rightarrow \{S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i\}$
broadcast $S_i - T_i;$
$\{S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i\}$
cobegin $\underset{k \in pred_i}{//}$ $\{S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i\}$
receive $V_i[k]$ **from** k
$\{P0_i \wedge P1_i \wedge V_i[k] = Q_k^{c_i}\}$
coend;
$\{P0_i \wedge P1_i \wedge (\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$
$c_i, S_i, T_i := c_i + 1, S_i \cup (\cup k: k \in pred_i: V_i[k]), S_i$
$\{P0_i \wedge P1_i\}$
od;
$\{c_i > diam(i) \wedge P0_i \wedge P1_i\}$
broadcast $S_i - T_i$
$\{c_i > diam(i) \wedge P0_i \wedge P1_i\}$
$\{R_i\}$

3.4. Satisfaction Proof

General Approach

A satisfaction proof establishes the validity of the postcondition of each **receive**. Such a proof is needed because the postcondition of a **receive** can depend on the value of the message that is received, which in turn is based on the state of the sender, and not the precondition of the **receive**. The proof rules and satisfaction formula presented here are derived from those for virtual circuits presented in [Schlichting & Schneider 83].

The state of a link $(i,j) \in L$ is modeled by auxiliary variables σ_{ij} and ρ_{ij}; σ_{ij} is the sequence of messages sent on (i,j) and ρ_{ij} is the sequence of messages received on (i,j). The following notation will be used for operations on sequences $s = <s[0], s[1], \cdots, s[n]>$:

$s + val$ is the sequence that results when val is appended to s.

$s \leq t$ is true if s is a prefix of t.

$t-s$ is the sequence obtained by deleting prefix s from t.

$|s|$ is the number of elements in s.

Φ is the empty sequence.

Because each link is a virtual circuit, the following holds:

Virtual Circuit Network Axiom: $(\forall (i,j): (i,j) \in L : \rho_{ij} \leq \sigma_{ij})$.

The proof rules for **broadcast** and receive are as follows. The statement **broadcast** m_i is equivalent to

cobegin
  // **send** m_i **to** j
$j \in succ_i$
coend

where **send** is the asynchronous **send** of [Schlichting & Schneider 83], which does not delay its invoker. Thus, each **send** is equivalent to the assignment

$$\sigma_{ij} := \sigma_{ij} + m_i .$$

Using *wp* for this assignment statement, we get the

Send Axiom: $\{P^{\sigma_{ij}}_{\sigma_{ij}+m_i}\}$ **send** m_i **to** j $\{P\}$.

Let σ_i denote the vector of variables σ_{ij} for all $j \in succ_i$ (in some fixed order) and $\sigma_i + m_i$ the vector of values $\sigma_{ij} + m_i$ for all $j \in succ_i$ (in the same order). Then, using the **cobegin** rule[6] [Owicki & Gries 76], we can combine instances of the Send Axiom to obtain an axiom for a **broadcast**:

Broadcast Axiom: $\{P^{\sigma_i}_{\sigma_i+m_i}\}$ **broadcast** m_i $\{P\}$.

[6]Non-interference follows trivially because all variables altered by the components of the cobegin are disjoint.

The axiom for a **receive** is

Virtual Circuit Receive Axiom: $\{P\}\ r_{ki}$: **receive** x_i **from** k $\{Q\}$,

where P and Q are arbitrary predicates. The axiom allows anything to be asserted after a **receive**; a later satisfaction proof ensures that the asserted postcondition is valid.

Execution of r_{ki} is equivalent to

$$x_i, \rho_{ki} := MTEXT,\ \rho_{ki} + MTEXT \tag{3.4.0}$$

when $(\sigma_{ki} - \rho_{ki}) \neq \Phi$ and $MTEXT = (\sigma_{ki} - \rho_{ki})[0]$. Thus, using *wp* for (3.4.0) with respect to $post(r_{ki})$, a satisfaction formula $Sat_{vc}(r_{ki})$ whose validity ensures the validity of $post(r_{ki})$ is

$$(pre(r_{ki}) \wedge (\sigma_{ki} - \rho_{ki}) \neq \Phi \wedge MTEXT = (\sigma_{ki} - \rho_{ki})[0]) \Rightarrow post(r_{ki})^{x_i,\ \rho_{ki}}_{MTEXT,\ \rho_{ki} + MTEXT}.$$

Satisfaction for Our Program

In order to perform the satisfaction proof, the proof outline for our program is strengthened by adding predicates about message sequences. Define

PR_i: $(\forall k: k \in pred_i: |\rho_{ki}| = c_i)$
PS_i: $(\forall j: j \in succ_i: |\sigma_{ij}| = c_i)$.

PR_i relates c_i to the number of messages received by site i. PS_i relates c_i to the number of messages sent by site i to each of its successors. It is easily verified that both are loop invariants for the process at site i.

The assertion

SI: $(\forall i,j: (i,j) \in L: (\forall t: 0 \leq t < |\sigma_{ij}|: \sigma_{ij}[t] = Q_i^j))$

relates the contents of a message sent over a given link to its relative position in the sequence of messages sent over that link. SI is true at all sites throughout the computa-

tion, as shown informally by the following inductive argument.

Before any messages are sent, SI is trivially true because the range $0 \leq t < |\sigma_{ij}|$ is empty. The σ_{ij} are altered only by execution of **broadcast** $S_i - T_i$ by site i, which we know by (3.2.3) is equivalent to **broadcast** $Q_i^{c_i}$. Since PS_i is a loop invariant, it is true before performing the **broadcast**. Therefore, from PS_i, the message broadcast is $\sigma_{ij}[c_i]$ on each link (i,j), so SI is true.

The program for site i with the additional assertions about the sequences of messages sent is shown below:

$\{(\forall k: k \in pred_i: |\rho_{ki}|=0) \wedge (\forall j: j \in succ_i: |\sigma_{ij}|=0)\}$
$c_i, S_i, T_i, := 0, W_i, \emptyset;\ \{P0_i \wedge P1_i \wedge PR_i \wedge PS_i \wedge SI\}$
do $S_i - T_i \neq \emptyset \rightarrow \{S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i \wedge PR_i \wedge PS_i \wedge S_i\}$
broadcast $S_i - T_i$;
$\{S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i \wedge PR_i \wedge (\forall j: j \in succ_i: |\sigma_{ij}|=c_i+1) \wedge SI\}$
cobegin $\underset{k \in pred_i}{//}$ $\{S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i \wedge |\rho_{ki}|=c_i \wedge$
$(\forall j: j \in succ_i: |\sigma_{ij}|=c_i+1) \wedge SI\}$
r_{ki}: **receive** $V_i[k]$ **from** k
$\{P0_i \wedge P1_i \wedge |\rho_{ki}|=c_i+1 \wedge$
$(\forall j: j \in succ_i: |\sigma_{ij}|=c_i+1) \wedge SI \wedge V_i[k] = Q_k^{c_i}\}$
coend;
$\{P0_i \wedge P1_i \wedge (\forall k: k \in pred_i: |\rho_{ki}|=c_i+1) \wedge$
$(\forall j: j \in succ_i: |\sigma_{ij}|=c_i+1) \wedge SI \wedge (\forall k: k \in pred_i: V_i[k] = Q_k^{c_i})\}$
$c_i, S_i, T_i := c_i+1, S_i \cup (\cup k: k \in pred_i: V_i[k]), S_i$
$\{P0_i \wedge P1_i \wedge PR_i \wedge PS_i \wedge SI\}$
od;
$\{c_i > diam(i) \wedge P0_i \wedge P1_i \wedge PR_i \wedge PS_i \wedge SI\}$
broadcast $S_i - T_i$
$\{c_i > diam(i) \wedge P0_i \wedge P1_i \wedge PR_i \wedge (\forall j: j \in succ_i: |\sigma_{ij}|=c_i+1)\}$
$\{R_i\}$

Sat_{vc} can now be proved valid for each **receive**. We give only the details for r_{ki}; the others follow by symmetry. Substituting $pre(r_{ki})$ and $post(r_{ki})$ from the proof out-

line above into $Sat_{vc}(r_{ki})$ yields:

$$S_i - T_i \neq \emptyset \wedge P0_i \wedge P1_i \wedge |\rho_{ki}| = c_i \wedge (\forall j: j \in succ_i: |\sigma_{ij}| = c_i + 1) \wedge SI \wedge (\sigma_{ki} - \rho_{ki}) \neq \Phi \wedge MTEXT = (\sigma_{ki} - \rho_{ki})[0]) \Rightarrow (P0_i \wedge P1_i \wedge |\rho_{ki} + MTEXT| = c_i + 1 \wedge (\forall j: j \in succ_i: |\sigma_{ij}| = c_i + 1) \wedge SI \wedge MTEXT = Q_k^{c_i}).$$

All but the last conjunct of the consequent follow immediately from the antecedent. To see that the last conjunct is implied by the antecedent, note that

$$(|\rho_{ki}| = c_i \wedge (\sigma_{ki} - \rho_{ki}) \neq \Phi \wedge MTEXT = (\sigma_{ki} - \rho_{ki})[0]) \Rightarrow MTEXT = \sigma_{ki}[c_i].$$

From SI, we deduce $MTEXT = Q_k^{c_i}$, and the satisfaction proof is completed.

3.5. Non-interference

The final step in proving partial correctness is to show non-interference. This establishes that the proofs remain valid even if the sites execute in parallel, as would be the case in a distributed system.

Consider site i. The only references in assertions of the proof outline to variables that are modified by another process are in SI. SI is a conjunct in every assertion, so it is not interfered with. Consequently, non-interference follows.

3.6. Termination of Loops

A variant function (called a bound function in [Gries 81]) for the loop at site i is

v: $diam(i) + 1 - c_i$.

Each iteration of the loop decreases v. When $v = 0$ then $diam(i) + 1 = c_i$. Thus, by (3.2.2), $Q_i^{c_i} = \emptyset$, and so by (3.2.3) the guard $S_i - T_i \neq \emptyset$ is false and the loop terminates.

3.7. Absence of Deadlock

Suppose the process at site i is deadlocked, waiting forever at some **receive** r_{ki} for a message on link (k,i). By lemma 1 below, k is also deadlocked. By induction and the finiteness of the system, there exists a cycle of deadlocked processes, each waiting for a message from the next. For each such link (k,i) we have

$c_i = |\rho_{ki}|$ from PR_i

$= |\sigma_{ki}|$ because link (k,i) is empty forever

$= c_k + 1$ from $pre(r_{jk})$ at site k.

By transitivity and the fact that the links form a cycle, we conclude $c_i > c_i$, a contradiction. Hence, there is no deadlock.

Lemma 1: If process i is deadlocked at a **receive** r_{ki}, then k is deadlocked.

Proof: Since no message is forthcoming from k, k has terminated or is deadlocked. We assume k has terminated and prove the contradiction $diam(i) > diam(i)$.

$diam(i) \geq c_i$ from $P0_i$ and $S_i - T_i \neq \Phi$ in $pre(r_{ki})$, and (3.2.2), and (3.2.3).

$= |\rho_{ik}|$ from PR_i.

$= |\sigma_{ik}|$ because link (k,i) is empty forever.

$= c_k + 1$ from the postcondition of process k.

$> diam(k) + 1$ from the postcondition of process k.

$\geq diam(i)$ due to (3.3.0).

4. Discussion

The strategy we used to derive this distributed algorithm is essentially the "programming calculus" first proposed for sequential programs in [Dijkstra 76]. When nonlocal values were required (as in computing $Q_i^{c_i+1}$), a **receive** statement was employed and we assumed that the correct values would be received when it executed. A

broadcast ensured that correct values were available for receipt; an invariant (SI) characterized these correct values. A satisfaction proof established that receive statements had the desired effect.

The behavior of the algorithm when run on a network with bidirectional communication links is identical to that of a well known bidirectional network routing algorithm, attributed to R. G. Gallager in [Friedman 79]. There, tables are constructed by successively adding information from farther nodes.

Acknowledgements

We would like to thank Bowen Alpern, David Gries, Abha Moitra and Dave Wright for their comments on early drafts of this paper and Carl E. Landwehr for bringing the problem to our attention.

References

[Dijkstra 76] Dijkstra, E.W. *A Discipline of Programming.* Prentice Hall, 1976.

[Friedman 79] Friedman, D.U. Communication Complexity of Distributed Shortest Path Algorithms. LIDS-TH-886, M.I.T., February 1979.

[Gries 81] Gries, D. *The Science of Programming.* Springer Verlag, New York, 1981.

[Owicki & Gries 76] Owicki, S.S., and D. Gries. An Axiomatic Proof Technique for Parallel Programs I. *Acta Informatica 6* (1976), 319-340.

[Schlichting & Schneider 83] Schlichting, R., and F.B. Schneider. Using Message-Passing for Distributed Programming: Proof Rules and Disciplines. TR 82-491, Department of Computer Science, Cornell University, May 1982.

[Tannenbaum 81] Tannenbaum, A. *Computer Networks.* Prentice Hall, Englewood Cliffs, New Jersey, 1981.