Correctness Proofs of Distributed Termination Algorithms

KRZYSZTOF R. APT
L.I.T.P., Université Paris 7

The problem of correctness of the solutions to the distributed termination problem of Francez [7] is addressed. Correctness criteria are formalized in the customary framework for program correctness. A very simple proof method is proposed and applied to show correctness of a solution to the problem. It allows us to reason about liveness properties of temporal logic (see, e.g., Manna and Pnueli [12]) using a new notion of *weak total correctness*.

Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs.

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: CSP, deadlock, global invariant, weak total correctness

1. INTRODUCTION

This paper deals with the distributed termination problem of Francez [7] which has received a great deal of attention in the literature. Several solutions to this problem, or its variants, have been proposed, their correctness, however, has been rarely discussed. In fact, it is usually not even explicitly stated what properties such a solution should satisfy.

Notable exceptions in this matter are papers of Dijkstra, Feijen, and Van Gasteren [5] and Topor [14] in which solutions to the problem are systematically derived together with their correctness proofs. On the other hand, they are presented in a simplistic abstract setting in which, for example, no distinction can be made between deadlock and termination. Also, as we shall see in the next section, not all desired properties of a solution are addressed there. Systematically derived solutions in the abstract setting of [5] are extremely helpful in understanding corresponding solutions presented in CSP. However, such a presentation should not relieve us from providing rigorous correctness proofs—an issue we address in this paper.

A preliminary version of this paper appeared in *Logics and Models of Concurrent Systems*, K. R. Apt, Ed., Springer Verlag, NATO ASI Series, vol. F13, 1985.
Author's address: L.I.T.P. Université Paris 7, 2, Place Jussieu, 75251 Paris, France.

Clearly, it would be preferable to derive the solutions in CSP *together* with their correctness proofs, perhaps by transforming accordingly the solutions first provided in the abstract setting. Unfortunately, such techniques are not presently available.

This paper is organized as follows. In the next section we define the problem and propose the correctness criteria that the solutions to the problem should satisfy. Then, in Section 3, we formalize these criteria in the usual framework for program correctness, and in Section 4 we propose a very simple proof method which allows us to prove the criteria. In Section 5 we provide a simple solution to the problem, and in the next section give a detailed proof of its correctness. Finally, in section Section 7, we assess the proposed proof method.

Throughout this paper we assume that the reader is familiar with the language of Communicating Sequential Processes (CSP), as defined in Hoare [10], and has some experience in the proofs of correctness of very simple loop-free sequential programs.

2. DISTRIBUTED TERMINATION PROBLEM

Suppose that a CSP program

$$P = [P_1 \parallel \ldots \parallel P_n],$$

where for every $1 \leq i \leq n$, $P_i :: \mathrm{INIT}_i ; *[S_i]$ is given. We assume that each S_i is of the form $\underset{j \in \Gamma_i}{\square} g_{i,j} \rightarrow S_{i,j}$ for an index set Γ_i and that

(i) each $g_{i,j}$ contains an I/O command addressing P_j,
(ii) none of the statements INIT_i, $S_{i,j}$ contains an I/O command.

We say then that P is in a *normal form*. Suppose moreover that we associate with each P_i a *stability condition*, B_i, a Boolean expression involving variables of P_i and possibly some auxillary variables. By a *global stability* we mean a situation in which each process is at the main loop entry with its stability condition B_i true.

We now adopt the following two assumptions:

(a) No communication can take place between a pair of processes while both of their stability conditions hold.
(b) Whenever deadlock takes place the global stability condition is reached.

The *distributed termination* problem (see Francez [7]) is the problem of transforming P into another program P' which eventually properly terminates whenever the global stability condition is reached. Its formulation presupposes that P' has a particular format so that each computation of P' can be restricted to a computation of P by omitting the instructions involving new variables.

We say that the global stability condition is (not) reached in a computation of P' if it is (not) reached in its restriction to a computation of P. In turn, the global stability condition is reached (not reached) in a computation of P if it holds (does not hold) in a (any) possible global state of the computation. We consider here partially ordered computations in the sense of [11].

We now postulate four properties that a solution P' to the distributed termination problem should satisfy (see Apt and Richier [4]):

(1) Whenever P' properly terminates then the global stability condition is reached.
(2) There is no deadlock.
(3) If the global stability condition is reached then P' eventually properly terminates.
(4) If the global stability condition is not reached then eventually a statement from the original program is executed.

The last property excludes the situations in which the transformed parallel program endlessly executes the added control parts dealing with termination detection.

In the abstract framework of [5] only the first property is proved. The second property is not meaningful, as there deadlock coincides with termination. In turn, satisfaction of the third property is argued informally, and the fourth property is not mentioned.

Solutions to the distributed termination problem are obtained by arranging some additional communications between the processes P_i. Most of them are programs, $P' = [P_1 \| \ldots \| P_n]$, in a normal form where for every i, $1 \leq i \leq n$,

$P_i ::$ INIT$_i$; . . . .
*[$\Box_{j \in \Gamma_i}$ . . . ; $g_{i,j} \rightarrow$ . . . ; $S_{i,j}$
□ CONTROL PART$_i$
]

where . . . stand for some added Boolean conditions or statements not containing I/O commands, and CONTROL PART$_i$ stands for a part of the loop dealing with additional communications.

For solutions of the above form we now express the four properties which we have introduced, using the customary terminology dealing with program correctness.

3. FORMALIZATION OF THE CORRECTNESS CRITERIA

Let p, q, I be assertions from an assertion language and let S be a CSP program. We say that $\{p\}\ S\ \{q\}$ holds in the sense of *partial correctness* if all properly terminating computations of S starting in a state satisfying p terminate in a state satisfying q. We say that $\{p\}\ S\ \{q\}$ holds in the sense of *weak total correctness* if it holds in the sense of partial correctness and, moreover, no computation of S starting in a state satisfying p fails or diverges. We say that S is *deadlock-free relative to* p if in the computations of S starting in a state satisfying p no deadlock can arise. If $p \equiv$ *true*, then we simply say that P is *deadlock-free*.

Finally, we say that $\{p\}\ S\ \{q\}$ holds in the sense of *total correctness* if it holds in the sense of weak total correctness and, moreover, S is deadlock-free relative to p. Thus, when $\{p\}\ S\ \{q\}$ holds in the sense of total correctness, then all computations of S starting in a state satisfying p properly terminate.

Also, for CSP programs in a normal form we introduce the notion of a global invariant I. We say that I is a *global invariant of* P *relative to* p if in all

computations of P, starting in a state satisfying p, I holds whenever each process P_i is at the main loop entry. If $p \equiv true$, then we simply say that I is a *global invariant of P*.

Now property 1 simply means that

$$\{true\}\ P'\{\bigwedge_{i=1}^{n} B_i\} \tag{1}$$

holds in the sense of partial correctness.

Property 2 means that P' is deadlock-free.

Property 3 cannot be expressed by referring directly to the program P'. Even though it refers to the termination of P', it is not equivalent to its (weak) total correctness because the starting point—the global stability situation—is not the initial one. It is a control point that can be reached in the course of a computation.

However, we can still express property 3 by referring to the weak total correctness of a program derived from P'. Consider the following program:

CONTROL PART =
$[P_1 :: *[\text{CONTROL PART}_1] \parallel \ldots \parallel P_n :: *[\text{CONTROL PART}_n]]$.

We now claim that to establish property 3 it is sufficient to prove for an appropriately chosen global invariant I of P' that

$$\{I \wedge \bigwedge_{i=1}^{n} B_i\}\ \text{CONTROL PART}\ \{true\} \tag{2}$$

in the sense of total correctness.

Indeed, suppose that in a computation of P' the global stability condition is reached. Then, $I \wedge \wedge_{i=1}^{n} B_i$ holds where I is a global invariant of P'. By assumption (a), concerning the original program P, no statement from P can be executed any more. Thus the part of P' that remains to be executed is equivalent to the program CONTROL PART. Now, because of (2), property 3 holds.

Now consider property 4. As before, we express it by referring to the program CONTROL PART. Assuming that property 2 holds, property 4 clearly holds if

$$\{I \wedge \neg \bigwedge_{i=1}^{n} B_i\}\ \text{CONTROL PART}\ \{true\} \tag{3}$$

holds in the sense of weak total correctness. Indeed, (3) guarantees that if the global stability condition is not currently holding, then every segment of the computation that performs actions from the CONTROL PART only must either terminate or deadlock. In the absence of deadlock this is equivalent to property 4.

Assuming that property 2 is already established, to show property 3 it is sufficient to prove (2) in the sense of weak total correctness. Now (2) and (3) can be combined into the formula:

$$\{I\}\ \text{CONTROL PART}\ \{true\} \tag{4}$$

in the sense of weak total correctness.

4. PROOF METHOD

We now present a simple proof method which allows us to handle the properties discussed in the previous section. The proof method uses to great advantage the coarse granularity that can be imposed on CSP programs in defining the atomic transitions due to the disjointness of processes. It can only be applied to CSP programs being in a normal form. So assume that $P = [P_1 \parallel \ldots \parallel P_n]$ is such a program.

Given a guard $g_{i,j}$, we denote by $b_{i,j}$ the conjunction of its Boolean parts. We say that guards $g_{i,j}$ and $g_{j,i}$ *match* if one contains an input command and the other an output command whose expressions are of the same type. The notation implies that these I/O commands address each other (i.e., they are within the texts of P_i and P_j, respectively, and address P_j and P_i, respectively).

Given two matching guards $g_{i,j}$ and $g_{j,i}$, we denote by $\mathrm{Eff}(g_{i,j}, g_{j,i})$ the effect of the communication between their I/O commands. It is the assignment whose left-hand side is the input variable and the right-hand side is the output expression. Finally, let

$$\mathrm{TERMINATED} = \bigwedge_{\substack{1 \le i \le n \\ j \in \Gamma_i}} \neg\, \mathrm{b}_{i,j}$$

Observe that TERMINATED holds upon termination of P.

Now consider partial correctness. We propose the following proof rule:

Rule 1. *Partial Correctness*

$$\frac{\begin{array}{l}\{p\}\ \mathrm{INIT}_1 ; \ldots ; \mathrm{INIT}_n\{I\}, \\ \{I \wedge b_{i,j} \wedge b_{j,i}\}\mathrm{Eff}(g_{i,j}, g_{j,i})\ ;\ S_{i,j}\ ;\ S_{j,i}\{I\} \\ \text{for all } i, j \text{ s.t. } i \in \Gamma_j,\ j \in \Gamma_i, \text{ and } g_{i,j},\ g_{j,i} \text{ match}\end{array}}{\{p\}P\{I \wedge \mathrm{TERMINATED}\}}$$

This rule has to be used in conjunction with the usual proof system for *partial* correctness of nondeterministic programs (see, e.g., Apt [1]) in order to be able to establish its premises. If the premises of this rule hold, then we can also deduce that I is a global invariant of P relative to p. Informally, it can phrased as follows. If I is established upon execution of all the INIT_i sections and is preserved by a joint execution of each pair of branches of the main loops with matching guards, then I holds upon exit. Thus, I holds upon exit for such special "synchronized" computations. Intuitively this rule is sound because due to the disjointness of processes each terminating computation of P can be appropriately rearranged so that it becomes a "synchronized" computation.

Consider now weak total correctness. We adopt the following proof rule:

Rule 2. *Weak Total Correctness*

$$\frac{\begin{array}{l}\{p\}\mathrm{INIT}_1 ; \ldots ; \mathrm{INIT}_n\{I \wedge t \ge 0\}, \\ \{I \wedge b_{i,j} \wedge b_{j,i} \wedge z = t \wedge t \ge 0\}\mathrm{Eff}(g_{i,j}, g_{j,i})\ ;\ S_{i,j}\ ;\ S_{j,i}\{I \wedge 0 \le t < z\} \\ \text{for all } i, j \text{ s.t. } i \in \Gamma_j,\ j \in \Gamma_i, \text{ and } g_{i,j},\ g_{j,i} \text{ match}\end{array}}{\{p\}P\{I \wedge \mathrm{TERMINATED}\}}$$

where z does not appear in P or t and t is an integer-valued expression.

This rule has to be used in conjunction with the standard proof system for *total* correctness of nondeterministic programs (see, e.g., Apt [1]) in order to establish its premises. It is a usual modification of the rule concerning partial correctness which guarantees lack of divergence.

Finally, consider deadlock freedom. Let

$$\text{BLOCKED} = \bigwedge \{\neg b_{i,j} \vee \neg b_{j,i} : 1 \leq i, j \leq n,\ i \in \Gamma_j,\ j \in \Gamma_i,\ g_{i,j} \text{ and } g_{j,i} \text{ match}\}$$

Observe that in a given state of P the formula BLOCKED holds if and only if no communication between the processes is possible. We now propose the following proof rule:

Rule 3. *Deadlock Freedom*

$$\frac{I \text{ is a global invariant of } P \text{ relative to } p,\quad I \wedge \text{BLOCKED} \rightarrow \text{TERMINATED}.}{P \text{ is deadlock-free relative to } p.}$$

The above rules are used in conjunction with a rule of auxiliary variables.

Let A be a set of variables of a program S. A is called a set of *auxiliary variables* of S if:

(i) All variables from A appear in S only in assignments.
(ii) No variable of S which is not in A depends on the variables from A. In other words, there does not exist an assignment $x := t$ in S such that $x \notin A$ and t contains a variable from A.

Thus, for example, $\{z\}$ and $\{y, z\}$ are the only nonempty sets of auxiliary variables of the program

$$[P_1 :: z := y\ ;\ P_2! \times \| P_2 :: P_1?\ u\ ;\ u := u + 1].$$

We now adopt the following proof rule first introduced in Owicki and Gries [13].

Rule 4. *Auxiliary Variables*

Let A be a set of auxiliary variables of a program S. Let S' be obtained from S by deleting all assignments to the variables in A. Then,

$$\frac{\{p\}\ S\ \{q\}}{\{p\}\ S'\ \{q\}}$$

provided q has no free variable from A.

Also, if S is deadlock-free relative to p, then so is S'.

We use this rule both in the proofs of partial and of (weak) total correctness. Also, we freely use the well-known consequence rule which allows us to strengthen the preconditions and weaken the postconditions of a correctness statement $\{p\}\ S\ \{q\}$.

5. A SOLUTION

We now present a simple solution to the distributed termination problem. It is a combination of the solutions proposed by Francez, Rodeh, and Sintzoff [8] and (in an abstract setting) Dijkstra, Feijen, and Van Gasteren [5].

We assume that the graph consisting of all communication channels within P contains a Hamiltonian cycle (i.e., a communication path visiting each process exactly once). In the resulting ring the neighbors of P_i are P_{i-1} and P_{i+1}, where counting is done within $\{1, \ldots, n\}$ cyclically.

We first present a solution in which the global stability condition is detected by one process, say P_1. It has the following form, where the newly introduced variables s_i, send$_i$, and moved$_i$ do not appear in the original program P:

For $i = 1$
$P_i ::$ INIT$_i$; send$_i$:= *true* ;
*[□ $g_{i,j} \rightarrow S_{i,j}$
$j \in \Gamma_i$
□ B_i ; send$_i$; $P_{i+1}!$ *true* → send$_i$:= *false*
□ $P_{i-1}?$ $s_i \rightarrow$ [$s_i \rightarrow$ halt □ ¬ $s_i \rightarrow$ send$_i$:= *true*]
]

and, for $i \neq 1$,

$P_i ::$ send INIT$_i$; := *false* ; moved$_i$:= *false* :
*[□ $g_{i,j} \rightarrow$ moved$_i$:= *true* ; $S_{i,j}$
$j \in \Gamma_i$
□ $P_{i-1}?$ $s_i \rightarrow$ send$_i$:= *true*
□ B_i ; send$_i$; $P_{i+1}!$ ($s_i \wedge$ ¬ moved$_i$) → send$_i$:= *false* ;
moved$_i$:= *false*
]

In this program we use the halt instruction with an obvious meaning. Informally, P_1 decides to send a *probe true* to its right-hand side neighbor when its stability condition B_1 holds. A probe can be transmitted by a process P_i further to its right-hand neighbor when in turn its stability condition holds. Each process writes into the probe, its current status being reflected by the variable moved; moved turns to *true* when a communication from the original program takes place and turns to *false* when the probe is sent to the right-hand side neighbor. Thus, moved$_i$ indicates whether process P_i has participated in a communication from the original program since the *last time* it had passed the probe. P_1 decides to stop its execution when a probe has made a full cycle remaining true. This will happen if all the moved variables are false at the moment of *receiving* the probe from the left-hand side neighbor.

We now modify this program by arranging that P_1 send a final termination wave through the ring once it detects the global stability condition. To this purpose, we introduce in all P_is two new Boolean variables, detected$_i$ and done$_i$. The program has the following form:

For $i = 1$
$P_i ::$ done$_i$ send$_i$:= *true* ; done$_i$:= *false* ; detected$_i$:= *false* ;
*[□ INIT$_{ij}$ done$_i$, $g_{i,j} \rightarrow S_{i,j}$
$j \in \Gamma_p$
□ ¬ done$_i$; B_i ; send$_i$; $P_{i+1}!$ *true* → send$_i$:= *false*
□ ¬ done$_i$; $P_{i-1}?$ $s_i \rightarrow$
[$s_i \rightarrow$ detected$_i$:= *true* □ ¬ $s_i \rightarrow$ send$_i$:= *true*]
□ detected$_i$; $P_{i+1}!$ *end* → detected$_i$:= *false*
□ ¬ done$_i$; $P_{i-1}?$ *end* → done$_i$:= *true*
]

and for $i \neq 1$

P_i :: INIT$_{ij}$ send$_i$:= *false* ; moved$_i$:= *false* ; done$_i$:= *false* ; detected$_i$:= *false* ;
*[□ ¬ done$_i$; $g_{i,j}$ → moved$_i$:= *true* ; $S_{i,j}$
$j \in \Gamma_i$
□ ¬ done$_i$; P_{i-1}? s_i → send$_i$:= *true*
□ ¬ done$_i$; B_i ; send$_i$; P_{i+1}! (s_i ¬ moved$_i$) → send$_i$:= *false* ;
moved$_i$:= *false*
□ ¬ done$_i$; P_{i-1}? *end* → detected$_i$:= *true* ; done$_i$:= *true*
□ detected$_i$; P_{i+1}! *end* → detected$_i$:= *false*
]

We assume that *end* is a signal of a new type not used in the original program. (Actually, to avoid confusion in the transmission of the probe, we also have to assume that in the original program no messages are of type *Boolean*. If this is not the case, then we can always replace the probe by a Boolean-valued message of a new type.)

6. CORRECTNESS PROOF

We now prove correctness of the solution given in the previous section using the proof method introduced in Section 4. We do this by proving properties 1–4 of Section 2 as formalized in Section 3.

PROOF OF PROPERTY 1. We first modify the program given in the previous section by introducing in process P_1 auxiliary variables received$_1$ and forward$_1$. The variable received$_1$ is introduced in order to distinguish the situation when s_1 is initially true from the one when s_1 turns true after the communication with P_n. The variable forward$_1$ is used to remember whether or not P_1 sent the *end* signal to P_2. Note that this fact cannot be expressed by referring to the variable detected$_1$. This augmented version of P_1 has the following form:

P_1 :: INIT$_1$; send$_1$:= *true* ; done$_1$:= *false* ; detected$_i$:= *false* ;
received$_1$:= *false* ; forward$_1$:= *false* ;
*[□ ¬ done$_1$; $g_{1,j}$ → $S_{1,j}$
$j \in \Gamma_1$
□ ¬ done$_1$; B_1 ; send$_1$; P_2! *true* → send$_1$:= *false*
□ ¬ done$_1$; P_n? s_1 → received$_1$:= *true* ;
[s_1 → detected$_1$:= *true* □ ¬ s_1 → send$_1$:= *true*]
□ detected$_1$; P_2! *end* → forward$_1$:= *true* ;
detected$_1$:= *false*
□ ¬ done$_1$; P_n? *end* → done$_1$:= *true*
]

Other processes remain unchanged. Call this modified program R. In order to establish property 1, it is sufficient, by rule 4, to find a global invariant of R which upon its termination implies $\wedge_{i=1}^{n} B_i$.

We do this by establishing a sequence of successively stronger global invariants whose final element is the desired I. We call a program Eff($g'_{i,j}$, $g'_{j,i}$) ; $S'_{i,j}$; $S'_{j,i}$, corresponding to a joint execution of two branches of the main loops with matching guards in a program being in a normal form, a *transition*. For a convenient expression of loop invariants, we identify the Boolean values *false*, *true* with 0 and 1, respectively. To avoid excessive use of brackets we assume

that "→" binds weaker than other connectives. Let

$$I_1 \equiv \sum_{i=1}^{n} \text{send}_i \leq 1.$$

Then I_1 is clearly a global invariant of R: it is established by the initial assignments and is preserved by every transition, as the setting of a send variable to *true* is always accompanied by changing another send variable from *true* to *false*. Now consider

$$I_2 \equiv \forall i > 1\ [s_i \wedge \text{send}_i \rightarrow \forall j(1 \leq j < i \rightarrow B_j) \vee \exists j \geq i\ \text{moved}_j] \wedge [s_1 \wedge \text{received}_1 \rightarrow \forall j\ (1 \leq j \leq n \rightarrow B_j)].$$

We now claim that $I_1 \wedge I_2$ is a global invariant of R. First note that I_2 is established by the initial assignments in a trivial way.

Next, consider a transition corresponding to a communication from the original program P. Assume that initially $I_1 \wedge I_2$ and that the Boolean conditions of the guards hold.

Now consider the first conjunct of I_2. If, initially, for no $i > 1$ does $s_i \wedge \text{send}_i$ hold, then this conjunct is preserved since the transition does not alter s_i or send_i. Suppose now that initially for some $i > 1$, $s_i \wedge \text{send}_i$ holds. If initially also $\exists j \geq i\ \text{moved}_i$ holds, then this conjunct is preserved. If initially $\forall j\ (1 \leq j < i \rightarrow B_j)$ holds, then, by assumption (a) in Section 2, at least one of the processes involved in the transition has an index $\geq i$. The transition sets its moved variable to true which establishes $\exists j \geq i\ \text{moved}_j$.

The second conjunct of I_2 is obviously preserved—if initially $s_1 \wedge \text{received}_1$ does not hold, then it does not hold at the end of the transition either. If initially $s_1 \wedge \text{received}_1$ holds, then also $\forall j\ (1 \leq j \leq n \rightarrow B_j)$ initially holds, and again, by assumption (a) of Section 2, the discussed transition cannot take place.

Now consider a transition corresponding to a sending of the probe from P_i to P_{i+1} $(1 \leq i \leq n)$. Suppose that at the end of the transition $s_k \wedge \text{send}_k$ for some k $(1 < k \leq n)$ holds. Due to the global invariant I_1 and the form of the transition, we can conclude that $k = i + 1$. Thus, in the initial state, $B_i \wedge s_i \wedge \neg\ \text{moved}_i \wedge \text{send}_i$ holds. Now, due to I_2, initially

$$\forall j\ (1 \leq j < i \rightarrow B_j) \vee \exists j \geq i\ \text{moved}_j$$

holds. Thus, initially

$$\forall j\ (1 \leq j < i + 1 \rightarrow B_j) \vee \exists j \geq i + 1\ \text{moved}_j$$

holds. This formula is not affected by the execution of the transition. Thus, at the end of the transition, the first conjunct of I_2 holds.

Now suppose that at the end of the transition $s_1 \wedge \text{received}_1$ holds. If initially $s_1 \wedge \text{received}_1$ holds, then also $\forall j\ (1 \leq j \leq n \rightarrow B_j)$ initially holds. Now suppose that initially $s_1 \wedge \text{received}_1$ does not hold. Then the transition consists of sending the probe from P_n to P_1. Therefore, initially $B_n \wedge s_n \wedge \neg\ \text{moved}_n \wedge \text{send}_n$ holds, and because of I_2, $\forall j\ (1 \leq j \leq n \rightarrow B_j)$ initially holds as well. But this formula is preserved by the execution of the transition. So at the end of the transition the second conjunct of I_2 holds.

The other transitions do not affect I_2. So $I_1 \wedge I_2$ is indeed a global invariant of R. We do not have as yet that, upon termination of R, $I_1 \wedge I_2$ implies $\wedge_{i=1}^{n} B_i$. But it is now sufficient to show that upon termination of R the conjunction $s_1 \wedge \text{received}_1$ holds.

Now consider

$$I_3 \equiv \text{detected}_1 \rightarrow s_1 \wedge \text{received}_1.$$

It is straightforward to see that I_3 is a global invariant of R. Next, let

$$I_4 \equiv \text{forward}_1 \rightarrow s_1 \wedge \text{received}_1.$$

Then $I_3 \wedge I_4$ is a global invariant of R. Indeed, when forward_1 becomes true, initially detected_1 holds, so by virtue of I_3, $s_1 \wedge \text{received}_1$ initially holds. But $s_1 \wedge \text{received}_1$ is not affected by the execution of the transition in question.

Now we show that, upon termination of R, forward_1 holds. To this end, consider

$$I_5 \equiv \text{done}_2 \rightarrow \text{forward}_1.$$

Clearly, I_5 is a global invariant: done_2 and forward_1 become true in the same transition. Now let

$$I \equiv \bigwedge_{i=1}^{5} I_i.$$

Then I is the desired global invariant: upon termination of R, done_2 holds, and $\text{done}_2 \wedge I$ implies $\wedge_{i=1}^{n} B_i$.

PROOF OF PROPERTY 2. We now also modify processes P_i for $i \neq 1$, by introducing in each of them the auxiliary variable forward_i for the same reasons as in P_1.

The augmented versions of P_i ($i \neq 1$) have the following form:

```
P_i :: INIT_ij send_i := false ; moved_i := false ; done_i := false ;
    detected_i := false ; forward_i := false ;
    *[□ ¬ done_i ; g_i,j → moved_i := true ; S_i,j
      j ∈ Γ_i
      □ ¬ done_i ; P_i-1? s_i → send_i := true
      □ ¬ done_i ; B_i ; send_i ; P_i+1!(s_i ∧ ¬ moved_i) → send_i := false :
                                                            moved_i := false
      □ ¬ done_i ; P_i-1? end → detected_i := true ;
                                done_i := true
      □ detected_i ; P_i+1! end → forward_i := true ;
                                  detected_i := false
    ]
```

Call this augmented version of the program S. We now prove that S is deadlock-free. In the subsequent proofs it will be more convenient to consider the second premise of rule 3 in the equivalent form:

$$I \wedge \neg \text{TERMINATED} \rightarrow \neg \text{BLOCKED}.$$

Let, for $i = 1, \ldots, n$,

$$\text{TERMINATED}_i \equiv \text{done}_i \wedge \neg \text{detected}_i.$$

Note that, if in a deadlock situation of S, TERMINATED_i holds, then P_i has terminated. The following natural decomposition of $\neg$ TERMINATED allows us to carry out a case analysis.

$$\begin{aligned}\neg\ &\text{TERMINATED} \leftrightarrow \\ &[\neg\ \text{TERMINATED}_1 \wedge \forall i\ (i \neq 1 \rightarrow \text{TERMINATED}_i)] \\ \vee\ &\exists i\ (1 < i \leq n \wedge \neg\ \text{TERMINATED}_i \wedge \text{TERMINATED}_{i+1}) \\ \vee\ &\forall i \neg\ \text{TERMINATED}_i,\end{aligned}$$

where due to our convention of cyclic counting within $\{1, \ldots, n\}$, $n + 1$ is identified with 1.

Case 1. It corresponds to a deadlock situation in which P_1 did not terminate and all P_i for $i \neq 1$ have terminated. Let

$$\begin{aligned}I_6 &\equiv \neg\ \text{detected}_n \wedge \text{done}_n \rightarrow \text{forward}_n, \\ I_7 &\equiv \text{forward}_n \rightarrow \text{done}_1.\end{aligned}$$

It is straightforward to see that I_6 and I_7 are global invariants of S. Now let

$$\begin{aligned}I_8 &\equiv \text{done}_2 \rightarrow \text{forward}_1, \\ I_9 &\equiv \text{detected}_1 \rightarrow \sum_{i=1}^{n} \text{send}_i = 0, \\ I_{10} &\equiv \text{forward}_1 \rightarrow \sum_{i=1}^{n} \text{send}_i = 0, \\ I_{11} &\equiv \text{forward}_1 \rightarrow \neg\ \text{detected}_1.\end{aligned}$$

Then I_8, I_9, $I_9 \wedge I_{10}$, $I_9 \wedge I_{10} \wedge I_{11}$ are all global invariants of S. To see this, consider by way of example $I_9 \wedge I_{10} \wedge I_{11}$ under the assumption that $I_9 \wedge I_{10}$ is already shown to be a global invariant. It is obviously established by the initial assignments of S. In view of the invariance of $I_9 \wedge I_{10}$, the only transition which can falsify $I_9 \wedge I_{10} \wedge I_{11}$ is the one involving reception of the probe by P_1. But then intially send_n holds, so by I_{10} initially $\neg\ \text{forward}_1$ holds. The transition does not change the value of forward_1. So forward_1 remains false and I_{11} holds at the end of the transition. Now let

$$J = \bigwedge_{i=6}^{11} I_i.$$

J is a global invariant of S. Observe now that

$$J \wedge \text{TERMINATED}_n \rightarrow \text{done}_1$$

due to I_6 and I_7, and

$$J \wedge \text{TERMINATED}_2 \rightarrow \neg\ \text{detected}_1$$

due to I_8 and I_{11}. Thus,

$$J \wedge \text{TERMINATED}_2 \wedge \text{TERMINATED}_n \rightarrow \text{TERMINATED}_1.$$

That is,

$$J \wedge [\neg\ \text{TERMINATED}_1 \wedge \forall i\ (i \neq 1 \rightarrow \text{TERMINATED}_i)]$$

is unsatisfiable.

Case 2. This corresponds to a deadlock situation in which for some i, $1 < i \leq n$, P_i did not terminate, whereas P_{i+1} has terminated. Let some i, $1 < i \leq n$, be given. Let

$$I_{12} \equiv \text{done}_{i+1} \rightarrow \text{forward}_i,$$
$$I_{13} \equiv \text{detected}_i \rightarrow \text{done}_i,$$
$$I_{14} \equiv \text{forward}_i \rightarrow \text{done}_i \wedge \neg \text{detected}_i.$$

It is straightforward to see that I_{12}, I_{13}, and $I_{13} \wedge I_{14}$ are global invariants. Let

$$K \equiv I_{12} \wedge I_{13} \wedge I_{14}.$$

Then K is a global invariant and

$$K \wedge \text{TERMINATED}_{i+1} \rightarrow \text{TERMINATED}_i$$

results from I_{12} and I_{14}. Thus

$$K \wedge \neg \text{TERMINATED}_i \wedge \text{TERMINATED}_{i+1}$$

is unsatisfiable.

In fact, we show that neither case 1 nor case 2 can arise.

Case 3. This corresponds to a deadlock situation in which none of the processes have terminated. Let

$$I_{15} \equiv \text{done}_1 \rightarrow \text{forward}_n.$$

I_{15} is a global invariant. Also, I_{12} for all i s.t. $1 < i < n$ and $I_{13} \wedge I_{14}$ for all i s.t. $1 < i \leq n$ are global invariants. Let

$$L \equiv I_{15} \wedge \bigwedge_{i=2}^{n-1} I_{12} \wedge \bigwedge_{i=2}^{n} (I_{13} \wedge I_{14}).$$

Then L is a global invariant and

$$L \wedge \exists i\ (i \neq 2 \wedge \text{done}_i) \rightarrow \exists i\ \text{TERMINATED}_i,$$

due to I_{15}, $\wedge_{i=2}^{n-1} I_{12}$ and $\wedge_{i=2}^{n} I_{14}$. Thus,

$$L \wedge \forall i \neg \text{TERMINATED} \rightarrow \forall i\ (i \neq 2 \rightarrow \neg \text{done}_i). \tag{5}$$

Hence,

$$L \wedge \forall i \neg \text{TERMINATED}_i \wedge \text{done}_2$$
$$\rightarrow \text{detected}_2 \wedge \neg \text{done}_3 \rightarrow$$
$$\neg \text{BLOCKED, since } P_2 \text{ and } P_3 \text{ can communicate.}$$

It remains to consider the case when $\neg \text{done}_2$ holds. Let

$$I_{16} \equiv \sum_{i=1}^{n} \text{send}_i = 0 \rightarrow s_1 \wedge \text{received}_1,$$
$$I_{17} \equiv s_1 \wedge \text{received}_1 \wedge \neg \text{detected}_1 \rightarrow \text{forward}_1,$$
$$I_{18} \equiv \text{forward}_1 \rightarrow \text{done}_2.$$

Then I_{16}, I_{17}, and I_{18} are global invariants.

Let BLOCKED(P) stand for the formula BLOCKED (defined in Section 4) constructed for the original program P from Section 2. Assumption (b) of Section 2 states that whenever deadlock is reached, the global stability condition holds, and simply means that

$$\phi \equiv \text{BLOCKED}(P) \rightarrow \bigwedge_{i=1}^{n} B_i$$

is a global invariant of P. But by the form of S, ϕ is also a global invariant of S, as the added transitions do not alter the variables of P. Thus

$$M \equiv L \wedge I_{16} \wedge I_{17} \wedge I_{18} \wedge \phi$$

is a global invariant of S.

We now have

$M \wedge \forall i \neg \text{TERMINATED}_i \wedge \neg \text{done}_2 \wedge \text{BLOCKED} \rightarrow$ (by (5))
$M \wedge \forall i \neg \text{done}_i \wedge \text{BLOCKED} \rightarrow$ (by the form of S)
$M \wedge \forall i \neg \text{done}_i \wedge \text{BLOCKED} \wedge \text{BLOCKED}(P) \rightarrow$ (since ϕ is a part of M)
$M \wedge \forall i \neg \text{done}_i \wedge \text{BLOCKED} \wedge \wedge_{i=1}^{n} B_i \rightarrow$ (by the form of S)
$M \wedge \forall i \neg \text{done}_i \wedge \sum_{i=1}^{n} \text{send}_i = 0 \wedge \neg \text{detected}_1 \rightarrow$ (since I_{16}, I_{17}, and I_{18} are parts of M)
$\forall i \neg \text{done}_i \wedge \text{done}_2$

which is a contradiction. This simply means that

$$M \wedge \forall i \neg \text{TERMINATED}_i \wedge \neg \text{done}_2 \rightarrow \neg \text{BLOCKED},$$

which concludes the proof of case 3.

By rule 3, S is now deadlock-free where $J \wedge K \wedge M$ is the desired global invariant. By rule 4, P' is deadlock-free.

PROOF OF PROPERTIES 3 AND 4. As shown in Section 3, it suffices to find a global invariant I such that $\{I\}$ CONTROL PART $\{true\}$ holds in the sense of weak total correctness.

We first modify the program CONTROL PART by introducing in process P_1 an auxiliary variable count_1 which is used to count the number of times process P_1 has received the probe. Other processes remain unchanged. Thus, the processes have the following form:

```
For i = 1
P_i :: count_i := 0 ;
      *[ ¬ done_i ; B_i ; send_i ; P_{i+1}! true → send_i := false
       □ ¬ done_i ; P_{i-1}? s_i → count_i := count_{i+1} ;
       □ [s_i → detected_i := true □ ¬ s_i → send_i := true]
       □ detected_i ; P_{i+1}! end → detected_i := false
       □ ¬ done_i ; P_{i-1}? end → done_i := true
      ]
```

and for $i \neq 1$

$P_i :: *[\neg \text{done}_i ; P_{i-1}? \; s_i \rightarrow \text{send}_i := \textit{true}$
$\quad \square \neg \text{done}_i ; B_i ; \text{send}_i : P_{i+1}! \; (s_i \wedge \neg \text{moved}_i) \rightarrow$
$\qquad\qquad \text{send}_i := \textit{false} ;$
$\qquad\qquad \text{moved}_i := \textit{false}$
$\quad \square \neg \text{done}_i ; P_{i-1}? \; \textit{end} \rightarrow \text{detected}_i := \textit{true}:$
$\qquad\qquad \text{done}_i := \textit{true}$
$\quad \square \text{detected}_i ; P_{i+1}! \; \textit{end} \rightarrow \text{detected}_i := \textit{false}$
$]$

We call this program T. For the global invariant I of P', we choose the formula I_1 (i.e., $\sum_{i=1}^{n} \text{send}_i \leq 1$). We now establish a sequence of successively stronger global invariants of T (all relative to I_1). I_1 itself is clearly a global invariant of T. Let

$$I_{19} \equiv \forall i > 1 \; [\text{count}_1 = 1 \wedge \text{send}_i \rightarrow \forall j \; (1 < j < i \rightarrow \neg \text{moved}_j].$$

$I_1 \wedge I_{19}$ is clearly a global invariant of T. First, it is obviously established by the initial assignment to count_1. Second, it is preserved by every transition of T, as no moved_j variable is ever set to *true*, and whenever a send_i variable for $i > 2$ is set to *true*, then moved_{i-1} is set of *false*. Also, when count_1 becomes 1, then send_1 becomes *true* and, by invariance of I_1, no send_i for $i > 1$ can be *true*. Let

$$I_{20} \equiv \text{count}_1 = 2 \rightarrow \forall j \; (1 < j \leq n \rightarrow \neg \text{moved}_j).$$

Then, $I_1 \wedge I_{19} \wedge I_{20}$ is a global invariant : when count_1 becomes 2, then initially due to I_{19}, $\forall j \; (1 < j < i \rightarrow \neg \text{moved}_j)$ holds. In addition, at the end of the transition, $\neg \text{moved}_n$ holds. Moreover, no moved_i variable is ever set to *true*.

Let

$$I_{21} \equiv \forall i > 1 \; (\text{count}_1 = 2 \wedge \text{send}_i \rightarrow s_i).$$

Now consider $I_1 \wedge \wedge_{j=19}^{21} I_j$ and suppose that, by an execution of a transition, send_i is set to *true* when $\text{count}_1 = 2$. If $i = 2$, then s_2 holds, as s_2 is always set to *true*. So assume that $i > 2$. Then, initially, by I_{20} and I_{21}, $s_{i-1} \wedge \neg \text{moved}_{i-1}$ holds. At the end of the transition, $s_i = s_{i-1} \wedge \neg \text{moved}_{i-1}$, so s_i holds as desired.

Also, when count_1 becomes 2, then for the same reasons as in the case of I_{19}, no send_i for $i > 1$ can be *true*. This shows that $I_1 \wedge \wedge_{j=19}^{21} I_j$ is a global invariant.

Now let

$$I_{22} \equiv \text{count}_1 = 3 \rightarrow \sum_{i=1}^{n} \text{send}_i = 0.$$

Then, $I_1 \wedge \wedge_{j=19}^{22} I_j$ is a global invariant. Indeed, when at the end of a transition count_1 becomes 3, then initially on account of I_{19}, I_{20}, and I_{21}, $\text{send}_n \wedge \forall i < n \; \neg \text{send}_i \wedge s_n \wedge \neg \text{moved}_n$ holds. Thus, at the end of the transition, $s_1 \wedge \text{detected}_1 \wedge \forall i \neg \text{send}_i$ holds.

Also, $\sum_{i=1}^{n} \text{send}_i = 0$ is preserved by every transition. Finally, let

$$I_{23} \equiv \text{count}_1 \leq 3.$$

Then

$$N \equiv I_1 \wedge \bigwedge_{j=19}^{23} I_j$$

is a global invariant of T.

Indeed, when at the beginning of a transition, count_1 is 3, then on account of I_{22} no sending of the probe can take place, thus count_1 cannot be incremented. We thus showed that count_1 is bounded.

We can now prove formula (4) of Section 3. Indeed, consider the premises of rule 2 for the program T. Choose I_1 for p, I_1, the global invariant N of T, for I, and the expression

$$5n + 3 - [(n + 1)\text{count}_1 + \sum_{i=1}^{n} \text{done}_i + \text{holds(send)}]$$

for t, where holds(send) is the smallest j for which send_j holds if it exists, and 0 otherwise.

We have already shown that N is a global invariant. It is thus sufficient to show that t is always nonnegative and is decremented by each transition. But for all $b_{i,j}$ and $b_{j,i}$ mentioned in the premises of rule 2,

$$N \wedge b_{i,j} \wedge b_{j,i} \rightarrow t > 0,$$

so t is initially positive. Clearly, t is decremented by every transition and

$$N \rightarrow t \geq 0,$$

so t remains nonnegative after every transition. Thus by rule 2,

$$\{p\}T\{true\}$$

holds in the sense of weak total correctness, so by rule 4, formula (4) from Section 3 holds.

This concludes the correctness proof.

7. ASSESSMENT OF THE PROOF METHOD

The proof method proposed in Section 4 is so strikingly simple to state that it is perhaps useful to assess it and to compare it critically with other approaches to proving the correctness of CSP programs. First of all, we should explain why the introduced rules are sound.

Rules 1 and 2 are sound because the CSP programs considered in Section 4 are equivalent to a certain type of nondeterministic program. Namely, consider a CSP program P of the form introduced in Section 2. Let

$T(P) \equiv \text{INIT}_1 ; \ldots ; \text{INIT}_n ;$
$\quad *[\square_{(i,j) \in \Gamma} b_{i,j} \wedge b_{j,i} \rightarrow \text{Eff}(g_{i,j}, g_{j,i}) ; S_{i,j} ; S_{j,i}] ;$
$\quad [\text{TERMINATED} \rightarrow \text{skip}]$

where $\Gamma = \{(i, j) ; i \in \Gamma_j, j \in \Gamma_i, g_{i,j}, \text{and } g_{j,i} \text{ match}\}$.

Note that upon exit of the main loop of $T(P)$, BLOCKED holds (which does not necessarily imply TERMINATED). It is easy to see that P and $T(P)$ are

equivalent in the sense of partial correctness semantics (i.e., when divergence, failures, and deadlocks are not taken into account). Now both rules 1 and 2 exploit these equivalences.

Now consider rule 3. In a deadlock situation every process is either at the main loop entry or has terminated. Thus a global invariant holds in a deadlock situation. Moreover, the formula BLOCKED $\wedge \neg$ TERMINATED holds in a deadlock situation as well. Thus the premises of rule 3 do indeed ensure that no deadlock (relative to p) can arise.

Finally, rule 4 is sound because auxiliary variables affect neither the control flow of the program (by requirement (i)) or the values of the other variables (by requirement (ii)).

It is worthwhile to point out that the rule of auxiliary variables is not needed in the correctness proofs. This follows from two facts. First, it is not needed in the context of nondeterministic programs, as the theoretical completeness results show (see [1]). And second, due to the equivalence between P and $T(P)$ and the form of the rules, every correctness proof of $T(P)$ can be rewritten as a correctness proof of P. However, as we have seen in the previous section, this rule is very helpful in concrete correctness proofs.

It is true that the proposed proof method can only be applied to CSP programs in a normal form. On the other hand, many CSP programs exhibit this form.

Let us now relate our proof method to two other approaches to proving correctness of CSP programs—those of Apt, Francez, and De Roever [3] and of Manna and Pnueli [12].

When discussing the first approach, it is more convenient to consider its simplified and more comprehensive presentation given in [2]. The approach of [2] is based on the use of proofs from a set of assumptions. First, correctness of individual processes is proved from the assumptions about its subparts containing I/O commands (called *bracketed sections*). When proving correctness of a whole program, these assumptions are discarded when they satisfy a certain test (called a *cooperation test*).

Now consider a CSP program in the special form with all INIT_i parts being empty. Let each branch of the main loop constitute a bracketed section. Given a bracketed section, $\langle S \rangle$, associated with a branch that starts with a Boolean condition b within the text of process P_i, choose the assumption $\{b\}\ \langle S \rangle\ \{true\}$ for the proof of the $\{true\}\ P_i\ \{\text{TERMINATED}_i\}$. Then it is easy to see that

$$A_i \vdash \{true\}\ P_i\ \{\text{TERMINATED}\},$$

where A_i stands for the set of chosen assumptions and $\ldots \vdash \ldots$ denotes provability in the sense of partial correctness from a set of hypotheses. Now the premises of rule 1 are equivalent to the set of conditions stating that the chosen sets of assumptions cooperate w.r.t. the global invariant I. In their presentation, use of the communication axiom, formation rule, and arrow rule is combined.

This shows that (under the assumption that all INIT_i parts are empty) proof rule 1 can be derived in the proof system considered in [2]. This provides another, very indirect proof of its soundness.

Consider now proof rule 2. The main difference between this rule and the corresponding set of rules of [2] is that termination is proved here in a global

fashion—expression t can contain variables from various processes. To cast this reasoning into the framework of [2], one needs to consider for each process P_i a modified version of t in which variables of other processes are replaced by auxiliary variables. Once this is done, the premises of rule 2 can be reformulated appropriately and rule 2 can be derived.

Now proof rule 3 is nothing else but a succinct reformulation of the corresponding approach of [2] where the bracketed sections are chosen as above.

The way the INIT_i parts are handled is based on the observation that these program sections can be moved outside the scope of the parallel composition. In the terminology of Elrad and Francez [6], $[\mathrm{INIT}_1 \parallel \ldots \parallel \mathrm{INIT}_n]$ is a communication-closed layer of the original program.

In the approach of [2] and [3], bracketed sections can be chosen in a different way, thus shifting slightly the emphasis from global to more local reasoning (for example by reducing I_{11} to a local loop invariant). This cannot be done in the framework of the method proposed here.

Comparison with [12] can be made in a more succinct way. In [12], two types of transitions are considered in the case of CSP programs: local transitions and communication transitions. All proof rules refer to this set of transitions. When applied to CSP programs, their INV-rule becomes very similar to our rule 1. The main difference is that in our framework the only allowed transitions are those consisting of the joint execution of a pair of branches of the main loops with matching I/O guards. Such a choice of transitions does not make much sense in the framework of [12], where programs are presented in a flowchart-like form and thus have no structure. Appropriate combinations of IND and TRNS rules become from this point of view counterparts of rules 2 and 3.

The main difference lies in the way properties 3 and 4 stated in Section 2 are proved. According to the terminology of temporal logic (see, e.g., [12]), these are *liveness properties*. Using temporal logic they are proved by setting up a chain of inevitable events leading to the final one which is to hold eventually. In our approach, liveness properties are formulated as a combination of a weak total correctness statement and a deadlock freedom. This results in a different presentation of the proof.

From this discussion, it becomes clear that the proof method presented in Section 4 does not differ in essence from the approaches of [2, 3] and [12]. It simply exploits the particular form of CSP programs to which it is restricted.

ACKNOWLEDGMENTS

We would like to thank C. Delporte-Gallet and A. Pnueli for their interesting and helpful discussions on the subject of this paper. Also, we are grateful to all three referees for their detailed reports.

REFERENCES

1. APT, K. R. Ten years of Hoare's logic; a survey, part II. *Theor. Comput. Sci. 28* (1984), 83–109.
2. APT, K. R. Proving correctness of CSP programs: a tutorial. In *Proceedings of the International Summer School "Control Flow and Data Flow: Concepts of Distributed Programming,"* NATO ASI Series, vol. F14, M. Broy, Ed., Springer-Verlag, New York, 1985, 441–474.
3. APT, K. R., FRANCEZ, N., AND DE ROEVER, W. P. A proof system for Communicating Sequential Processes. *ACM Trans. Program. Lang. Syst. 2*, 3 (1980), 359–385.

4. APT, K. R., AND RICHIER, J. L. Real-time clocks versus virtual clocks. In *Proceedings of the International Summer School "Control Flow and Data Flow: Concepts of Distributed Programming,"* NATO ASI Series, vol. F14, M. Broy, Ed., Springer-Verlag, 1985, 475–503.
5. DIJKSTRA, E. W., FEIJEN, W. H., AND VAN GASTEREN, A. J. M. Derivation of a termination detection algorithm for distributed computations. *Inf. Process. Lett. 16*, 5 (1983), 217–219.
6. ELRAD, T. E., AND FRANCEZ, N. Decomposition of distributed programs into communication-closed layers. *Sci. Comput. Program. 2*, 3 (1982), 155–174.
7. FRANCEZ, N. Distributed termination. *ACM Trans. Program. Lang. Syst. 2*, 1 (1980), 42–55.
8. FRANCEZ, N., RODEH, M., AND SINTZOFF, M. Distributed termination with interval assertions. In *Proceedings of the International Colloquium on Formalization of Programming Concepts, Lecture Notes in Computer Science, 107*, (Peniscola, Spain), Springer-Verlag, New York, 1981.
9. GRUMBERG, O., FRANCEZ, N., MAKOWSKY, J., AND DE ROEVER W. P. A proof rule for fair termination of guarded commands. In *Algorithmic Languages*, J. W. de Bakker and J. C. Van Vliet, Eds., North-Holland, Amsterdam, 1981, 399–416.
10. HOARE, C. A. R. Communicating sequential processes. *Commun. ACM 21*, 8 (1978), 666–677.
11. LAMPORT, L. Time, clocks, and the ordering of events in a distributed system, *Commun. ACM 21*, 7 (1978), 558–565.
12. MANNA, Z., AND PNUELI, A. How to cook a temporal proof system for your pet language. In *Proceedings of the Symposium on Principles of Programming Languages* (Austin, Tex., 1983), ACM, New York.
13. OWICKI, S., AND GRIES, D. An axiomatic proof technique for parallel programs I. *Acta Inf. 6*, 1 (1976), 319–340.
14. TOPOR, R. W. Termination detection for distributed computations. *Inf. Process. Lett. 18*, 1 (1984), 33–36.

Received November 1984; revised December 1985; accepted December 1985.