CORRECTNESS PROOFS
OF
DISTRIBUTED TERMINATION ALGORITHMS

Krzysztof R. Apt
L.I.T.P, Université Paris 7
2, Place Jussieu, 75251 Paris, FRANCE

Abstract The problem of correctness of the solutions to the distributed termination problem of Francez [F] 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.

1. INTRODUCTION

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

A notable exception in this matter are papers of Dijkstra, Feijen and Van Gasteren [DFG] and Topor [T] 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 [DFG] are extremely helpful in understanding the final solutions presented in CSP. However, their presentation should not relieve us from providing rigorous correctness proofs of the latter ones - an issue we address in this paper.

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

This paper is organized as follows. In the next section we define the problem and propose the correctness criteria 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 to prove them. In section 5 we provide a simple solution to the problem and in the next section we give a detailed proof of its correctness. Finally, in section section 7 we assess the proposed proof method.

NATO ASI Series, Vol. F13
Logics and Models of Concurrent Systems
Edited by K. R. Apt

Throughout the paper we assume from the reader knowledge of Communicating Sequential Processes (CSP in short), as defined in Hoare [H], and some experience in the proofs of correctness of very simple loop free sequential programs.

2. DISTRIBUTED TERMINATION PROBLEM

Suppose that a CSP program

$$P \equiv [P_1 \parallel \cdots \parallel P_n],$$

where for every $1 \leq i \leq n$ $P_i :: INIT_i ; * [S_i]$ is given. We assume that each S_i is of the form $\square_{j \in \Gamma_i} g_{i,j} \rightarrow S_{i,j}$ for a multiset Γ_i and

i) each $g_{i,j}$ contains an i/o command adressing 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 with each P_i a *stability condition* B_i, a Boolean expression involving variables of P_i and possibly some auxiliary variables, is associated. By a *global stability condition* 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 whose stability conditions hold,

b) whenever deadlock takes place, the global stability condition is reached.

The *distributed termination problem* is the problem of transforming P into another program P' which eventually properly terminates whenever the global stability condition is reached.

This problem, due to Francez [F], has been extensively studied in the literature.

We say that the global stability condition is (not) reached in a computation of P' if it is (not) reached in the natural restriction of the computation to a computation of P. In turn, the global stability condition is reached (not reached) in a computation of P if it holds in a possible (no) global state of the computation. We consider here partially ordered computations in the sense of [L].

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

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' will eventually properly terminate.
4. If the global stability condition is not reached then infinitely often a statement from the original program P will be executed.

The last property excludes the situations in which the transformed parallel program endlessly executes the added control parts dealing with termination detection. We also postulate that the communication graph should not be altered.

In the abstract framework of [DFG] only the first property is proved. Second property is not meaningful as deadlock coincides there with termination. In turn, satisfaction of the third property is argued informally and the fourth one 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 $P' \equiv [P_1 \| \ldots \| P_n]$ in a normal form where for every i, $1 \leq i \leq n$

$$
\begin{array}{l}
P_i :: \mathrm{INIT}_i ; \ldots, \\
\qquad *[\ \underset{j \in \Gamma_i}{\Box} \cdots ; g_{i,j} \rightarrow \ldots ; S_{i,j} \\
\qquad\quad \Box\ \mathrm{CONTROL\ PART}_i \\
\\
\qquad]
\end{array}
$$

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. We assume that no variable of the original process $P_i :: \mathrm{INIT}_i ; *[S_i]$ can be altered in CONTROL PART$_i$ and that all i/o commands within CONTROL PART$_i$ are of new types.

We now express the introduced four properties for the case of solutions of the above form 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 $p \equiv \underline{\text{true}}$ 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 $p \equiv \underline{true}$ then we simply say that I is a *global invariant of* P.

Now, property 1 simply means that

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

holds in the sense of partial correctness.

Property 2 means that P' is deadlock free.

Property 3 cannot be expressed by refering 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 condition - is not the initial one. It is a control point which can be reached in the course of a computation.

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

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

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

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

in the sense of total correctness.

Indeed, suppose that in a computation of P' the global stability condition is reached. Then $I \wedge \bigwedge_{i=1}^{n} B_i$ holds where I is a global invariant of P'. By the 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, on virtue of (2) property 3 holds.

Consider now property 4. As before we can express it only by refering to the program CONTROL PART. Clearly property 4 holds if

$$\{ I \wedge \neg \bigwedge_{i=1}^{n} B_i \} \text{ CONTROL PART } \{\underline{true}\} \qquad (3)$$

holds in the sense of weak total correctness. Indeed, (3) guarantees that in no computation of P' the control remains from a certain moment on indefinitely within the added control parts in case the global stability condition is not reached.

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

$$\{I\} \text{ CONTROL PART } \{\underline{true}\} \qquad (4)$$

in the sense of weak total correctness.

The idea of expressing an eventuality property of one program by a termination property of another program also appears in Grumberg et al. [GFMR] in one of the clauses of a rule for fair termination.

4.PROOF METHOD

We now present a simple proof method which will allow us to handle the properties discussed in the previous section. It can be applied to CSP programs being in a normal form. So assume that $P \equiv [P_1 \parallel \ldots \parallel P_n]$ 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 $Eff(g_{i,j}, g_{j,i})$ 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 the output expression.

Finally, let

$$\text{TERMINATED} \equiv \bigwedge_{\substack{1 \le i \le n, \\ j \in \Gamma_i}} \neg b_{i,j}.$$

Observe that TERMINATED holds upon termination of P.

Consider now partial correctness. We propose the following proof rule:

RULE 1 : PARTIAL CORRECTNESS

$$\frac{\begin{array}{l}\{p\}\ INIT_1\ ;\ldots;\ INIT_n\ \{I\},\\ \{I \wedge b_{i,j} \wedge b_{j,i}\}\ Eff(g_{i,j},\ g_{j,i})\ ;\ S_{i,j}\ ;\ S_{j,i}\ \{I\}\\ \text{for all } i,j \text{ s.t. } i \in \Gamma_j,\ j \in \Gamma_i \text{ and } g_{i,j},\ g_{j,i} \text{ match}\end{array}}{\{p\}\ P\ \{I \wedge TERMINATED\}}$$

This rule has to be used in conjunction with the usual proof system for *partial* correctness of nondeterministic programs (see e.g. Apt [A1]) in order to be able to establish its premises. 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. If the premises of this rule hold then we can also deduce that I is a global invariant of P relative to p.

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

RULE 2 : WEAK TOTAL CORRECTNESS

$$\frac{\begin{array}{l}\{p\}\ INIT_1\ ;\ldots;\ INIT_n\ \{I \wedge t \geq 0\},\\ \{I \wedge b_{i,j} \wedge b_{j,i} \wedge z=t \wedge t \geq 0\}\ Eff(g_{i,j},g_{j,i});S_{i,j};S_{j,i}\{I \wedge 0 \leq t < z\}\\ \text{for all } i,j \text{ s.t. } i \in \Gamma_j,\ j \in \Gamma_i \text{ and } g_{i,j},\ g_{j,i} \text{ match}\end{array}}{\{p\}\ P\ \{I \wedge TERMINATED\}}$$

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 [A1]) in order to establish its premises. It is a usual modification of the rule concerning partial correctness.

Finally, consider deadlock freedom. Let

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

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

$$\frac{\begin{array}{l}I \text{ is a global invariant of } P \text{ relative to } p,\\ I \wedge BLOCKED \rightarrow TERMINATED\end{array}}{P \text{ is deadlock free relative to } p}$$

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

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

i) all variables from A appear in S only in assignments,
ii) no variable of S from outside of 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 $\{z\}$ is the only (nonempty) set of auxiliary variables of the program

$$[P_1 :: z:=y \; ; \; P_2 ! \, x \parallel P_2 :: P_1 ? \, u \; ; \; u:=u+1]$$

We now adopt the following proof rule first introduced by Owicki and Gries in [OG1, OG2].

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 shall use this rule both in the proofs of partial and of (weak) total correctness. Also without mentioning we shall use in proofs the well-known consequence rule which allows to strengthen the preconditions and weaken postconditions of a program.

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 [FRS] and (in an abstract setting) Dijkstra, Feijen and Van Gasteren [DFG].

We assume that the graph consisting of all communication channels within P contains a Hamiltonian cycle. In the resulting ring the neighbours of P_i are P_{i-1} and P_{i+1} where counting is done within $\{1,\dots,n\}$ clockwise.

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 introduced variables s_i, $send_i$ and $moved_i$ do not appear in the original program P :

For i = 1

```
P_i :: send_i := true ;
       *[ □   g_i,j → S_i,j
         j∈Γ_i
          □ B_i ; send_i ; P_i+1! true → send_i := false
          □ P_i-1 ? s_i → [s_i → halt □ ¬ s_i → send_i := true]
        ]
```

and for i ≠ 1

```
P_i :: send_i:=false ; moved_i:=false ;
       *[ □ g_i,j → moved_i:=true ; S_i,j
         j∈Γ_i
          □ P_i-1? s_i → send_i:=true
          □ B_i ; send_i ; P_i+1! (s_i ∧ ¬ 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 neighbour when its stability condition B_1 holds. A probe can be transmitted by a process P_i further to its right hand side neighbour 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 neighbour. 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 neighbour.

We now modify this program by arranging that P_1 sends a final termination wave through the ring once it detects the global stability condition. To this purpose we introduce in all P_i's two new Boolean variables $detected_i$ and $done_i$. The program has the following form :

For i = 1

```
P_i ::  send_i:=true ; done_i:=false ; detected_i:=false ;
       * [ □ ¬ done_i, g_i,j → S_i,j
          j∈Γ_i
           □ ¬ done_i ; B_i ; send_i ; P_i+1! true → send_i:=false
           □ ¬ done_i ; P_i-1 ? s_i →
                       [ 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 $i \neq 1$

$$
\begin{array}{lll}
P_i :: & \multicolumn{2}{l}{send_i{:=}\underline{false} \ ; \ moved_i{:=}\underline{false} \ ; \ done_i{:=}\underline{false} \ ; \ detected_i{:=}\underline{false} \ ;} \\
& \multicolumn{2}{l}{* \ [\ \underset{j \in \Gamma_i}{\Box} \neg done_i \ ; \ g_{i,j} \rightarrow moved_i{:=}\underline{true} \ ; \ S_{i,j}} \\
& \quad \Box \ \neg done_i \ ; \ P_{i-1} \ ? \ s_i \rightarrow send_i{:=}\underline{true} & \\
& \quad \Box \ \neg done_i \ ; \ B_i \ ; \ send_i \ ; \ P_{i+1}!(s_i \wedge \neg moved_i) \rightarrow & \\
& & send_i{:=}\underline{false} \ ; \\
& & moved_i{:=}\underline{false} \\
& \quad \Box \ \neg done_i \ ; \ P_{i-1} \ ? \ \underline{end} \rightarrow detected_i{:=}\underline{true} \ ; \ done_i{:=}\underline{true} & \\
& \quad \Box \ detected_i \ ; \ P_{i+1}! \ \underline{end} \rightarrow detected_i{:=}\underline{false} & \\
& \ \ ] &
\end{array}
$$

We assume that <u>end</u> 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 <u>Boolean</u>. 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 the formalized in section 3 versions of properties 1-4 from section 2.

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. $forward_1$ is used to express the fact that P_1 sent the <u>end</u> signal to P_2. Note that this fact cannot be expressed by referring to the variable $detected_1$. This refined version of P_1 has the following form :

$$
\begin{array}{lll}
P_1 :: & \multicolumn{2}{l}{send_1 := \underline{true}; \ done_1{:=}\underline{false} \ ; \ detected_1{:=}\underline{false} \ ;} \\
& \multicolumn{2}{l}{\quad received_1{:=}\underline{false} \ ; \ forward_1{:=}\underline{false} \ ;} \\
& \multicolumn{2}{l}{* \ [\ \underset{j \in \Gamma_1}{\Box} \neg done_1 \ ; \ g_{1,j} \rightarrow S_{1,j}} \\
& \quad \Box \ \neg done_1 \ ; \ B_1 \ ; \ send_1 \ ; \ P_2! \ \underline{true} \rightarrow send_1{:=}\underline{false} & \\
& \quad \Box \ \neg done_1 \ ; \ P_n?s_1 \rightarrow received_1{:=}\underline{true} \ ; & \\
& \qquad [s_1 \rightarrow detected_1{:=}\underline{true} \ \Box \ \neg s_1 \rightarrow send_1{:=}\underline{true}] & \\
& \quad \Box \ detected_1 \ ; \ P_2! \ \underline{end} \rightarrow forward_1{:=}\underline{true} \ ; & \\
& & detected_1{:=}\underline{false} \\
& \quad \Box \ \neg done_1 \ ; \ P_n \ ? \ \underline{end} \rightarrow done_1{:=}\underline{true} & \\
& \ \ ] &
\end{array}
$$

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

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 a *transition*. Here and elsewhere we occasionally identify the Boolean values false, true with 0 and 1, respectively. To avoid excessive use of brackets we assume that "$\rightarrow$" binds weaker than other connectives.

Let

$$I_1 \equiv \sum_{i=1}^{n} send_i \leqslant 1.$$

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

Consider now

$$I_2 \equiv \forall i > 1 \; [s_i \wedge send_i \rightarrow (\forall j (1 \leqslant j < i \rightarrow B_j) \vee \exists j \geqslant i \; moved_j]$$
$$\wedge \; [s_1 \wedge received_1 \rightarrow \forall j \; (1 \leqslant j \leqslant n \rightarrow B_j)].$$

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 the Boolean conditions of the guards hold.

Consider now the first conjunct of I_2. If initially for no $i > 1$ $s_i \wedge send_i$ holds then this conjunct is preserved since the transition does not alter s_i or $send_i$. Suppose now that initially for some $i > 1$ $s_i \wedge send_i$ holds. If initially also $\exists j \geqslant i \; moved_j$ holds then this conjunct is preserved. If initially $\forall j \; (1 \leqslant j < i \rightarrow B_j)$ holds then by assumption a) from section 2 at least one of the processes involved in the transition has an index $\geqslant i$. The transition sets its moved variable to true which establishes $\exists \; j \geqslant i \; moved_j$.

The second conjunct of I_2 is obviously preserved - if initially $s_1 \wedge received_1$ does not hold then it does not hold at the end of the transition either. If initially $s_1 \wedge received_1$ holds then also $\forall j \; (1 \leqslant j < n \rightarrow B_j)$ initially holds so by assumption a) from section 2 the discussed transition cannot take place.

Consider now a transition corresponding to a sending of the probe from P_i to P_{i+1} $(1 \leqslant i \leqslant n)$. Suppose that at the end of the transition $s_k \wedge send_k$ for some k $(1 < k \leqslant n)$ holds. Due to the global invariant I_1 and the form of the transition $k = i+1$. Thus in the initial state $B_i \wedge s_i \wedge \neg moved_i \wedge send_i$ holds. Now, on virtue of I_2 initially

$$\forall j \; (1 \leqslant j < i \rightarrow B_j) \vee \exists j \geqslant i \; moved_j$$

holds. Thus initially

$$\forall j \; (1 \leqslant j < i+1 \rightarrow B_j) \vee \exists j \geqslant i+1 \; moved_j$$

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.

Suppose now 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. Suppose now that initially $s_1 \wedge \text{received}_1$ does not hold. Thus the transition consists of sending the probe from P_n to P_1. Then initially $B_n \wedge s_n \wedge \neg\,\text{moved}_n \wedge \text{send}_n$ holds so on virtue of I_2 initially $\forall j\ (1 \leq j \leq n \rightarrow B_j)$ 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. Now, $I_1 \wedge I_2$ upon termination of R does not imply yet $\bigwedge_{i=1}^{n} B_i$. But it is now sufficient to show that upon termination of R $s_1 \wedge \text{received}_1$ holds.

Consider now

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

Then I_3 is clearly 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 on 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 hold. To this purpose 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.

Let now

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

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

Proof of property 2

We now also modify processes P_i for $i \neq 1$, by introducing in it the auxiliary variable $forward_i$ for the same reasons as in P_1.

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

```
Pi :: sendi:=false ; movedi:=false ; donei:=false ;
      detectedi:=false ; forwardi:=false ;
     *[ □ ¬donei ; gi,j → movedi:=true ; Si,j
        j∈Γi
        □ ¬donei ; Pi-1 ? si → sendi:=true
        □ ¬donei ; Bi ; sendi ; Pi+1!(si ∧ ¬movedi) →
                          sendi:=false ;
                          movedi:=false
        □ ¬donei ; Pi-1 ? end → detectedi:=true ;
                                donei:=true
        □ detectedi ; Pi+1! end → forwardi:=true ;
                                  detectedi:=false
      ]
```

Call this refined version of the program S. We now prove that S is deadlock free. In the subsequent proofs it will be more convenient to consider second premise of rule 3 in the form $I \wedge \neg TERMINATED \rightarrow \neg BLOCKED$. Let for $i = 1,\dots,n$

$$TERMINATED_i \equiv done_i \wedge \neg detected_i.$$

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.

$$\begin{aligned}\neg TERMINATED \equiv\ & [\neg TERMINATED_1 \wedge \forall i\ (i\neq 1 \rightarrow TERMINATED_i)] \\ & \vee \exists i\ (1 < i < n \wedge \neg TERMINATED_i \wedge TERMINATED_{i+1}) \\ & \vee \forall i\ \neg TERMINATED_i .\end{aligned}$$

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

$$I_6 \equiv \neg detected_n \wedge done_n \rightarrow forward_n,$$
$$I_7 \equiv forward_n \rightarrow done_1.$$

It is straightforward to see that I_6 and I_7 are global invariants of S. Let now

$$I_8 \equiv done_2 \rightarrow forward_1,$$
$$I_9 \equiv detected_1 \rightarrow \sum_{i=1}^{n} send_i = 0,$$

$$I_{10} \equiv \text{forward}_1 \rightarrow \sum_{i=1}^{n} \text{send}_i = 0,$$

$$I_{11} \equiv \text{forward}_1 \rightarrow \neg\, \text{detected}_1.$$

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. The only transition which can falsify $I_9 \wedge I_{10} \wedge I_{11}$ in view of invariance of $I_9 \wedge I_{10}$ is the one involving recpetion of the probe by P_1. But then initially 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.

Let now

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

J is a global invariant of S. Observe now that

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

on the account of I_6 and I_7

and

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

on the account of I_8 and I_{11}.

Thus

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

i.e.

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

is unsatisfiable.

Case 2 It corresponds to a deadlock situation in which for some i, $1 < i < n$, P_i did not terminate whereas P_{i+1} did terminate. Let some i, $1 < i < n$, 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

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

on the account of I_{12} and I_{14}.

Thus

$$K \wedge \neg \mathrm{TERMINATED}_i \wedge \mathrm{TERMINATED}_i$$

is unsatisfiable.

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

Case 3 It corresponds to a deadlock situation in which none of the processes has terminated.

Let

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

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

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

on the account of I_{15}, $\bigwedge_{i=2}^{n-1} I_{12}$ and $\bigwedge_{i=2}^{n} I_{14}$.

Thus

$$L \wedge \forall i\, \neg \mathrm{TERMINATED}_i \rightarrow \forall i\, (i \neq 2 \rightarrow \neg \mathrm{done}_i). \qquad (5)$$

Hence

$$L \wedge \forall i\, \neg\, \mathrm{TERMINATED}_i \wedge \mathrm{done}_2 \rightarrow \mathrm{detected}_2 \wedge \neg \mathrm{done}_3 \rightarrow \neg\, \mathrm{BLOCKED}.$$

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

$$I_{16} \equiv \sum_{i=0}^{n} \mathrm{send}_i = 0 \rightarrow s_1 \wedge \mathrm{received}_1,$$

$$I_{17} \equiv s_1 \wedge \mathrm{received}_1 \wedge \neg \mathrm{detected}_1 \rightarrow \mathrm{forward}_1,$$

$$I_{18} \equiv \mathrm{forward}_1 \rightarrow \mathrm{done}_2.$$

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

Let BLOCKED (P) stand for the formula BLOCKED constructed for the original program P from section 2. Assumption b) of section 2 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

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

$\forall i \neg$ done$_i \wedge$ done$_2$

which is a contradiction.

This simply means that

$M \wedge \forall i \neg$ TERMINATED$_i \wedge \neg$ done$_2 \rightarrow \neg$ 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.

<u>Proof of properties 3 and 4</u>

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 :

Let

$$I_{20} \equiv count_1 = 2 \rightarrow \forall j\ (1 < j \rightarrow \neg moved_j)$$

Then $I_1 \wedge I_{19} \wedge I_{20}$ is a global invariant : when $count_1$ becomes 2 then initially due to I_{19} $\forall j\ (1 < j < i \rightarrow \neg moved_j)$ holds. At the end of the transition additionally $\neg moved_n$ holds. Moreover, no $moved_i$ variable is ever set to <u>true</u>.

Let

$$I_{21} \equiv \forall i > 1\ (count_1 = 2 \wedge send_i \rightarrow s_i).$$

Consider now $I_1 \wedge \bigwedge_{j=19}^{21} I_j$ and suppose that by an execution of a transition $send_i$ is set to <u>true</u> when $count_1 = 2$. If $i = 2$ then s_2 holds as s_2 is always set to <u>true</u>. So assume that $i > 2$. Then initially by I_{20} and I_{21} $s_{i-1} \wedge \neg moved_{i-1}$ holds. At the end of the transition $s_i = s_{i-1} \wedge \neg moved_{i-1}$ 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 <u>true</u>. This shows that $I_1 \wedge \bigwedge_{j=19}^{21} I_j$ is a global invariant.

Let now

$$I_{22} \equiv count_1 = 3 \rightarrow \sum_{i=1}^{n} send_i = 0$$

Then $I_1 \wedge \bigwedge_{j=19}^{22} I_j$ is a global invariant. Indeed, when at the end of a transition, $count_1$ becomes 3 then initially on the account of I_{19}, I_{20} and I_{21} $send_n \wedge \forall i < n\ \neg send_i \wedge s_n \wedge \neg moved_n$ holds. Thus at the end of the transition $s_1 \wedge detected_1 \wedge \forall i\ \neg send_i$ holds.

Also $\sum_{i=1}^{n} send_i = 0$ is preserved by every transition.

Finally, let

$$I_{23} \equiv count_1 \leqslant 3.$$

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 the 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) from section 3. Indeed, consider premises of rule 2 for the program T. Choose for p I_1, for I the global invariant N of T and for t the expression

$$5\,n + 3 - [(n+1).count_1 + \sum_{i=1}^{n} done_i + holds(send)]$$

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

We already showed that N is a global invariant. It is thus sufficient to show that t is always non-negative and 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 non-negative after every transition.

Thus by rule 2

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

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

Soundness of rules 1 and 2 has to do with the fact that the CSP programs considered in section 4 are equivalent to a certain type of nondeterministic programs. Namely consider a CSP program P of the form introduced in section 2. Let

$$\begin{aligned} T(P) \equiv\ & INIT_1\ ;\ldots;\ INIT_n\ ; \\ & *[\ \mathop{\Box}_{(i,j)\in\Gamma}\ b_{i,j} \wedge b_{j,i} \rightarrow Eff(g_{i,j}, g_{j,i})\ ;\ S_{i,j}\ ;\ S_{j,i}]\ ; \\ & [TERMINATED \rightarrow skip] \end{aligned}$$

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) and "almost" in the sense of weak total correctness semantics (i.e. when deadlocks are not taken into account) as deadlocks in P translate into failures at the end of execution of $T(P)$. Now, both rules 1 and 2 exploit these equivalences.

Consider now 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 indeed ensure that no deadlock (relative to p) can arise.

Finally, as is well known, 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 [A1]). And secondly, 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 be only applied to CSP programs in a normal form. On the other hand it is easy to prove that every CSP program (without nested parallelism) can be brought into this form (see Apt and Clermont [AC]). Thus in principle this proof method can be applied to prove correctness of arbitrary CSP programs. What is perhaps more important, many CSP programs exhibit a normal form.

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

When discussing the first approach it is more convenient to consider its simplified and more comprehensive presentation given in [A2]. Consider then 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 $\{b\}\ \langle S \rangle\ \{\underline{true}\}$ for the proof of the $\{\underline{true}\}\ P_i\ \{TERMINATED_i\}$. Then it is easy to see that

$$A_i \vdash_N \{\underline{true}\}\ P_i\ \{TERMINATED\}$$

where A_i stands for the set of chosen assumptions (and according to the notation of [A2] the subscript "N" indicates a provability in the sense of partial correctness). 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. The simple form of the premises is due to the fact that 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 [A2]. 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 [A2] 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 [A2] 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, premises of rule 2 can be reformulated appropriately and rule 2 can be derived.

Now, proof rule 3 is nothing else but a succint reformulation of the corresponding approach of [A2] 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 [EF] $[INIT_1 \parallel \ldots \parallel INIT_n]$ is a communication closed layer of the original program.

In the approach of [AFR] and [A2] 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 proposed here method.

Comparison with [MP] can be made in a much more succint way. In [MP] two type 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 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 [MP] 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.

From this discussion it becomes clear that the proof method presented in section 4 does not differ in essence from the approaches of [AFR] [A2] and [MP]. It simply exploits the particular form of CSP programs to which it is restricted.

Acknowledgements We would like to thank to L. Bougé, C. Delporte-Gallet, N. Francez and A. Pnueli for interesting and helpful discussions on the subject of this paper. Also we are grateful to Mrs A. Dupont for her speedy and efficient typing of the manuscript.

REFERENCES

[A1] APT, K.R., Ten years of Hoare's logic, a survey, part II, Theoretical Computer Science 28, pp. 83-109, 1984.

[A2] APT, K.R., Proving correctness of CSP programs, a tutorial, Tech. Report 84-24, LITP, Université Paris 7, 1984 (also to appear in the Proc. International Summer School "Control Flow and Data Flow : Concepts of Distributed Programming", Marktobedorf, 1984).

[AC] APT, K.R. and CLERMONT Ph., Two normal form theorems for CSP programs, in preparation.

[AFR] APT, K.R., FRANCEZ N. and DE ROEVER, W.P., A proof system for Communicating Sequential Processes, ACM TOPLAS 2 No 3, pp. 359-385, 1980.

[AR] APT, K.R. and RICHIER, J.L., Real time clocks versus virtual clocks, Tech. Report 84-34, LITP, Université Paris 7, 1984, (also to appear in the Proc. International Summer School "Control Flow and Data Flow : Concepts of Distributed Programming", Marktobedorf, 1984).

[DFG] DIJKSTRA, E.W., FEIJEN, W.H. and van GASTEREN, A.J.M., Derivation of a termination detection algorithm for distributed computations, Inform. Processing Letters 16, 5, pp. 217-219, 1983.

[EF] ELRAD, T.E. and FRANCEZ, N., Decomposition of distributed programs into communication closed layers, Science of Computer Programming 2, No 3, pp. 155-174, 1982.

[F] FRANCEZ, N., Distributed termination, ACM TOPLAS 2, No 1, pp. 42-55, 1980.

[FRS] FRANCEZ , N., RODEH, M. and SINTZOFF, M., Distributed termination with interval assertions, in : Proc. Int Colloq. Formalization of Programming Concepts, Peniscola, Spain, Lecture Notes in Comp. Science, vol. 107, 1981.

[GFMR] GRUMBERG, O., FRANCEZ N., MAKOWSKY J., and DE ROEVER W.P., A proof rule for fair termination of guarded commands, in : J.W. de Bakker and J.C. Van Vhet eds., Algorithmic languages, IFIP, North Holland, Amsterdam, pp. 399-416, 1981.

[H] HOARE, C.A.R., Communicating sequential processes, CACM 21, 8, pp. 666-677, 1978.

[L] LAMPORT, L., Time, clocks and the ordering of events in a distributed system, CACM 21, 7, pp. 558-565, 1978.

[MP] MANNA, Z. and PNUELI, A., How to cook a temporal proof system for your pet language, in : Proc. of the Symposium on Principles of Programming Languages, Austin, Texas, 1983.

[T] TOPOR, R.W., Termination detection for distributed computations, Inform. Processing Letters 18, 1, pp. 33-36, 1984.