ON THE CORRECTNESS OF A TERMINATION DETECTION ALGORITHM

Devendra Kumar

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188

TR-87-08 March 1987

Abstract

We show that the termination detection algorithm for distributed computations proposed in [Arora 86] is incorrect. We identify specific issues that have not been adequately addressed in the above algorithm, leading to its incorrect operation.

Keywords: Distributed program, distributed termination detection, message communication.

Table of Contents

1. Introduction

In section 2, we present counter examples to show that the distributed termination detection algorithm of Arora et. al. [Arora 86] may detect "false termination". We also discuss the underlying reasons for its failure. We point out the error in the proof of correctness of the algorithm as presented in the above paper. Then we discuss methods to make the algorithm correct. In section 3, we discuss situations where the algorithm fails to detect "true termination". Again, we point out the error in the proof of correctness as given in the above paper, and discuss how to correct this aspect of the algorithm.

We give our counter examples for several different assumptions regarding the nature of message communication. (These assumptions are those usually made in the literature.) The reason for considering different assumptions is that the above paper leaves its assumptions regarding message communication ambiguous. In particular, is it assumed to be synchronous, as in CSP [Hoare 78], or asynchronous? If asynchronous, are the messages on a given channel received in the same order as they were sent (referred to as the first-in-first-out, or FIFO, property of the communication channel)? This ambiguity in the paper is evident from the following. In section 2 of the discussed paper, it seems that the algorithm assumes synchronous communication, because the definition of C (the conjunction of local predicates c_i of the processes i) is not very meaningful if asynchronous communication is assumed. Obviously, under asynchronous message communication, C cannot take into account the states of the channels. On the other hand, in the proof of assertion (1) in section 3.2, the messages "in transit" are considered. This assumes asynchronous message communication. Moreover, in the algorithm description, deadlock situations may arise if synchronous communication is assumed, e.g., if all processes are trying to send I-am-passive messages.

Our counter examples show that the algorithm does not work under any of the above assumptions. In particular, in all our examples, the sequence of events satisfies the FIFO property for the channels. Thus any such counter example remains valid whether or not the algorithm, in case of asynchronous message communication, assumes FIFO property for communication channels.

The reader is assumed to be familiar with the details of the proposed algorithm [Arora 86].

In the following discussion we use i to refer to process p_i. A communication channel from process i to j is denoted by the ordered pair (i,j). Also, we write $state_i(i)$ to refer to the variable state(i) at process i.

In examples 1,2,3, and 4 below, we consider a network of N processes 0,1,...,(N-1), where the value of N will be specified in each example. For any process i, communication channels between processes i and [(i+1) mod N], in each direction, exist. Additional channels may be defined in an example. (Thus, each process has at least four adjacent channels.) The ring R for probe messages is defined by the successor function, successor(i) = (i+1) mod N for all i=0,1,...,(N-1). One may assume either synchronous or asynchronous message communication, unless otherwise specified. Initially all processes are assumed to be active.

2. Counter Examples: Detecting False Termination

For the sake of clarity, in a history of events in this section, we omit histories of all but one specific probe b, identified in the history. The shown history may not be completed, since termination may be declared earlier due to some other probe.

Example 1 below shows the following problem in the proposed algorithm. It is possible to have $state_i(j)$ = passive for all i,j, and yet there is a basic message m in transit on a channel that is not on the unidirectional ring R on which the probes travel. In such a situation, a probe may make a complete round of the ring R while the message m remains in transit. Thus a "false termination" would be detected.

Example 1: Let N be 4. Assume channels (1,3) and (3,1) also exist. Assume asynchronous message communication. Consider the following scenario.

1. Process 3 sends a basic message m to process 1.
2. All processes become passive, and send I-am-passive messages to all their neighbors.
3. All the above messages are received, except for those on the channel (3,1).
   At this point, $state_i(j)$ = passive for all i,j, except that $state_1(3)$ = active.
4. Process 1 receives m, becomes active, and sends a basic message m' to process

3.

5. Process 1 receives I-am-passive message from process 3.

   At this point, $state_i(j)$ = passive for all i,j, except that $state_1(1)$ = active.

6. Process 1 becomes passive and sends I-am-passive messages to all its neighbors.

7. All these I-am-passive messages are received, except for the one from process 1 to process 3.

   At this point, $state_i(j)$ = passive for all i,j, and the channel (1,3) contains the message m' followed by an I-am-passive message.

8. A probe message b is initiated by process 0.

9. Probe b makes a complete round of the ring R, and thus termination is declared (unless it is declared before, due to some other probe message).

   [Note: the two messages in the channel (1,3) remain in transit.]

Examples 2 and 3 below illustrate the following situation. When a probe message b arrives at a process i, it is possible that $state_i(j)$ = passive for all j; however, i may have previously sent a basic message m to some process k, and k received m after it propagated b. Thus b is unaware of the behind-the-back computation resulting from the reception of m at k. Example 3 is more involved than example 2; but it is included here because it drives home the point more clearly.

Example 2: Let N be 8. Consider the following sequence of events.

1. All processes other than 6 become passive, and send I-am-passive messages to their neighbors. All these I-am-passive messages are received.

2. Process 0 sends probe message b. The probe b is propagated by processes 1,2,3, and is finally received at process 4.

3. Process 6 sends a basic message to 7, which is received by 7, making it active.

   Process 7 similarly makes process 0 active.

Process 0 similarly makes process 1 active.

Process 1 similarly makes process 2 active.

Processes 6,7,0,and 1 now become idle. They send I-am-passive messages to their neighbors. All these I-am-passive messages are received.

At this point $state_i(j)$ = passive for all i, j such that [(j ≠ 2) and {(i=j) or (i and j are neighbors)}].

4. Probe b travels from process 4 to 5 to 6 to 7 to 0. Termination is declared (unless it is declared before, due to some other probe message). But process 2 is still active!

Example 3: Consider any N > 3.

Below we define a specific probe message b and show a history of events where b continuously traverses along ring R without ever getting purged. Obviously, termination will be declared in a finite time. However, in this sequence of events, the system does not terminate. Variables I and J below are auxiliary variables used to facilitate our description of the history of events. As before, events corresponding to probe messages other than b are not shown.

1. I := 0;

   if even(N) then J := N/2 else J := (N-1)/2;

   All processes other than J become passive. They send I-am-passive messages to their neighbors. All these I-am-passive messages are received.

   Process I is ready to send a probe message b on the ring R.

2. At this point, invariant A, defined as the conjunction of the following clauses, holds.

   a. I and J refer to processes, i.e., they are in the set {0,...,(N-1)}.

   b. Processes I and J are not neighbors.

   c. Process J is active.

   d. $state_i(j)$ = passive for all i,j such that [(j ≠ J) and {(i=j) or (i and j are

neighbors)}].

/* Note: In particular, every process other than J is passive. */

e. No basic or I-am-passive messages are in transit.

f. The probe b is at process I.

Now the following events take place.

J sends a basic message to successor(J). Then J becomes passive, and sends I-am-passive messages to its two neighbors. All these three messages are received. Successor(J) becomes active.

I sends the probe message b to successor(I). This message is received by successor(I).

At this point we redefine the auxiliary variables I,J by I := successor(I) and J := successor(J).

Now the invariant A again holds.

3. The sequence of events in step 2 above is repeated, until termination is declared, when b or another probe completes a round of ring R.

Error In The Proof: The following statement in proof of assertion (3), on page 314 of the paper, is incorrect: "This means that process p_k must purge the probe-message upon receiving it." It is possible that before receiving the probe message, p_k becomes passive and has $state_i(j)$ = passive for all its neighbors j.

Methods Of Correction

Several approaches have appeared in the literature to deal with the problem of transient messages illustrated in example 1. For example, [Misra 83] proposes flushing out the transient messages by requiring the probe message to go through every channel of the network (with the assumption of FIFO property). [Chandy 85a] suggests receiving an acknowledgment message for every basic message sent. [Kumar 85] suggests a method using the counts of number of basic messages sent and the number of basic messages received in the system.

The problem of behind-the-back computation, as illustrated by examples 2 and 3, is usually solved by requiring the probe to traverse the ring R more than once, and declaring termination only when in the current round of the ring R, the probe finds that each process i has been "continuously passive" ever since its last visit at i. For details, we refer the reader to [Misra 83, Chandy 85a, Kumar 85].

3. Counter Example: Failing To Detect True Termination

Note that a passive process does not necessarily become active on receiving a basic message. For example, for some i, suppose c_i (the condition for process i to be passive) is defined to be: [process i has received at least one basic message]. Then, after receiving its first basic message, process i becomes passive. On receiving its second basic message, it *remains* passive. Therefore at this point i will not send any I-am-passive messages (since it did not *become* passive). This may result in a termination state remaining undetected forever, as exemplified below.

Example 4: Let N be 2. Consider the following sequence of events.

1. Process 1 become passive. It sends an I-am-passive message to 0. Process 0 receives this message.
2. Process 0 sends a basic message m to process 1. At this point $state_0(1)$ = active.
3. Process 1 receives m, but *remains* passive.
   /* Hence it does not send an I-am-passive message. */
4. Process 0 becomes passive. It sends an I-am-passive message to process 1. Process 1 receives this message. Process 1 sends a probe message to 0.
5. Process 0 receives the probe message, and purges it since $state_0(1)$ = active.
6. No further events take place in the system; hence termination is never detected. But the system is terminated!

Error In The Proof: In proof of assertion (1) on page 313 of the paper, the following statement is incorrect: "Upon receipt of the last I-am-passive message from one of its

neighbours, p_j will thus qualify for initiation of a probe-message." Process p_j may have sent a basic message m to some p_k after receiving an I-am-passive message from p_k, and p_k may *remain* passive. Thus p_j would, in future, always assume that p_k is active.

Also, as illustrated by example 4, the following statement in proof of assertion (2), page 313 is incorrect: "Thus, the probe-message of p would be forwarded by each process and hence would reach back to p."

Methods Of Correction

To correct such a situation, one may take any of the following two standpoints: (a) redefine the termination detection problem so that a passive process on receiving a basic message always becomes active, at least momentarily, or (b) on reception of a basic message, if a previously passive process remains passive, then it sends out I-am-passive messages to all its neighbors.

A problem similar to example 4 may arise if a process is initially passive. To avoid this, one may require that (a) initially every process is active, or (b) initially if a process is passive, it sends out I-am-passive messages to all its neighbors.

4. Conclusions

We showed that the termination detection algorithm proposed in [Arora 86] is incorrect – sometimes it detects false termination, and sometimes it fails to detect a true termination. We identified specific issues leading to the incorrect operation. We mentioned methods to correct the algorithm with regard to these issues.

References

[Arora 86] R. K. Arora, S.P. Rana, and M. N. Gupta, "Distributed Termination Detection Algorithm For Distributed Computations", *Information Processing Letters*, vol. 22, no. 6, pp. 311-314, 1986.

[Chandy 85a] K. M. Chandy and J. Misra, "A Paradigm for Detecting Quiescent Properties in Distributed Computations", working paper, Department of Computer Sciences,

University of Texas, Austin, Texas 78712, January 9, 1985.

[Chandy 85b] K. M. Chandy and L. Lamport, "Distributed Snapshots: Determining Global States of Distributed Systems", *ACM Transactions on Computing Systems*, vol. 3, no. 1, February 1985, pp.63-75.

[Dijkstra 80] E. W. Dijkstra and C. S. Scholten, "Termination Detection for Diffusing Computations", *Information Processing Letters*, Vol. 11, No. 1, August 1980.

[Francez 80] N. Francez, "Distributed Termination", *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 1, pp. 42-55, January 1980.

[Francez 81] N. Francez, M. Rodeh, and M. Sintzoff, "Distributed Termination with Interval Assertions", *Proceedings of Formalization of Programming Concepts*, Peninsula, Spain, April 1981. Lecture Notes in Computer Science 107, (Springer-Verlag).

[Francez 82] N. Francez and M. Rodeh, "Achieving Distributed Termination Without Freezing", *IEEE-TSE*, Vol. SE-8, No. 3, pp.287-292, May 1982.

[Gligor 80] V. Gligor and S. Shattuck, "On Deadlock Detection in Distributed Data Bases", *IEEE-TSE*, Vol. SE-6, No. 5, September 1980.

[Gouda 81] M. Gouda, "Distributed State Exploration For Protocol Validation", Technical Report TR 185, Department of Computer Sciences, University of Texas, Austin, October 1981.

[Hoare 78] C. A. R. Hoare, "Communicating Sequential Processes", *Communications of the ACM*, Vol. 21, No. 8, pp. 666-677, August 1978.

[Kumar 85] D. Kumar, "A Class of Termination Detection Algorithms For Distributed Computations", *Fifth Conference on Foundations of Software Technology & Theoretical Computer Science*, New Delhi, India, December 16-18, 1985.

[Lamport 78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", *Communications of the ACM*, Vol. 21, No. 7, July 1978.

[Misra 83] J. Misra, "Detecting Termination of Distributed Computations Using Markers", *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Montreal Canada, August 17-19, 1983.