Übungen (3)

1) Man zeige, dass der Algorithmus von Arora et al.
("Distributed termination detection algorithm for distributed computations", Information Processing Letters, Vol 22 No 6, pp. 311-314, 1986) fehlerhaft ist. Dazu

a) gebe man ein möglichst einfaches Gegenbeispiel an,
b) lokalisiere man den Fehler im Korrektheitsbeweis,
c) identifiziere man den eigentlichen Denkfehler der Autoren.

2) Wenn man davon ausgehen kann, dass sich Nachrichten nicht überholen (FIFO-Kanäle), lässt sich das "flushing-" oder "channel-sweeping-Prinzip" anwenden: Um sicherzustellen, dass auf einem Kanal keine Basisnachricht mehr unterwegs ist (oder: unterwegs war?), schiebt man mit einer Kontrollnachricht eventuelle Basisnachrichten aus dem Kanal heraus. Man setze diese Idee in einen Algorithmus zur Feststellung der verteilten Terminierung um, und zwar:
a) für Modelle mit aktiven und passiven Prozessen, wo Nachrichten beliebig lange unterwegs sind, die Nachrichtenkanäle jedoch die FIFO-Eigenschaft besitzen;
b) für das allgemeine Modell (wo die Kanäle nicht unbedingt FIFO sind), indem man die gefundene Lösung entsprechend adaptiert (ohne die FIFO-Eigenschaft für die Basiskommunikation zu erzwingen!)

3) In der Vorlesung wurden drei Berechnungsmodelle (nachrichtengesteuertes Modell, Atommodell, Synchronmodell) vorgestellt, für die man das Terminierungsproblem lösen kann. Man zeige für jedes der Modelle, wie man eine Lösung in diesem Modell als Lösung für eines der anderen Modelle verwenden kann bzw. in eine entsprechende Lösung in systematischer Weise transformieren kann.

Information Processing Letters 22 (1986) 311–314
North-Holland

30 May 1986

DISTRIBUTED TERMINATION DETECTION ALGORITHM FOR DISTRIBUTED COMPUTATIONS

R.K. ARORA, S.P. RANA and M.N. GUPTA

Department of Computer Science and Engineering, Indian Institute of Technology, Hauz Khas, New Delhi 110016, India

Communicated by W.L. Van der Poel
Received 4 July 1985

A fully distributed and symmetric algorithm for solving the distributed termination problem is presented along with its correctness arguments. The algorithm does not make use of time-stamps and clock-synchronization and is very simple.

Keywords: Distributed program, distributed termination detection, Hamiltonian ring structure, message communication

1. Introduction

The distributed termination problem requires taking of explicit or implicit snapshots of the state of distributed processes and then testing a termination criterion over the snapshot. The majority of the algorithms in the literature [3,4,5,6,7,11], relegates this responsibility to a unique process in the system. Because of its specialized role, the unique process becomes a point of centralized control and hence a performance bottleneck. In this paper we are interested in fully distributed algorithms, where all processes have identical code added to them for the purpose of solving the termination problem.

Few algorithms [2,10] do belong to the above category. However, we further depart from these and we avoid the use of time-stamps and clock-synchronization.

After introducing the termination problem in Section 2, a new algorithm is developed with the correctness arguments in Section 3. The last section concludes with the comments on the performance aspects of the presented algorithm.

2. Distributed termination problem

Consider a distributed program P consisting of n communicating sequential processes $p_1, p_2, \ldots, p_n$. Let a local predicate c_i be associated with process p_i, $1 \leqslant i \leqslant n$. Let C be the conjunction of local predicates c_i where the values of the c_i's correspond to the same time instant for all the processes. We refer to C as the distributed termination condition. The necessary condition for a process p_i $(1 \leqslant i \leqslant n)$ to terminate is the truth of c_i. However, it can only be terminated after asserting the truth of C.

When c_i is true, $1 \leqslant i \leqslant n$, the corresponding process p_i is said to be in *passive* state, otherwise it is in *active* state. An active process may change the state of a passive process by engaging into basic communication with that process. The basic communication refers to the communication inherent in the distributed program P. In addition to basic communication, control communication, which takes place around the Hamiltonian ring in anti-clockwise direction, is superimposed for the purpose of detecting distributed termination.

A passive process never initiates a basic communication; however, a process in any state can initiate or engage in control communication.

ACHTUNG: Algorithmus ist falsch!

An active process p_i, $1 \leqslant i \leqslant n$, may engage in basic communication only with the processes belonging to the set N_i, referred to as the set of neighbours of p_i.

3. Detection of distributed termination

Algorithms for termination of distributed programs primarily consist of two distinct phases viz. a detection phase followed by a termination phase. In the detection phase, the truth of the distributed termination condition is asserted, whereas the termination phase actually terminates the program.

First, we focus our attention on the detection phase. In the next section, comments on the termination phase will be provided.

3.1. Detection phase

The detection phase employs control—messages are referred to as *probe-messages*. The processes are assumed to have unique identifications. A probe-message always carries the identification of its initiator process.

In the algorithm below, we allow probe-messages to propagate along a unidirectional ring, linking the n processes. The underlying strategy in the presented algorithm is briefly stated as follows: "Upon satisfaction of a local condition, a process sends a probe-message with its identification to a successor process. Whenever a process receives a probe-message, it again makes a check and depending upon this check it either forwards the probe-message or purges it. However, if the probe-message received by a process is its own message, the last process concludes the truth of the distributed termination condition and thus enters into termination phase."

The algorithm is elaborated by answering the following questions:

- What local information is to be satisfied so as to initiate a probe message?
- How is a response to an incoming probe message generated by a process?

In the present algorithm, each process is required to maintain the following information:

(1) its own state information (active or passive), and

(2) the information about the state of its neighbours.

This information is maintained by modifying the code in the following manner: Whenever a process sends a message to a neighbouring node, it records the state of the neighbouring node as active. Whenever a node becomes passive, it sends an I-am-passive message to all its neighbours. As soon as a process receives an I-am-passive message from a process, it records the state of the last process as passive, or remains passive, if it was already passive.

The algorithm is fully described as follows.

Algorithm for process p_i ($1 \leqslant i \leqslant n$)

1. *Upon becoming passive*:
   state(p_i) := passive;
   / * state(p_i) has been used for recording the status of p_i * /
   for each $p \in$ {neighbours of p_i} **do**
   send I-am-passive to p;
2. *Upon receiving an I-am-passive message from* p ($p \in$ {neighbours of p_i}):
   begin
   state$_i$(p) := passive;
   / * state$_i$(p) is used for recording the status of process p in p_i * /
   if state$_i$(p_j) = passive for all $p_j \in$ {neighbours of p_i}
   and state(p_i) = passive
   then send a probe-message to successor(p_i)
   / * successor(p_i) is used for referring successor of p_i on the ring * /
   end
3. *Upon receiving a probe-message initiated by a process other than* p_i:
   begin
   if state$_i$(p_j) = passive for al $p_j \in$ {neighbour of p_i}
   and state(p_i) = passive
   then forward the probe-message to successor(p_i)
   else purge the above probe-message
4. *Upon receiving a probe-message initiated by* p_i *itself*:
   enter in termination phase;

ACHTUNG: Algorithmus ist falsch!