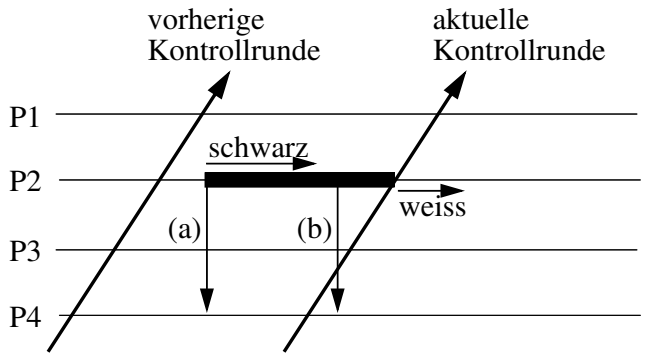
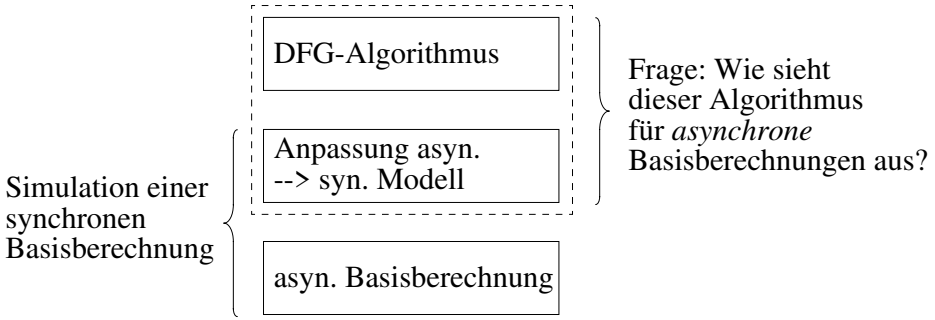
DFG-Algorithmus - "falscher Alarm"



- Die beiden Situationen (a) und (b) werden nicht unterschieden, obwohl nur (b) gefährlich ist
- Konsequenz: Wenn im Gebiet zwischen den beiden Runden eine Nachricht an einen höheren Prozess gesendet wird, ist in jedem Fall noch eine weitere Runde nötig
- Vereinfachung von Regel 1 (Konsequenz?)

Regel 1': Ein Prozess, der eine Basisnachricht sendet, wird schwarz

---

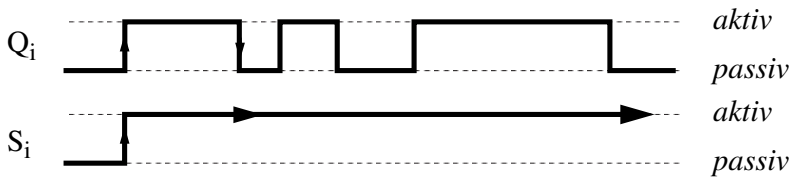


Sticky-flags-Methode

- Voraussetzung: Synchrone Kommunikation
  - Terminierungskriterium: "Sind alle Prozesse (gleichzeitig) passiv?"
- Q_i: interner Zustand {aktiv, passiv}
  (aus der Basisberechnung)
- S_i: "Sticky flag" {aktiv, passiv}
  (zusätzlich vom Kontrollalgorithmus)

} für jeden Prozess

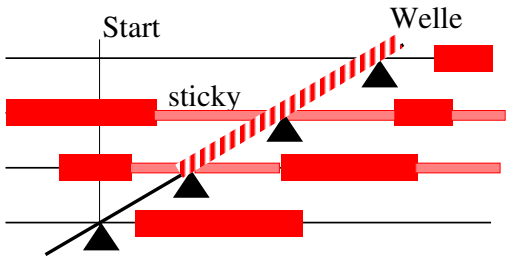
- "Sind alle Q_i passiv?" entlang einer Welle --> falsch!
- Idee: "Sticky flag" S_i statt dessen betrachten!



- S_i = aktiv wenn P_i aktiv ist / wird
- S_i *bleibt* aktiv, wenn P_i passiv wird

- Terminierung melden, wenn alle S_i passiv sind

nicht: Q_i!



Safety und liveness

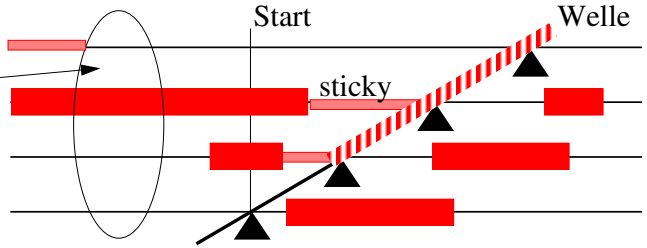
Beh.: Wenn ein P_j bei Start der Kontrollwelle aktiv war
==> Terminierung wird nicht gemeldet

<==> Terminierung wird gemeldet ==>
alle Prozesse passiv bei Start der Welle

(safety)

- Safety gilt, aber was ist mit liveness?

Setze $S_i := Q_i$ für alle Prozesse (nicht notwendig gleichzeitig!)



- Lösung:
  - Setze zunächst S_i auf "korrekten" Wert
  - Starte *dann* Welle
  - Bem.: Safety bleibt erhalten!

- Also:
  - 1. Welle setzt S_i auf momentanen Wert
  - 2. Welle fragt danach den Wert ab

2. Welle, 1. Welle

- Kombinieren zu einer einzigen Welle (Vorder- / Rückflanke)

- Variante: Kombinierte Welle besucht nur passive Prozesse
  --> "momentaner Wert Q_i" ist stets passiv
  --> sticky flag S_i wird dabei stets zurückgesetzt auf "passiv"

Sticky flags: formalere Spezifikation

Aktion, die bei *Besuch des Tokens* TOK ausgeführt wird:

```
{Q_i = passive}
receive <TOK>;
if S_i = active then TOK := active fi;
if i = n  and  TOK = passive  then " termination!"
                              else if i ≠ n then send <TOK> to P_{i+1}
                                            else send <passive> to P_1
                                   fi;
                                   S_i := passive
fi
```

ist "echter" Wert Q_i des Prozesses zu diesem Zeitpunkt ("Rückflanke" der Welle)

Aktion, die bei *Empfang einer Nachricht* ausgeführt wird:

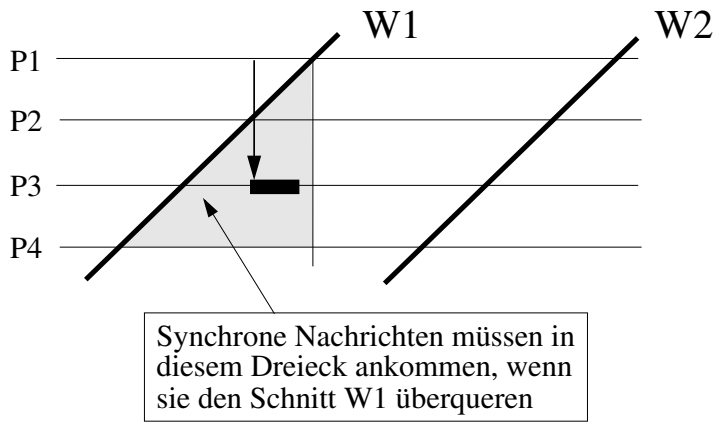
```
receive <...>;
S_i := active
```

da in diesem Moment Q_i *active* ist

Bei dem "sticky flag" S_i handelt es sich also eigentlich um ein *Empfangsflag* für Basisnachrichten, das bei Besuch der Welle wieder zurückgesetzt wird

Man vgl. das mit dem *Sendeflag* des DFG-Algorithmus
(was ist besser?)

Empfangsflags bei syn. Kommunikation



Idee: Feststellen, *ob* im Dreieck eine Nachricht ankommt

- W1 *schärft* ein Empfangsflag (d.h. setzt es zurück)
- Empfang einer Basisnachricht *setzt* das flag
- W2 (gestartet nach Ende von W1) *prüft*, ob ein flag gesetzt wurde

Wenn

einfach realisierbar!

- W1 keine aktiven Prozesse "durchtrennt" hat
- W2 kein gesetztes flag feststellt

dann terminiert (nach Ende von W1 spätestens)

Rolle von W1 und W2 kann zusammengefasst werden

- kombinierte Welle testet erst und setzt dann das flag zurück

---

Denkübungen:

- Exakter Beweis? (Ohne mit "senkrechten" Pfeilen die Geometrie zu bemühen)
- Prinzip auf asynchrone Kommunikation übertragen? (Modelltransformation)

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.

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;

3.2. Termination phase

To finally terminate the process of the distributed program P, let each process make use of Boolean variables SENDTM and RCVTM with initial value ***false***. For a process, SENDTM is set to ***true*** upon its forwarding the termination-message to the successor while RCVTM is set to ***true*** upon its receiving a termination-message from the predecessor.

Based upon the above, the steps to be taken in terminating the processes are briefly given as follows:

1. *Upon determining the distributed termination condition by a process* p_i:
   begin
   SENDTM := ***true***;
   send termination-message to successor(p_i)
   end
2. *Upon receipt of termination-message by* p_i:
   begin
   RCVTM := ***true***;
   if SENDTM **then terminate**
   else begin
   SENDTM := ***true***;
   send termination-message to successor(p_i);
   terminate
   end
   end

Correctness of the algorithm

To establish the correctness of the above algorithm so as to ensure that the false termination condition is not detected and any deadlock situation does not arise, we need to state and prove the following assertions:

(1) There is at least one process that succeeds in initiating the probe-message.

(2) There is at least one process that gets its probe-message back, i.e., it succeeds in entering the termination phase by detecting the truth of the distributed termination condition.

(3) No process can enter the termination phase without the distributed termination condition being ***true***.

Proof of assertion (1)

Consider a process, say p, that is last to become passive. As p belongs to the set the numbers of which are neighbours of p, all the neighbours of p will qualify to initiate probe-messages.

Further, consider any neighbour p_j of p. Since p is the last process to become passive, p_j and all neighbours of p_j are already passive. Thus, p_j has either received all I-am-passive messages except from p when p becomes passive or I-am-passive messages are in transit. Eventually, p_j will receive all I-am-passive messages. 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. Hence, the assertion follows.

Proof of assertion (2)

Once the distributed termination condition has become ***true***, all processes are in passive state and only I-am-passive messages may be in transit. Such messages will eventually be delivered causing one or more processes to initiate probe-messages. Let p be the process among the above processes to initiate the probe-message last. Obviously, when p initiates the probe-message, all processes would be passive, and p has received all I-am-passive messages. Thus, the probe-message of p would be forwarded by each process and hence would reach back to p. Thus, there is at least one process in the system getting its probe-message back.

Proof of assertion (3)

The proof follows by contradiction. Consider a probe-message initiated by a process, say p. Let p get its probe-message back and hence conclude the truth of the distributed termination condition. Further, assume that there is an active process, say p_j, thus violating the above conclusion.

Obviously, p_j has become active after forwarding the probe-message of p. In other words, some active process p_i has communicated with p_j after p_j's forwarding the probe-message of p. There are two cases. Either p_j has not yet received the probe-message of p or has already forwarded it. If p_j has already forwarded it, this means that p_j in turn has been activated by some other process. Continuing the argument in this manner, we can conclude that there is some process, say p_k, that before receiving the probe-message of p communicates with a process that has already forwarded the probe-message of p. This means that process p_k must purge the probe-message upon receiving it. But, according to our assumption, all processes forward the probe-message of p, hence a contradiction.

ACHTUNG: Algorithmus ist falsch!

Volume 22, Number 6 INFORMATION PROCESSING LETTERS 30 May 1986

4. Concluding remarks

We have presented a fully distributed and symmetric algorithm for solving the distributed termination problem that does not make use of time-stamp and clock-synchronization. We have looked into three aspects:

- initiation of probe-messages to detect the distributed termination condition,
- detection phase,
- termination phase.

The issuance of probe-message from a process has been based upon the concept of neighbouring processes, i.e., that set of processes with which the process enters into basic communication for solving the distributed termination problem. The probe-message is issued from a process only when itself and all its neighbours have become passive. This results in considerably less control message traffic as compared to other reported approaches [2,10].

The detection phase is based upon two factors:

- maintenance of local information so as to initiate or respond to probe-messages,
- propagation of probe-message.

The local information in terms of keeping the status of neighbours of a process in its control section is automatically generated when basic communication takes place between the process and its neighbours.

The probe-message once issued from a process propagates around the ring in anti-clockwise direction and either reaches back its originating process thereby determining the truth of the distributed termination condition or gets purged on its way on encountering either an active process or a passive process with at least one of its neighbours in active state. Finally, upon determining the truth of the distributed termination condition, the termination phase has been discussed where a process gets terminated only after it has both received and forwarded a termination-message issued by a process or processes.

Thus, we find that our presented algorithm is simple and involves less message overheads.

References

[1] K.R. Apt and J.I.. Richier, Real time clocks versus virtual clocks, Tech. Rept. #84-34. LITP, University of Paris 7, 1984.

[2] R.K. Arora and N.K. Sharma, A methodology to solve distributed termination problem, Information Systems 8 (1) (1983) 37–39.

[3] K.M. Chandy and J. Misra, Termination detection of diffusing computations in communicating sequential processes, Tech. Rept. TR#144, The University of Texas at Austin, 1980; also: ACM-TOPLAS 4 (1) (1982) 37–43.

[4] E.W. Dijkstra and C.S. Scholten, Termination detection for diffusing computations, Inform. Process. Lett. 11 (1) (1980) 1–4.

[5] N. Francez, Distributed termination, ACM-TOPLAS 2 (1) (1980) 42–45.

[6] N. Francez and M. Rodeh, Achieving distributed termination without freezing, Tech. Rept. TR#72, IBM Israel Scientific Center, 1979.

[7] N. Francez, M. Rodeh and M. Sintzoff, Distributed termination with interval assertions, Tech. Rept. TR#186, Computer Science Dept., TECHNION—Israel Institute of Technology, 1980.

[8] C.A.R. Hoare, Communicating sequential processes, Comm. ACM 21 (8) (1978) 666–777.

[9] L. Lamport, Time clocks and ordering of events in a distributed system, Comm. ACM 21 (7) (1978) 558–565.

[10] S.P. Ranu, A distributed solution of the distributed termination problem, Inform. Process. Lett. 17 (1983) 43–46.

[11] R.W. Topor, Termination detection for distributed computations, Inform. Process. Lett. 18 (1984) 33–36.

ACHTUNG: Algorithmus ist falsch!