

Übungen (3)

1) Man zeige, dass der Terminierungsalgorithmus 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

- gebe man ein möglichst einfaches Gegenbeispiel an,
- lokalisierere man den Fehler im Korrektheitsbeweis,
- 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:

- für Modelle mit aktiven und passiven Prozessen, wo Nachrichten beliebig lange unterwegs sind, die Nachrichtenkanäle jedoch die FIFO-Eigenschaft besitzen;
- 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. Zeigen Sie 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.

DISTRIBUTED TERMINATION DETECTION ALGORITHM FOR DISTRIBUTED COMPUTATIONS

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

Departments 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, \dots, p_n . Let a local predicate c_i be associated with process p_i , $1 \leq i \leq n$. 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 ($1 \leq i \leq n$) 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, $1 \leq i \leq n$, 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 , $1 \leq i \leq n$, 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 ($1 \leq i \leq n$)

1. *Upon becoming passive:*
`state(p_i) := passive;`
/ state(p_i) has been used for recording the status of p_i */*
for each $p \in \{\text{neighbours of } p_i\}$ do
`send I-am-passive to p ;`
2. *Upon receiving an I-am-passive message from p ($p \in \{\text{neighbours of } p_i\}$):*
begin
`state(p) := passive;`
/ state(p) is used for recording the status of process p in p_i */*
`if state(p_i) = passive for all $p_j \in \{\text{neighbours of } p_i\}$`
`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(p_j) = passive for all $p_j \in \{\text{neighbour of } p_i\}$`
`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 for-

warded 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.

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 dis-

tributed 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.

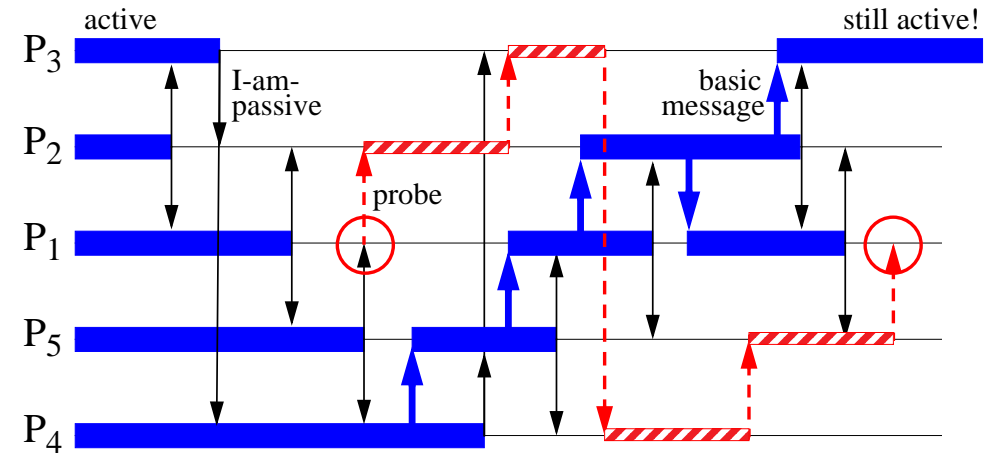
Thus, we find that our presented algorithm is simple and involves less message overheads.

References

- [1] K.R. Apt and J.L. 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#184, 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. Rana, 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.

Gegenbeispiel zum Algorithmus von Arora, Rana, Gupta

- Ringtopologie sowohl für Kontroll- als auch Basisnachrichten

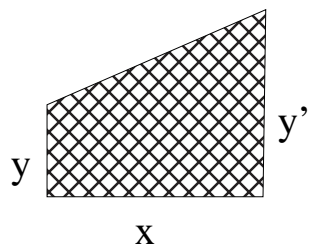
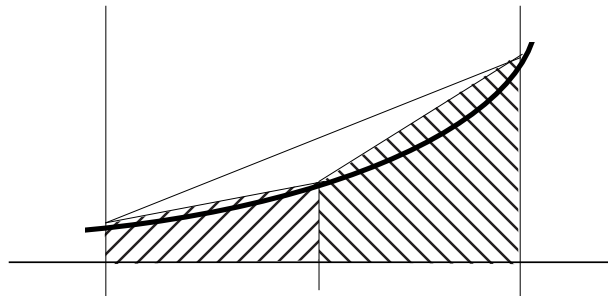


- Siehe auch: Letter to the Editor "Comments on Distributed Termination Detection Algorithm for Distributed Computation", Inf. Proc. Lett. 23 (1986) pp. 163

- Aber auch die Antwort der Autoren: Letter to the Editor "More Comments on Distributed Termination Detection Algorithm for Distributed Computation", Inf. Proc. Lett. 29 (1988) pp. 53-55

Parallele Berechnungsschemata und verteilte Terminierung

Einfaches Beispiel: Berechnung einer Fläche (numerische Integration: Trapezmethode)



$$A = x \frac{y + y'}{2}$$

Prinzip:

- x-Intervall iterativ / rekursiv halbieren bis zu einem *stop-Kriterium*
- Teiltrapezflächen dann aufaddieren

Beachte: Mathematische Aspekte hier ausgeklammert!

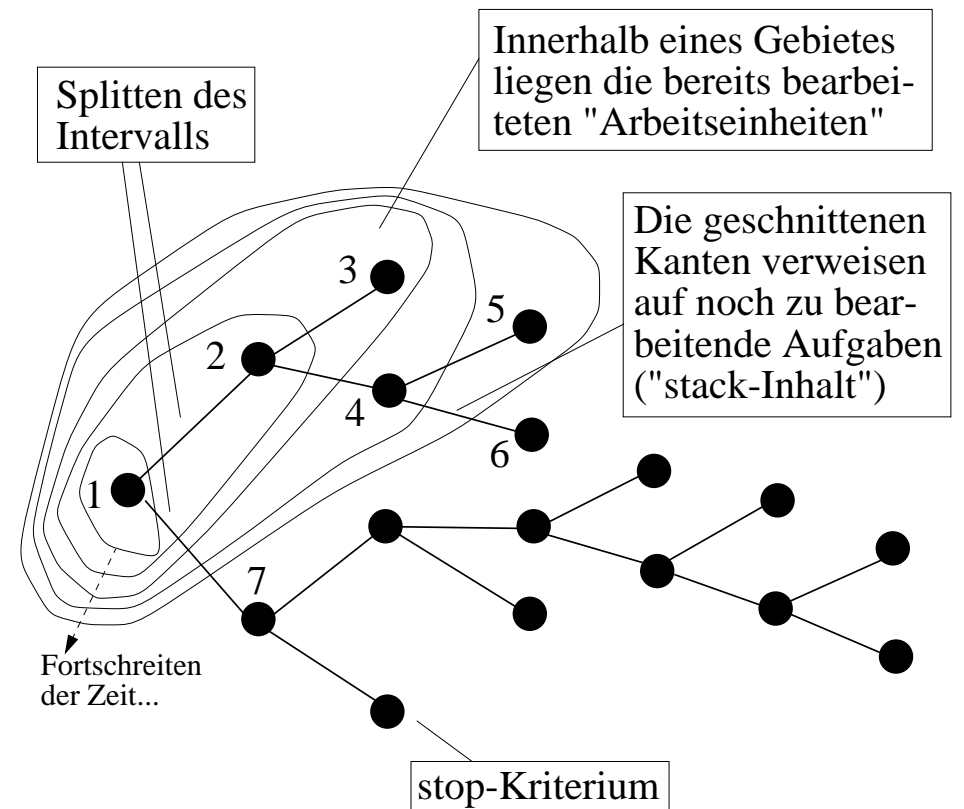
Mögliche *stop-Kriterien*:

- berechnete Trapezfläche bis auf ϵ gleich halber Fläche des "Vaterintervalls"
- Steigung Sekante / Mitteltangente nahezu gleich

Sequentielle Berechnung

- Iterative bzw. rekursive Berechnung ist (uns) klar

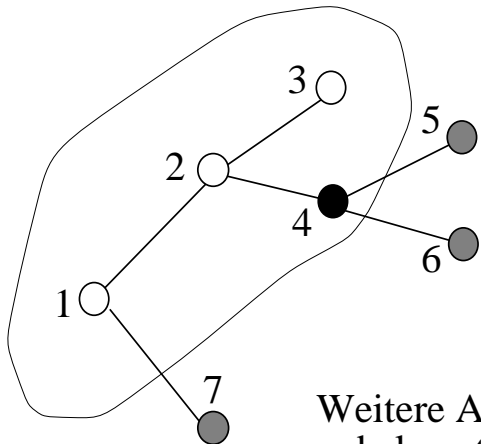
- *Visualisierung*:



- Teilflächenwerte stehen an den Blättern!

Parallelisierung

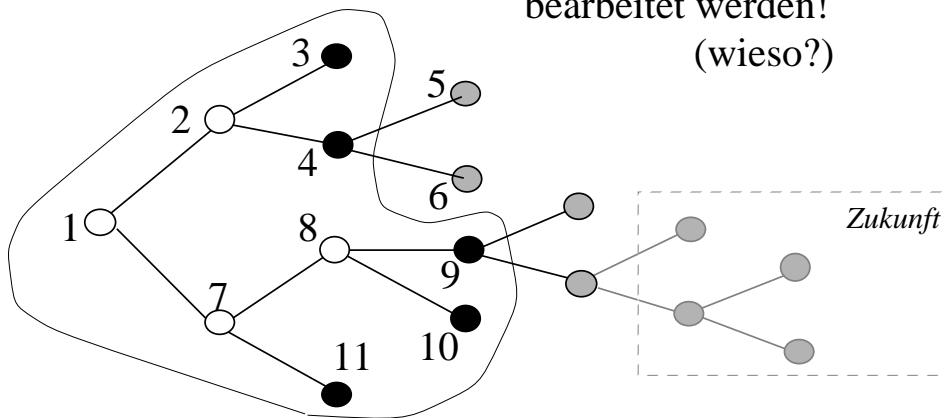
"Schnappschuss" der sequentiellen Berechnung:



- Aufgabe 4 wird gerade bearbeitet
- Aufgaben 5, 6 und 7 sind bereits "generiert"
- Aufgaben 1, 2 und 3 sind bereits bearbeitet

Weitere Aufgaben sind noch unbekannt, werden erst später generiert

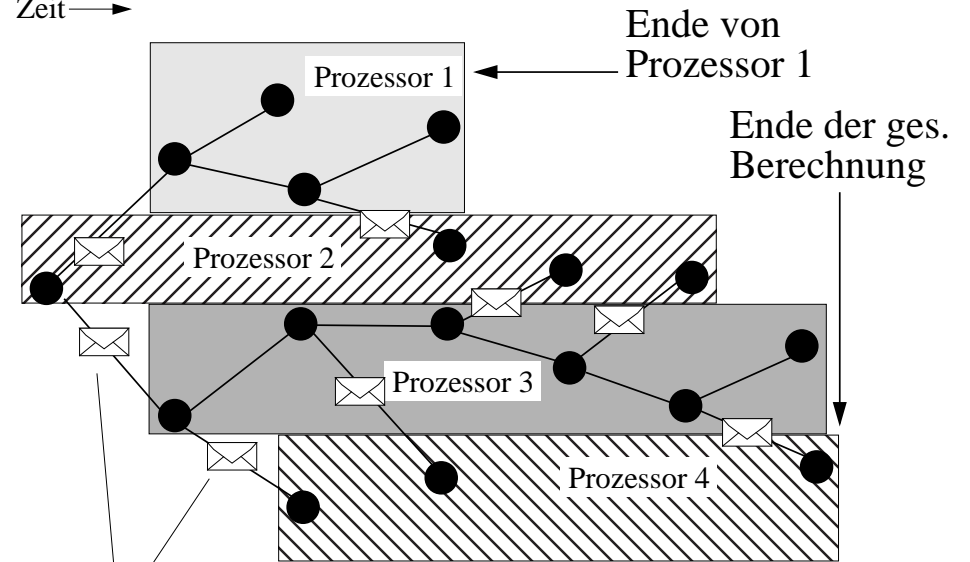
Parallelisierung: Aufgaben 3,4,9,10,11 können gleichzeitig bearbeitet werden!
(wieso?)



"Zeithorizont" bewegt sich in mehrere Richtungen gleichzeitig

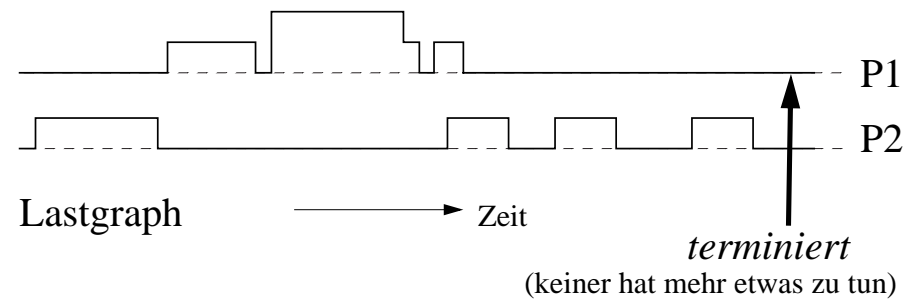
Lastausgleich und Terminierung

Hier globale Sicht:
Zeit →



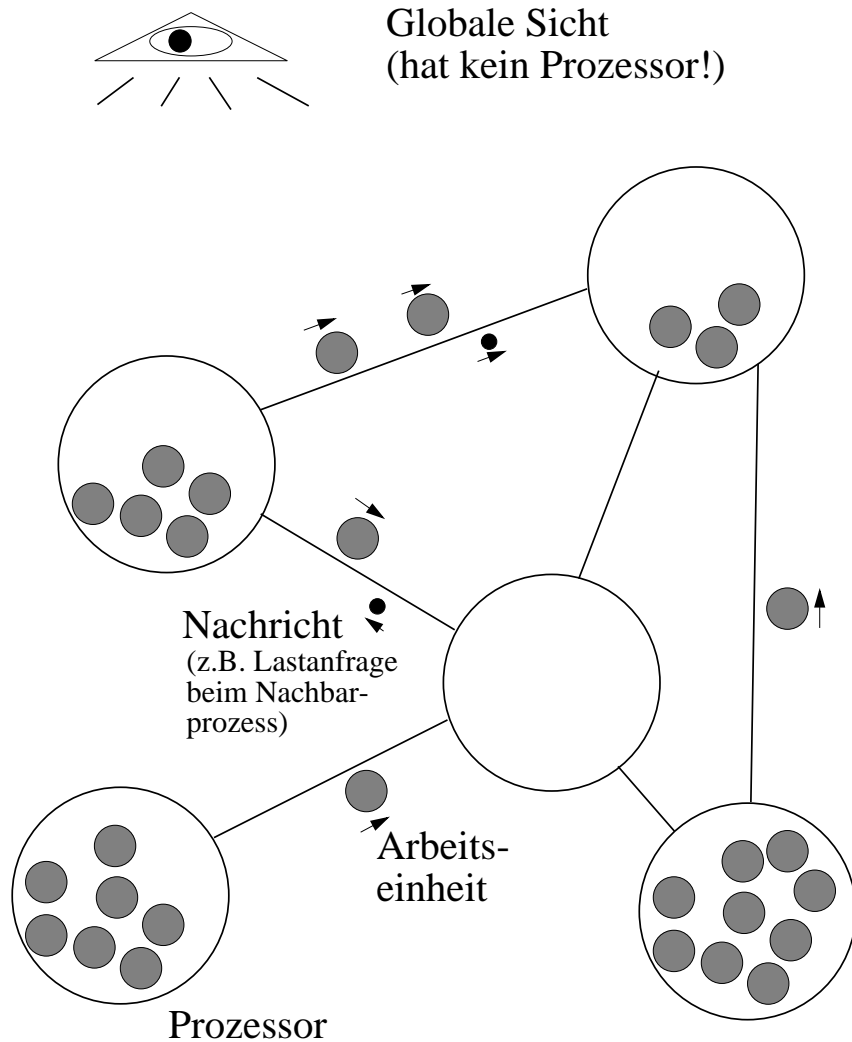
Übermittlung von Arbeitseinheiten an andere Prozessoren

- a) neu entstandene Einheiten
- b) Lastausgleich

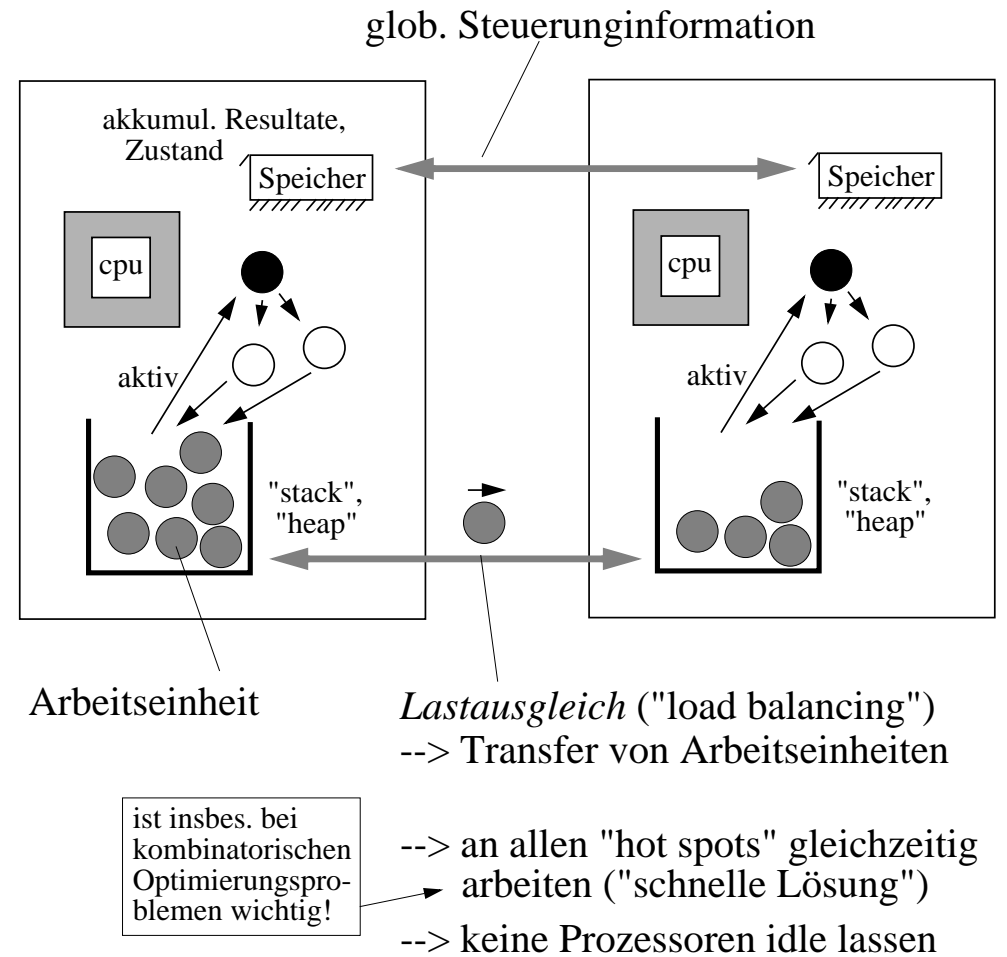


Terminiert, wenn Last überall 0 und nichts "unterwegs"

Globaler Schnappschuss einer Berechnung



"Operationale" Sichtweise



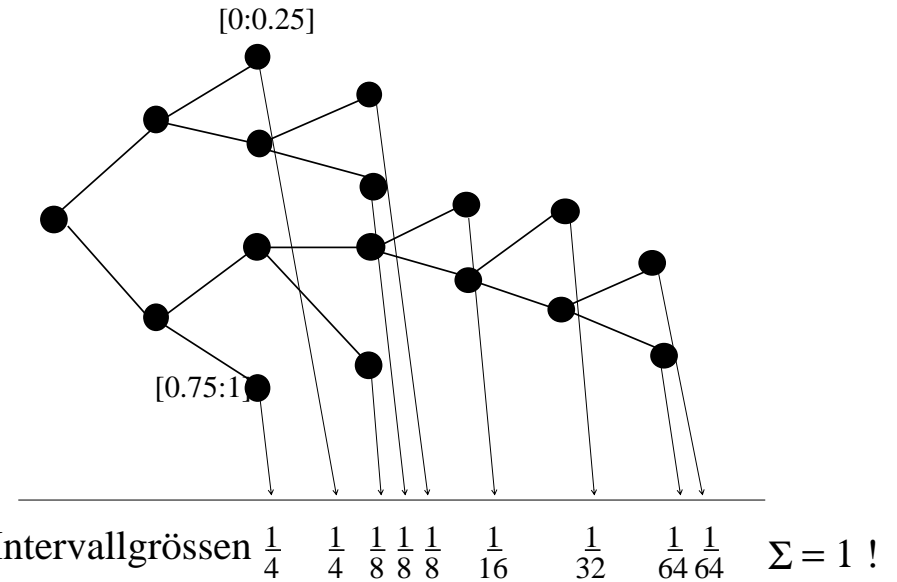
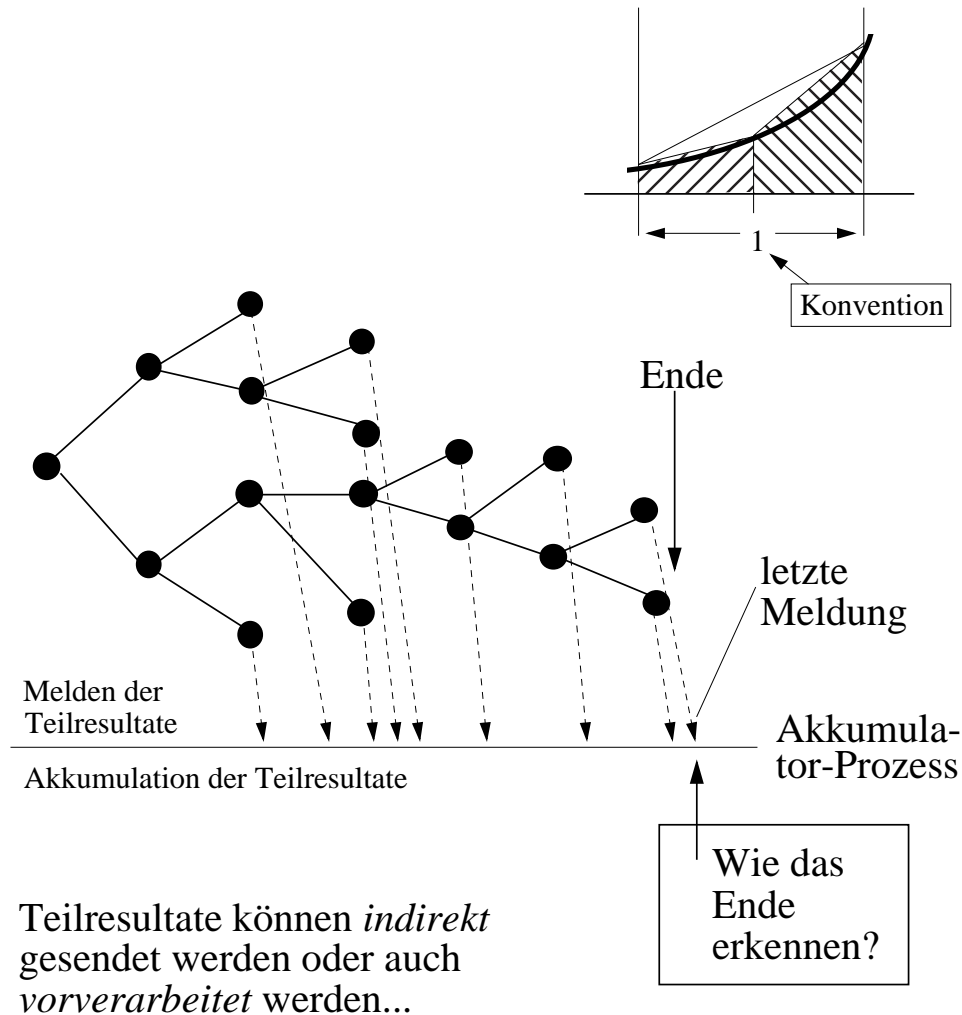
Was *ist* Last? (Mass für Last?)

Wie funktioniert Lastausgleich?

--> verschiedene Modelle, Strategien, Heuristiken, Verfahren...

Parallele numerische Integration: Erkennung der Terminierung

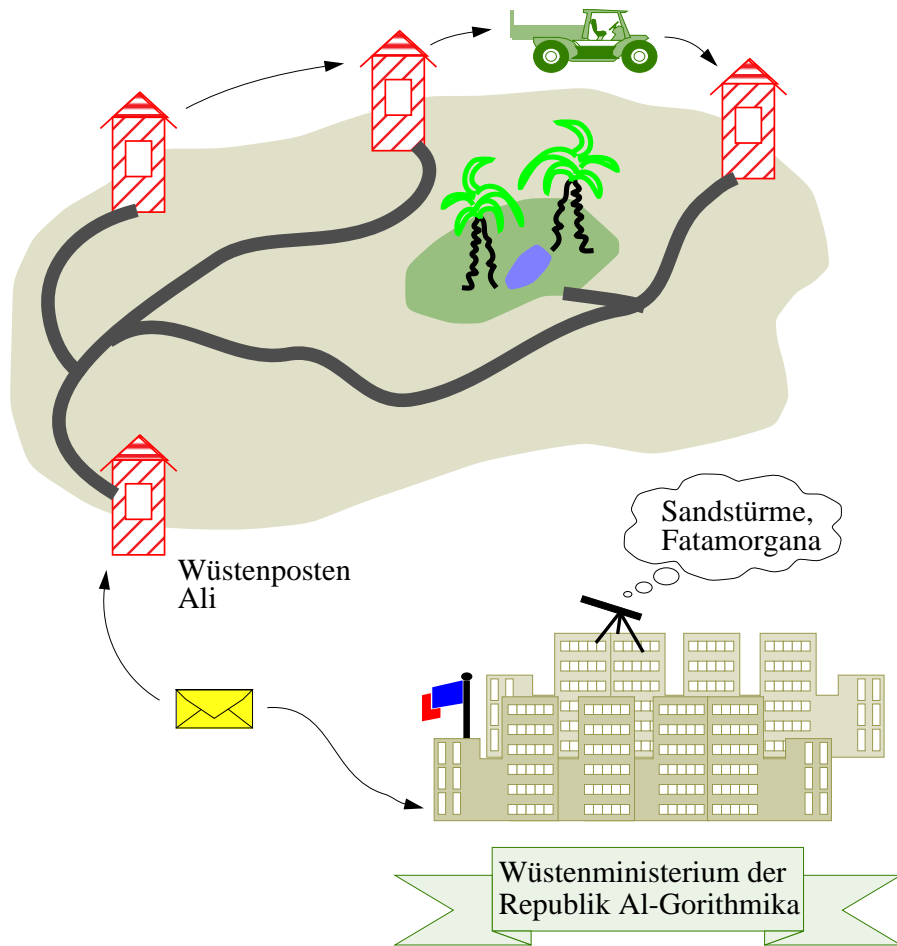
Lösung des Terminierungsproblems



Statt $\frac{1}{2^i}$ besser nur i versenden (bleibt "klein")

Lässt sich die Lösung auf allg. verteilte Berechnungen übertragen?

Das Problem der Wüstenposten



Vorschriften und Probleme...

- Reisende dürfen die Wüste nur "kontrolliert" (bei einem "wachen" Posten) betreten
- Nur ein wacher Posten lässt Personen in die Wüste einreisen
- Nur aus der Wüste kommende Reisende können einen Posten wecken
- Kontrolleure sollen die Beobachtungen der Wüstenposten an das Ministerium melden

Problem: Wann ist die Wüste leer?

- Fatamorgana und Sandstürme trüben die globale Sicht des Ministers
- Zählen der Reisende geht oft "schief", wie vorhin bereits eingesehen

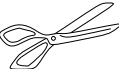
Eine orientalische Lösung


- Reisende erhalten *Eintrittskarten*
- Rückgabe bei Austritt aus der Wüste
- Ministerium gibt feste Zahl von Tickets aus
- Ministerium sammelt Tickets wieder ein

Invariante: Gesamtzahl der Tickets

- Aktiver Posten soll einen nichtleeren Vorrat an Tickets haben
- Wenn das Ministerium alle Tickets wieder eingesammelt hat, ist kein Reisender in der Wüste unterwegs und sind alle Posten passiv --> *terminiert*

Was tut ein Posten, wenn er sein letztes Ticket verkaufen müsste?

- Neue anfordern --> bürokratischer Aufwand!
- Orientalischer Trick: *Ticket halbieren* und nur das halbe Ticket verkaufen... 

 Puzzle-Abteilung im Ministerium...

Die Kreditmethode

- Idee:

Verallgemeinerung des beim Integrationsbeispiel gefundenen Prinzips

Also: Akkumulation eines Wertes, bis dieser =1

Bedingungen:

- (1) Urprozess startet die verteilte Berechnung
- (2) Prozesse und Nachrichten haben einen Kreditanteil $\in Q^+$
- (3) Summe aller Kreditanteile stets = 1
- (4) Aktiver Prozess hat Kreditanteil > 0
- (5) Nachricht hat Kreditanteil > 0

Invariante

Theorem: Dann Berechnung *terminiert*, wenn der Urprozess die Kreditsumme 1 wiedererlangt hat

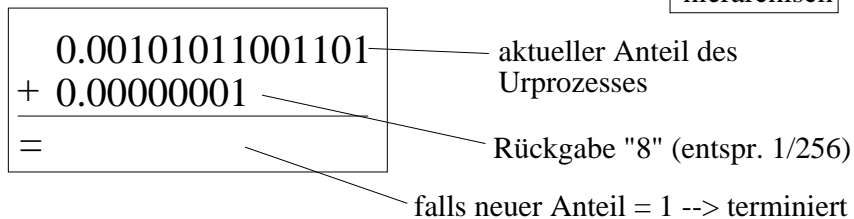
Man muss also die Bedingungen erfüllen ("*safety*") und dafür sorgen, dass der Urprozess die gesamte Kreditsumme wiedererhält ("*liveness*")

Kreditmethode - Realisierung

- (1) Wird ein Prozess passiv, übermittelt er seinen Kreditanteil an den Urprozess.
- (2) Der Kreditanteil einer ankommenden Nachricht wird dem Empfänger zugeschlagen.
- (3) Ausgesandte Nachricht erhält Hälfte des Anteils des Senders.

Implementierung:

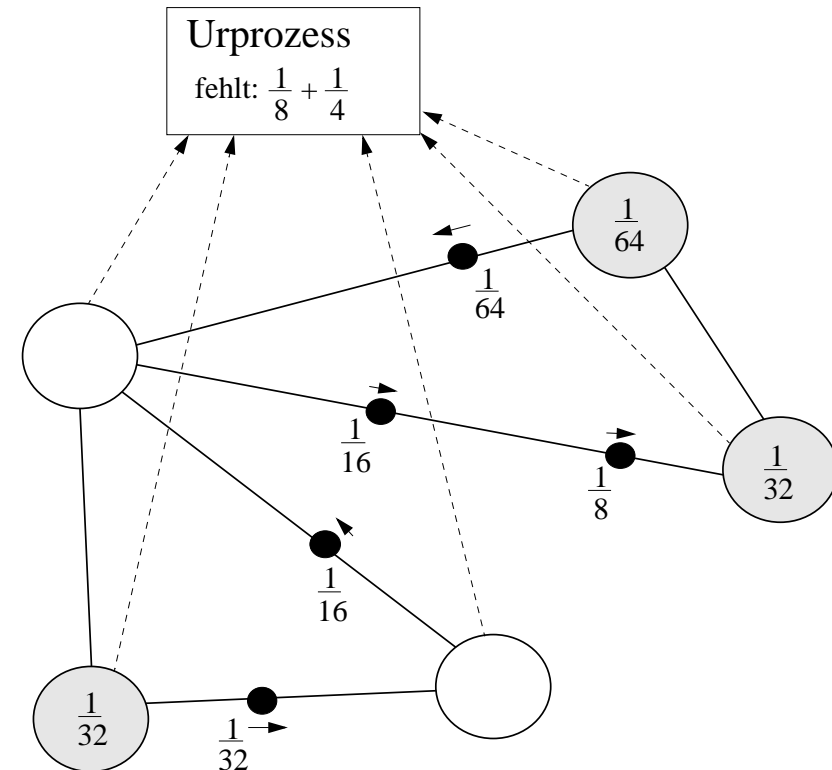
- (1) Gleitpunktzahlen für Kreditanteile unbrauchbar
Lösung: Bruchdarstellung
- (2) Problem: Nenner schnell zu gross
Lösung: Da stets nur negative 2er Potenzen
--> nur Exponent des Nenners ("negativer Logarithmus")
--> halbieren: KREDIT := KREDIT + 1
- (3) Problem: Rekombination (Addition) der Anteile beim Urprozess (bzw. auch den anderen Prozessen)



- (4) Problem: Länge der Bitleiste
Lösung: Anzahl *fehlender* Kreditanteile stets beschränkt ("klein") --> speichere *Komplement* als Menge

Kreditmethode - ein Beispiel

Alle Kreditanteile sind von der Form $\frac{1}{2^k}$

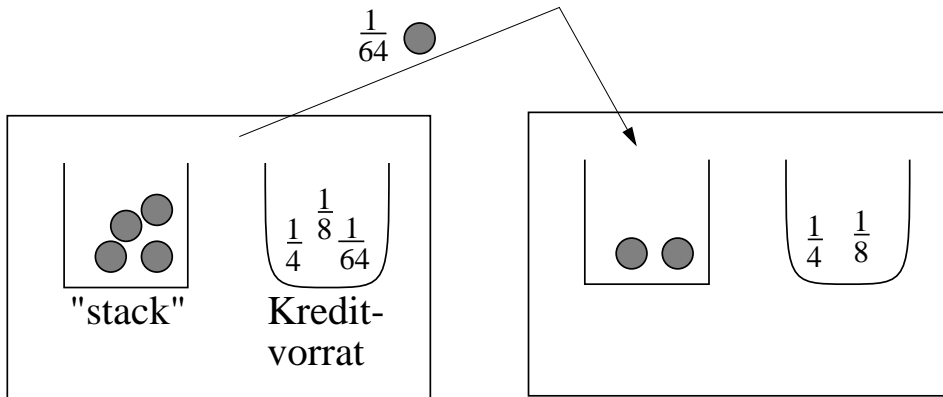


Der Urprozess muss nicht mehr "Krümel" halten, als davon im System (d.h. in aktiven Prozessen, in Basisnachrichten oder in Kontrollnachrichten zum Urprozess) sind.

(Der Urprozess kann die ihm noch fehlenden Krümel sogar oft kompakter speichern!)

Kreditmethode beim parallelen Berechnungsschema

1) Kreditanteile werden weitgehend dezentral von den Prozessoren verwaltet; lokale Arbeitseinheiten benötigen keinen Kredit.



- 2) Bei Versenden einer Arbeitseinheit: Diese bekommt einen Anteil aus dem lokalen Kreditvorrat
- 3) Lokaler Kreditvorrat darf bei einem "aktiven" Prozessor nie leer werden: ggf. muss ein Kreditanteil (der letzte) halbiert werden
- 4) Anteil eines ankommenden Arbeitseinheit wird dem Empfänger zugeschlagen; ggf. (falls gleicher Anteil dort schon vorhanden): iterativ rekombinieren (d.h. soweit möglich aufaddieren)
- 5) Wenn stack leer (oder auf explizite Anforderung): Kreditvorrat an den zentralen Prozess übermitteln

Rekombination der Kreditanteile

- Einsammeln der Kredite auch durch beliebigen Wellenalgorithmus möglich (initiiert durch Urprozess)
 - dabei Rekombination eingesammelter Kredite ggf. hierarchisch

- Rekombination der Kreditanteile beim *Urprozess*:

- Initial: $D = \{0\}$;

steht für 2^{KREDIT}

- Bei Empfang von $\langle \text{KRÜMEL} \rangle$:

```

K := KRÜMEL;
while K ∉ D begin
    D := D ∪ {K};
    K := K-1;
end;
D := D - {K}
if D = ∅ then terminated fi;
    
```

- Wieso klappt dieses Schema?

Rekombination - Ein Beispiel

Initial: $D = \{0\}$

$\frac{1}{4}$ ("2") $\rightarrow \cup\{2\}, \cup\{1\}, -\{0\} \rightarrow D = \{2, 1\}$

$\frac{1}{2}$ ("1") $\rightarrow -\{1\} \rightarrow D = \{2\}$

$\frac{1}{64}$ ("6") $\rightarrow D = \{6, 5, 4, 3\}$

$\frac{1}{8}$ ("3") $\rightarrow D = \{6, 5, 4\}$

$\frac{1}{64}$ ("6") $\rightarrow D = \{5, 4\}$

$\frac{1}{16}$ ("4") $\rightarrow D = \{5\}$

$\frac{1}{32}$ ("5") $\rightarrow D = \{\}$
 \rightarrow *Terminierung!*

$\Sigma = 1$

Kreditmethode - Bewertung

- Topologievoraussetzungen?
- Zentraler Urprozess: Engpass?
 - ggf. weitere Hierarchiestufen einführen
 - Kreditanteile durch einen Wellenalgorithmus einsammeln
- (Worst-case) Nachrichtenkomplexität? (= Overhead)
 - wenn passive Prozesse ihren Kreditanteil stets zurücksenden
 - wenn nicht mehr benötigte Anteile von einer Welle eingesammelt werden
- lok. Speicheraufwand? (Insbes. beim Urprozess)
 - wie gross kann die Menge dort gehaltener "Krümel" werden?
- lok. Berechnungsaufwand?
- "Detection Delay" (nach erfolgter Terminierung)?
- Qualitativer Vergleich mit anderen Verfahren?
- Varianten?

- Es gilt: Es gibt keinen Algorithmus zur Feststellung der verteilten Terminierung, der eine bessere Worst-case-Nachrichtenkomplexität als $O(m+n)$ hat

- m = Anzahl der Basisnachrichten; n = Anzahl der Prozesse
- wie könnte man eine solche Aussage beweisen?
- Kreditmethode ist daher "worst-case-optimal"!

\Rightarrow Prinzip: Binäre Subtraktion!

Variante: "Nachlaufverfahren"

- Idee: Kontrollnachrichten laufen auf "benutzten" Kanälen den Basisnachrichten hinterher, um die Kredite zurückzufordern
 - Überholen dabei keine Basisnachrichten (wie realisieren?)
 - Spalten sich bei Prozessen "geeignet" auf (Kreditanteile)
 - Kehren (direkt oder indirekt) zum Urprozess zurück, wenn kein benutzter Kanal mehr existiert
-

- Eine spezielle Ausprägung davon: hinter *jeder* Nachricht *direkt* herlaufen, um den Kredit wieder zurückzuholen
 - *Diesen* Typ von Kontrollnachrichten kann man sich sparen! ("Verheiraten" mit der zugehörigen Basisnachricht)
 - Acknowledgement wird sofort generiert, wenn der Empfänger aktiv ist.
 - Aber wenn der Empfänger passiv ist, wann dann?
 - Empfänger wartet mit der Rückgabe des Kredits (d.h. mit dem Acknowledgement), bis er selbst Acknowledgements zu allen von ihm selbst ausgesendeten Nachrichten erhalten hat
 - Kreditwerte zu versenden, kann man sich sparen: Man bekommt schliesslich genau den Wert zurück, den man versendet hat!
 - Was bleibt vom Verfahren also übrig? (--> Neuformulierung: Jeder Prozess zählt gesendete Nachrichten und empfangene Acknowledgements...)

Diese Idee *indirekter Acknowledgements* ("diffusing computations"-Verfahren von Dijkstra und Scholten, 1980) erinnert sehr an den *Echo-Algorithmus*!

Man könnte also überhaupt verteilte Berechnungen als Instanzen (einer einfachen *Variante*!) des Echo-Algorithmus hinsichtlich der Explorer-Nachrichten auffassen. Die Echo-Nachrichten übernehmen dann wie gehabt die Erkennung der Terminierung!

Denkübung: Wie genau sieht die Variante des Echo-Algorithmus aus?

- Tip: dynamischer Graph mit ggf. parallelen Kanten
- Wann Echo-Nachricht versenden? Gesamtzahl?