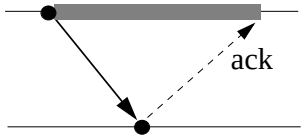
Ein Beispiel für eine Modelltransformation

Gegeben:

- Berechnung im Atommodell
- Terminierungsalgorithmus für Synchronmodell

statt Atommodell

Was tun? --> Einführung von "passiv" und "aktiv"
(Modelltransformation)



- Sende ack zurück bei Empfang einer Nachricht
- Sender wird "aktiv" beim Senden, "passiv", wenn alle Nachrichten quittiert

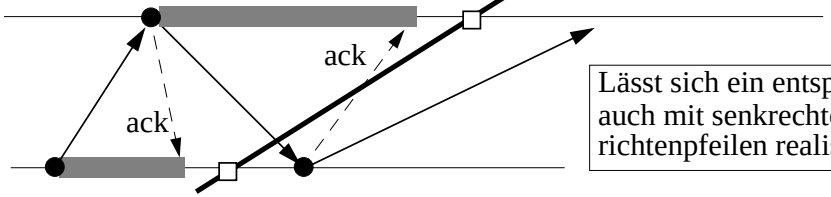
Dann gilt: Alle "passiv" ==> keine Nachricht unterwegs

stellt Terminierungsalgo. für Synchronmodell fest

ist Terminierungsdefinition für das Atommodell

Aber was ist damit?:

Zu jeder (bekannten) Nachricht wurde ein ack empfangen und alle sind passiv. Terminiert?

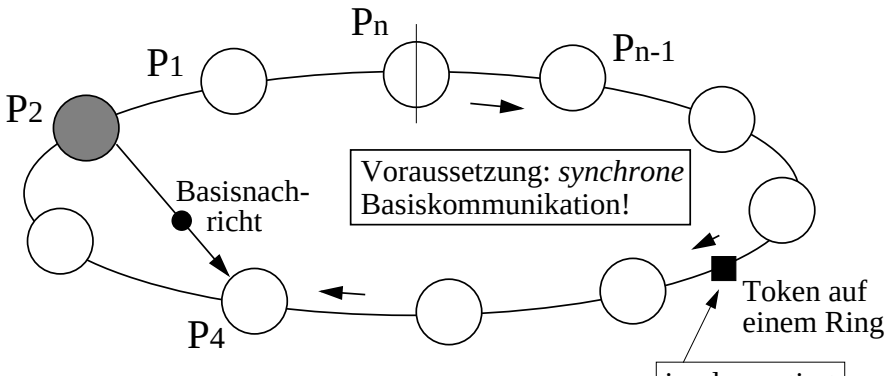


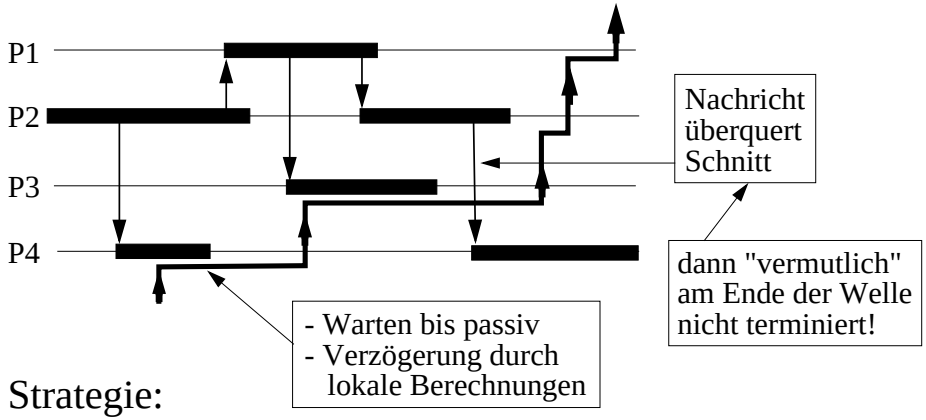
Lässt sich ein entspr. Szenario auch mit senkrechten Nachrichtenpfeilen realisieren?

- "Alle sind passiv" (entlang einer schiefen Schnittlinie!) ist kein korrektes Terminierungskriterium im synchronen Fall!
- Genausowenig wie "alle passiv und alle acks angekommen" im Transaktionsmodell!

Der DFG-Algorithmus

(Dijkstra, Feijen, Van Gasteren, 1983, Inf. Proc. Lett.)





Strategie:

- Prozesse und Token können schwarz oder weiss sein
- Prozess wird schwarz, wenn er an einen Prozess mit einer höheren Nummer etwas sendet
- Welle testet, ob ein Prozess schwarz ist und färbt Prozess ("auf der Rückflanke") weiss
- Terminiert, wenn alle weiss

DFG-Algorithmus - Verhaltensregeln

Regel 1: Ein Prozess, der eine Basisnachricht [an einen Prozess mit einem höheren Index] sendet, wird schwarz.

Regel 2: Wenn Prozess P_n passiv ist, kann er den Terminierungstest initiieren, indem er ein weisses Token an P_{n-1} sendet.

Regel 3: Ein aktiver Prozess behält das Token, bis er passiv wird.

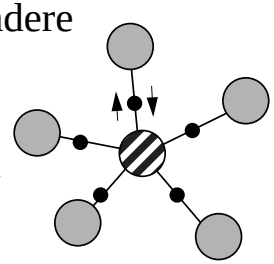
Regel 4: Ein passiver Prozess P_i ($i \neq n$), der das Token hat, reicht ein schwarzes Token weiter an P_{i-1}, wenn er oder das Token schwarz ist, ansonsten reicht er ein weisses Token weiter.

Regel 5: Ein Prozess, der das Token weiterreicht, wird weiss.

Regel 6: Wenn Prozess P_n ein weisses Token erhält, meldet er *Terminierung*.

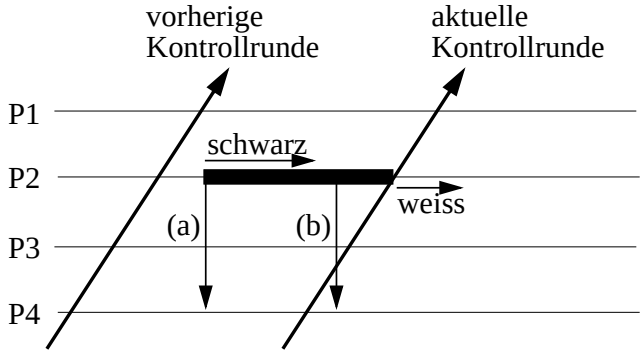
Regel 7: Wenn Prozess P_n ein schwarzes Token erhält, startet er eine neue Runde.

- Korrektheit ? (Safety; Liveness)
- Muss das Token mittels synchroner Kommunikation propagiert werden?
- Wieso klappt der Algorithmus nicht für asynchrone Basiskommunikation?
- Worst-case Nachrichtenkomplexität? "Detection delay"?
- Muss der Initiator eindeutig sein? Mehrere Initiatoren?
- Statt Kontrollring andere Realisierung der Kontrollwelle?
- Welche Farbe haben die Prozesse initial?



z.B. Stern mit einem zentralen Initiator, der *parallel* Token zu den Prozessen schickt, um deren Farbe zu ermitteln

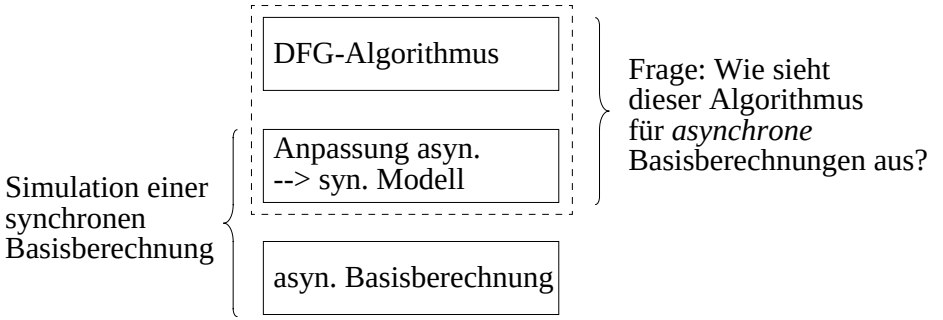
DFG-Algorithmus - "falscher Alarm"



- 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

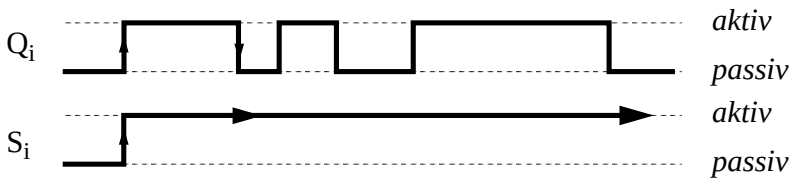
- Voraussetzung: Synchrone Kommunikation
  - Terminierungskriterium: “Sind alle Prozesse (gleichzeitig) passiv?”

- 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!

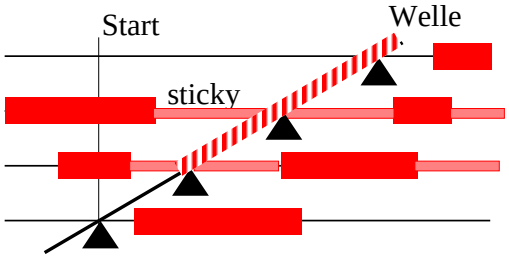
- Idee: “Sticky flag” S_i statt dessen betrachten!



- 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

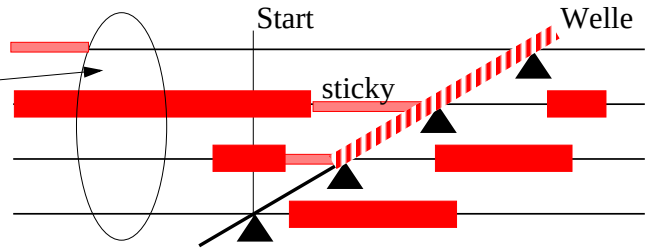
Beh.: Wenn ein P_j bei Start der Kontrollwelle aktiv war
==> Terminierung wird nicht gemeldet

<==> 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:

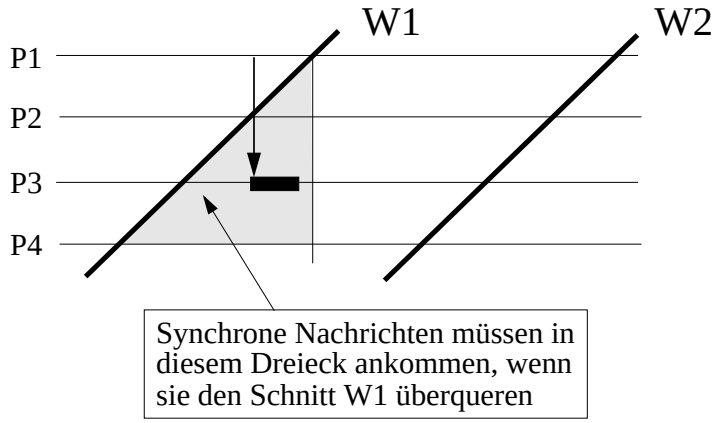
```
receive <...>;
S_i := active
```

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)