Formale Betrachtung paralleler Programme

1 Petri-Netze

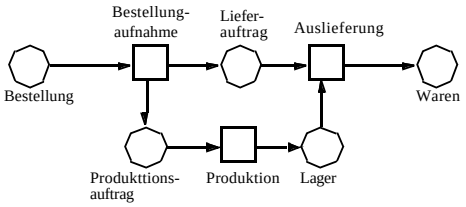
1.1 Einfache Petri-Netze

Die einfachsten Petri-Netze bilden einen gerichteten Graphen mit zwei disjunkten Mengen von Knoten.

Die Knoten werden als Stellen (*places*) und Transitionen (*transitions*) bezeichnet.

Die Gerichteten Kanten führen jeweils von einem Element der einen zu einem Element der anderen Knotenmengen. Die Menge der Kanten des Graphen nennt man Flußrelation (*flow relation*).

Beispiel:



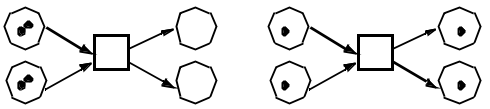
1.2 Stellen-Transitions-Netze

Ein Stellen-Transitions-Netz (*state transition net*) beschreibt den Zustand eines Systems, indem Stellen mit Marken (*tokens*) versehen werden.

Eine Zustandsänderung wird durch ein Schalten (*firing*) von Transitionen dargestellt.

Eine Transition kann schalten, wenn alle Stellen ihres Vorbereichs markiert sind. Durch das Schalten wird je eine Marke aus jeder Stelle des Vorbereichs genommen. In jeder Stelle des Nachbereichs wird eine Marke erzeugt.

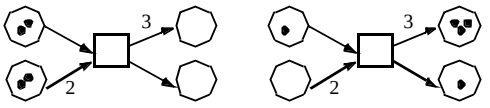
Beispiel:



Vor dem Schalten Nach dem Schalten

Kanten können eine Gewichtung (*weight*) tragen, die angibt, wie viele Marken beim Schalten der Transition von einer Eingangsstelle entfernt bzw. bei einer Ausgansstelle hinzufügt werden müssen.

Beispiel:

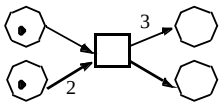


Vor dem Schalten Nach dem Schalten

1.3 Eigenschaften von Petri-Netzen

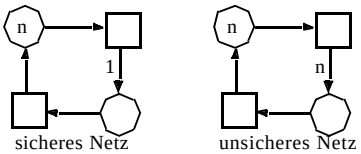
Wenn in einem Netz der Zustand eintreten kann, daß keine Transition schalten kann, nennt man das Netz todesgefährdet, sonst keißt es lebendig (*live*).

Beispiel:



Ein Petri-Netz heißt sicher (*safe*), wenn es nicht mehr Marken erzeugt, als es aufnimmt.

Beispiel
:

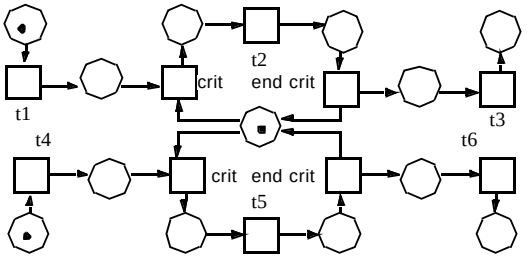


sicheres Netz unsicheres Netz

1.4 Synchronisation in Petri-Netzen

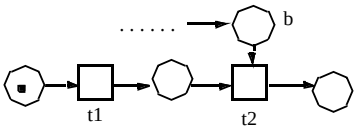
- Mehrseitige Synchronisation

```
P1: t1; crit k do t2 end crit; t3;
P2: t4; crit k do t5 end crit; t6;
```

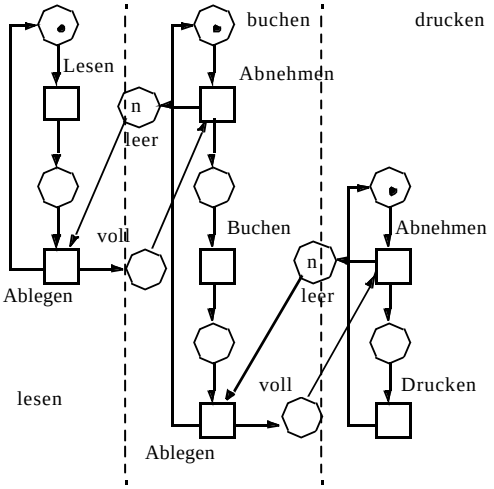


- Einseitige Synchronisation

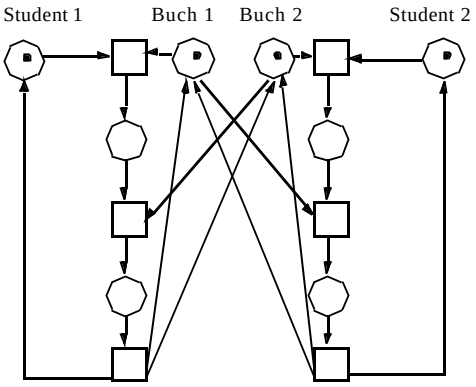
```
P:  t1; wait b; t2;
```



Beispiel: Produzenten und Konsumenten



Beispiel: Zwei Studenten



2 CSP und Modelchecking

CSP - *Communicating Sequential Process* (Hoare 1978) hat als Beschreibungsmittel von Prozessen, die miteinander Nachrichten austauschen, eine herausragende Bedeutung erlangt.

2.1 Sprachelemente von CSP

• Reihenfolge-Operation: a ?? P

Beispiel: Student 1

leihen.1.b1 ?? leihen.1.b2 ??
zurueckgeben.1.b1 ?? zurueckgeben.1.b2 ? ? . . .

• Auswahl-Operationen

externe (deterministische) Auswahl P [] Q
interne (nicht deterministische) Auswahl P |~|Q

Beispiel: Buch 1

leihen.1.b1 ?? zurueckgeben.1.b1 ?? . . .
[] leihen.2.b1 ?? zurueckgeben.2.b1 ? ? ? . .

leihen.1.b1 ?? zurueckgeben.1.b1 ?? . . .
|~| leihen.2.b1 ?? zurueckgeben.2.b1 ? ? ? . .

• Rekursion

Beispiel: Buch
Buch(i) = leihen.1.i ?? zurueckgeben.1.i ?? Buch(i)
[]
leihen.2.i ?? zurueckgeben.2.i ? ? ? Buch(i)

• Parallel-Operation: P |||Q
Beispiel: Buch(1) ||| Buch(2)

• SKIP und STOP

• Kommunikationsanweisungen

Eingabe: c?x

Ausgabe: c!a

2.2 Synchronisation mit CSP

P [| events |] Q

Beispiel: Timer-Prozeß

Timer(N) = setT ?? Timer_1(N)
Timer_1(n) = if (n==0)
then elaT ? ? Timer(N)
else tickT ? ? Timer_1(n-1)
Sender = send ? ? setT ? ? (ack ? ? Sender
[]
elaT ?? error ?? Sender)
TSender = Sender [| {| setT, elaT |} |] Timer

2.3 Kommunikation in CSP

P = in ? x ?? c1 ! x ?? c2 ? y ?? out ! y ?? P
Q = c1 ? z ?? c2 ! z*z ?? Q

PQ = P [| {| c1, c2 |} |] Q

Synchrone Kommunikation!

Beispiel: Produzenten und Konsumenten

```
Buffer(n, mess, N) =
      (n==0) & (write?m ?? Buffer(n+1, <m>, N))
      []
      (n==N) & (read!head(mess), Buffer(n-1, tail(mess), N))
      []
      (n>0 and n< N) &
          (write?m ?? Buffer(n+1, mess^<m>, N)
          []
           read!head(mess) ?? Buffer(n-1, tail(mess), N))

Producer = produce(p); write!p ?? Producer

Consumer = read?x ?? SKIP; consume(x); Consumer

PC_SYSTEM = Buffer(0, <>, 3)
                  [| {| read, write |} |]
                  (Producer ||| Consumer)
```

Beispiel: Zwei Studenten

```
Student_1 =  leihen.1.b1 ?? leihen.1.b2 ??
      zurueckgeben.1.b1 ?? zurueckgeben.1.b2 ?? Student_1

Student_2 =  leihen.2.b2 ?? leihen.2.b1 ??
      zurueckgeben.2.b1 ?? zurueckgeben.2.b2 ?? Student_2

Buch(i) = leihen.1.i ?? zurueckgeben.1.i ?? Buch(i)
          []
          leihen.2.i ?? zurueckgeben.2.i ?? Buch(i)

ST_SYSTEM =  (Buch(1) ||| Buch(2))
                  [| {| leihen, zurueckgeben |} |]
                  (Student_1 ||| Student_2)
```


2.4 Model-Checking mit FDR

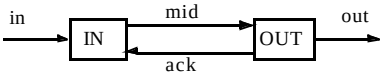
• Prüfung von Verklemmungsfreiheit

PC_SYSTEM :[deadlock free [F]]

ST_SYSTEM :[deadlock free [F]]

• Sicherheitsprüfung

COPY = in?x ?? out!x ?? COPY



IN = in?x?? mid!x ?? ack ?? IN
OUT = mid?x?? out!x?? ack ?? OUT
IO_SYS = (IN [| {| mid, ack |} |] OUT) \ {| mid, ack |}

COPY [T= IO_SYS

(IO_SYS verfeinert COPY im Traces-Modell)