Universität Bielefeld – Technische Fakultät
AG Rechnernetze und verteilte Systeme

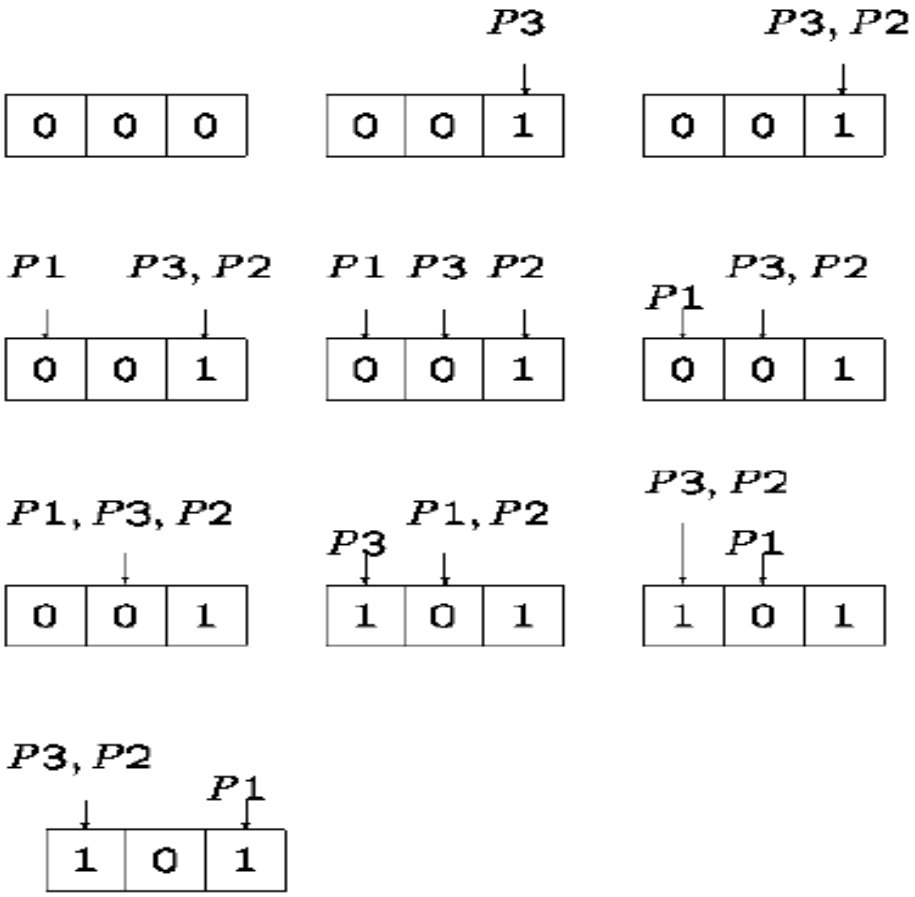
Vorlesung 9.2: Specification

Wintersemester 2001/2002

Peter B. Ladkin
ladkin@rvs.uni-bielefeld.de

Beschriebung eines Prozesses

- Um den Lauf eines Prozesses bzw. einiger nebenläufigen Prozessen darzustellen, braucht man
  - Eine Beschreibung der nacheinanderfolgenden Zuständen
  - Eine Beschreibung der Operationen, die innerhalb eines Prozesses nacheinander folgen oder, die unter den nebenläufigen Prozessen teilweise gleichzeitig, teilweise nacheinander laufen


P3
P3, P2
0 0 0
0 0 1
0 0 1
P1
P3, P2
P1 P3 P2
P3, P2
P1
0 0 1
0 0 1
0 0 1
P1, P3, P2
P3
P1, P2
P3, P2
P1
0 0 1
1 0 1
1 0 1
P3, P2
P1
1 0 1

1

Zustände

- 1. Zustand: = (Bit 0 ist 0; Bit 1 ist 0; Bit 11 ist 0)
- 2. Zust. = 3. Zust. = 4. Zust. = 5. Zust. = 6 Zust. = 7. Zust. = (B0 ist 1; B1 ist 0; B11 ist 0)
- 8. Zust. = 9. Zust. = 10. Zust. = (B0 ist 1; B2 ist 0; B11 ist 1)

Mehr über Zustände

- Nicht nur Wert von Bits, sondern auch Position der von P1, P2, P3 ausgeführte Befehle; also das Program Counter (PC1, PC2, PC3)

Zustände

- Also gibt es 6 Programm-Variablen
  - B0, B1, B11
  - PC1, PC2, PC3
- Jeder Zustand unterscheidet sich von den anderen über den gemeinsamen Wert der Reihe von Programm-Variablen
- Dies muss nicht unbedingt sein; Zustände könnten im allgemein wiederholt werden

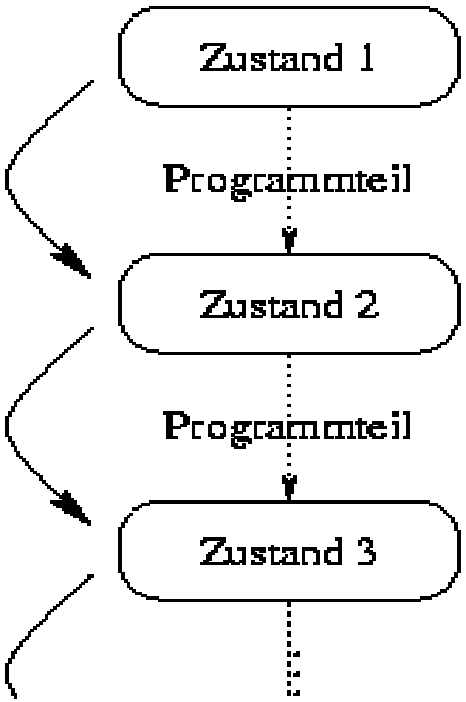
Wert(en) der Programm-Variablen

- { <B0,1>, <B1,0>,<B11,1>, <PC1,B11>, <PC2,B0>, <PC3,B0/B1> }
- Menge von Paaren
- Funktion mit Argumenten = Variabel-Namen und Werte = die Werte von den Variablen in dem Zustand
- Array mit Indizien = Variabel-Namen und Werte = die Werte von den Variablen.....

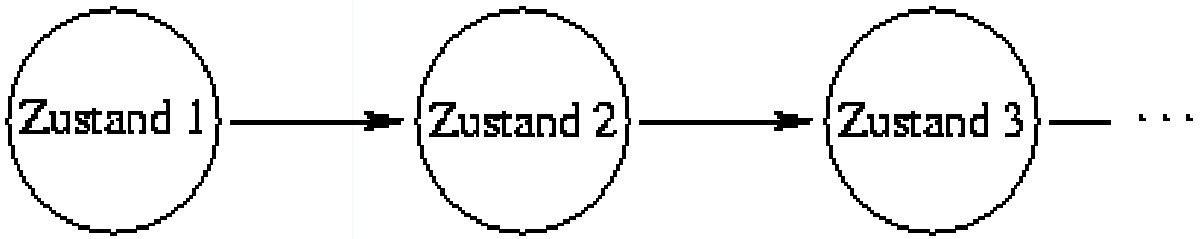
Alles umbenennen

- Variablen-Namen: Fluents
- Werte: die Werte
- Werte sind alle reine Mengen, genau wie in der Mathematik (aber Zahlen schrieben wir sowieso wie in der Mathematik)
- Also unterscheiden sich mathematischen Variablen (unbestimmte Werte) von Fluents (Programm-Variablen)

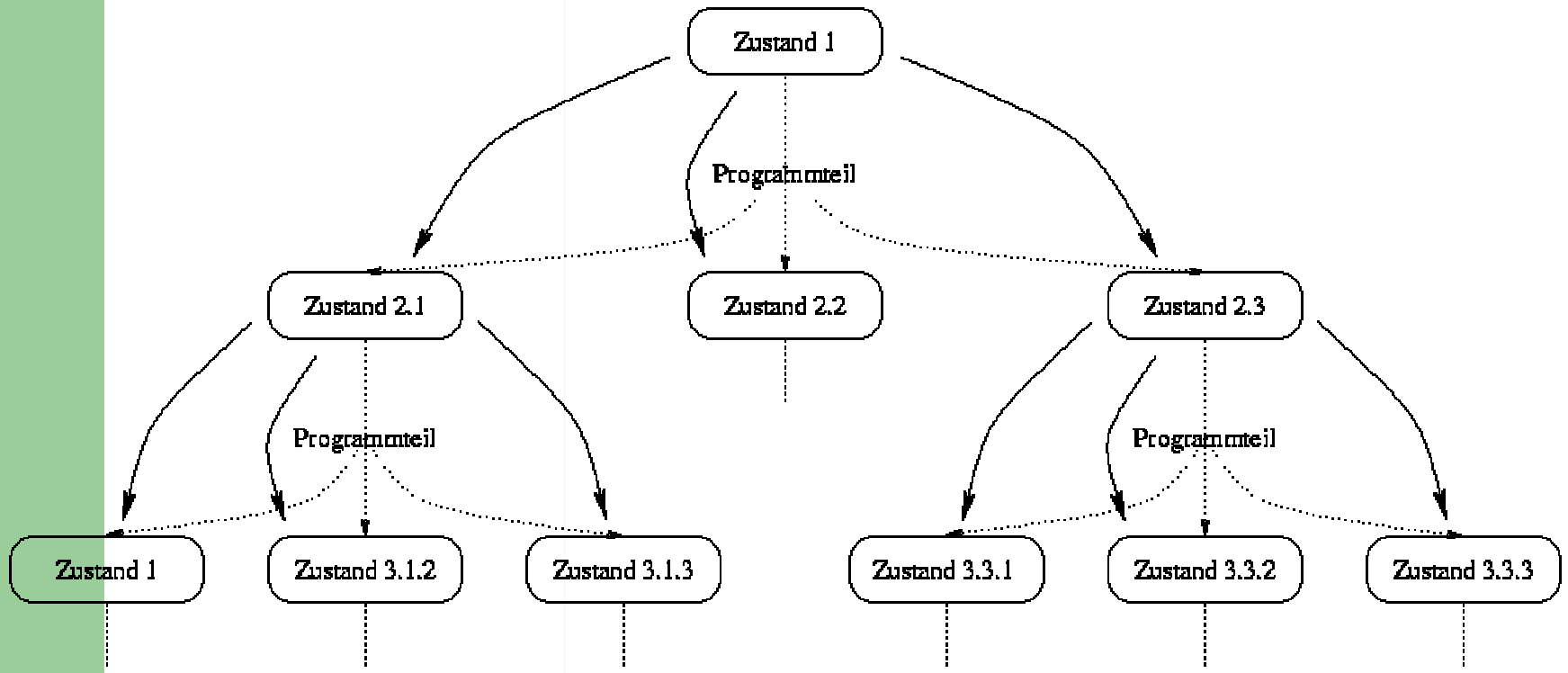
Programmlauf



Programmlauf anders'rum



Wie ein Programm laufen kann



Wie ein Programm laufen kann

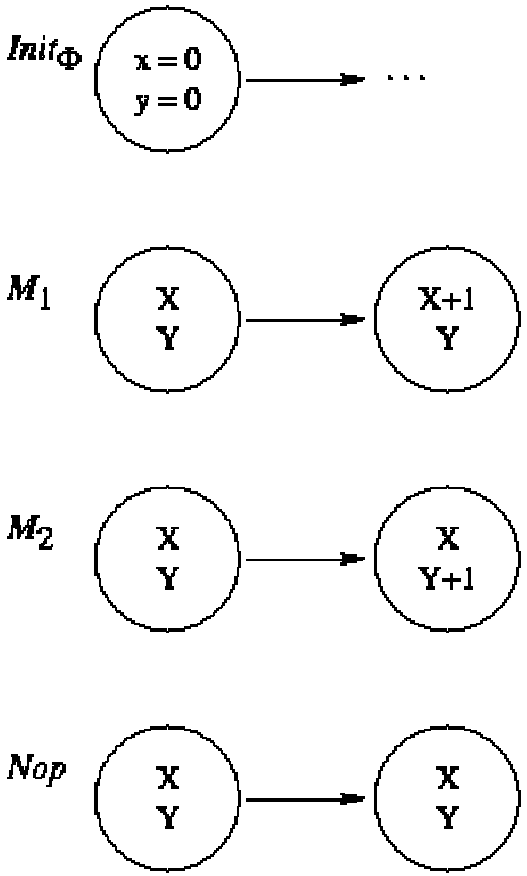
- Wird exponentiell gross, wenn so bezeichnet wie dargestellt
- Wir müssen eine Methode benutzen, die die entsprechenden Eigenschaften aller (oder genügender) "nächsten Zustände" gleichzeitig beschreiben kann

Dijkstra's Notation

- Var integer x,y = 0;

```
loop
   cobegin
      x <- x+1
   []
      y <- y+1
   coend
endloop
```

Zustand-Operation Notation



Zustand-Operation Notation

- Actions (individuelle Zustandsänderungen) sind binäre Relationen zwischen Zuständen
- Eine Relation ist eine Menge von geordneten Paaren
- Die Operation x <- x+1 ist die Operation, in der die Wert von x im Nachfogerzustand ist gleich dem Wert von x im Vorgängerzustand + 1
- Wir schreiben x' = x+1 statt x <- x+1

Übung: Datenstrukturennotation

- Wie sieht ein Zustand dieses Programmes als Menge aus?
- Wie sieht die Operation M1 als Menge aus?
- Wie sieht die Operation M2 als Menge aus?
- Wie sieht die Operation NoOp als Menge aus?

Zustand-Operation Notation

- Also, var integer x,y = 0;

loop
  cobegin
    x' = x+1 ∧ y' = y
  []
    y' = y+1 ∧ x' = x
  coend
endloop

Zustand-Operation Notation

- Oder var integer.....

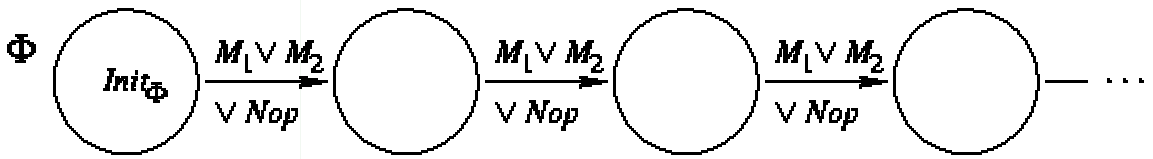
```
loop
   cobegin
     M1
   []
     M2
   coend
endloop
```

Zustand-Operation Notation

- Oder

  Auf jedem Programmschritt:
  M1 $\vee$ M2
  wird aufgeführt
- d.h., immer: M1 $\vee$ M2
- d.h., $\Box$ (M1 $\vee$ M2)

Zustand-Operation Notation



$\Phi \triangleq Init_\Phi \wedge \Box (M_1 \vee M_2 \vee Nop)$

"für immer"

Zustand-Operation Notation

- Allerdings:

$$
\begin{aligned}
&\wedge\ x = 0 \\
&\wedge\ y = 0 \\
&\wedge\ \Box\ (\ \vee \wedge\ x' = x+1 \\
&\qquad\qquad \wedge\ y' = y \\
&\qquad\quad \vee \wedge\ y' = y+1 \\
&\qquad\qquad \wedge\ x' = x \\
&\qquad\quad \vee\ \mathrm{NoOp}\ )
\end{aligned}
$$

Zustand-Operation Notation

- Beschreibt wie das Programm startet, und
- Was passiert bei jeder Action (entweder gar nichts, oder x inkrementiert und y bleibt, oder andersherum)
- Pure Logik- und Mathe-Notation
  - Mit ' und ❑

Lösung

- NichtLesen :== $\wedge$ PC1' = PC1
  $\wedge$ PC2' = PC2
  $\wedge$ PC1 = B0
  $\wedge$ PC2 = B0

Lösung

- PC1-Lesen :== $\wedge \neg$ PC1'= PC1
  $\wedge$ B0' = B0
  $\wedge$ B1' = B1
  $\wedge$ B11' = B11
  $\wedge$ PC3 = B0
  $\wedge$ PC3' = B0

Lösung

- PC2-Lesen :== $\wedge \neg$ PC2'= PC2
  $\wedge$ B0' = B0
  $\wedge$ B1' = B1
  $\wedge$ B11' = B11
  $\wedge$ PC3 = B0
  $\wedge$ PC3' = B0

Lösung

- $NoOp :== \wedge B0' = B0$
  $\wedge B1' = B1$
  $\wedge B11' = B11$
  $\wedge PC1' = PC1$
  $\wedge PC2' = PC2$
  $\wedge PC3' = PC3$

Lösung

- Schreiben :== $\vee$ $\wedge$ PC3 = B0
  $\wedge$ PC3' = B1
  $\wedge$ B1' = B1
  $\wedge$ B11' = B11
  $\vee$ $\wedge$ PC3 = B1
  $\wedge$ PC3' = B11
  $\wedge$ B0' = B0
  $\wedge$ B11' = B11
  $\vee$ NächsterTeil

Lösung

- $\text{NächsterTeil} :== \wedge\ PC3 = B11$
  $\wedge\ PC3' = B0$
  $\wedge\ B0' = B0$
  $\wedge\ B1' = B1$

Lösung: Das Programm

- MutexSchreiben :== $\wedge$ Schreiben
  $\wedge$ NichtLesen

- Program :==
  $\wedge$ B0 = B1 = B11 = 0
  $\wedge \Box$ ($\vee$ Mutexschreiben
  $\vee$ P1-Lesen
  $\vee$ P2-Lesen
  $\vee$ NoOp)