

Vorlesung 9.1: Erinnerung

Peter B. Ladkin

ladkin@rvs.uni-bielefeld.de

Wintersemester 2001/2002

Prozess Synchronisierung Puzzle I

- Process 1: (x: integer)
begin x ← 0; x ← x+1; stop; end
- Process 2: (x: integer)
begin read x; stop; end
- Was ist der gelesene Wert von x, wenn diese Programme concurrent laufen?

Prozess Synchronisierung Puzzle 2

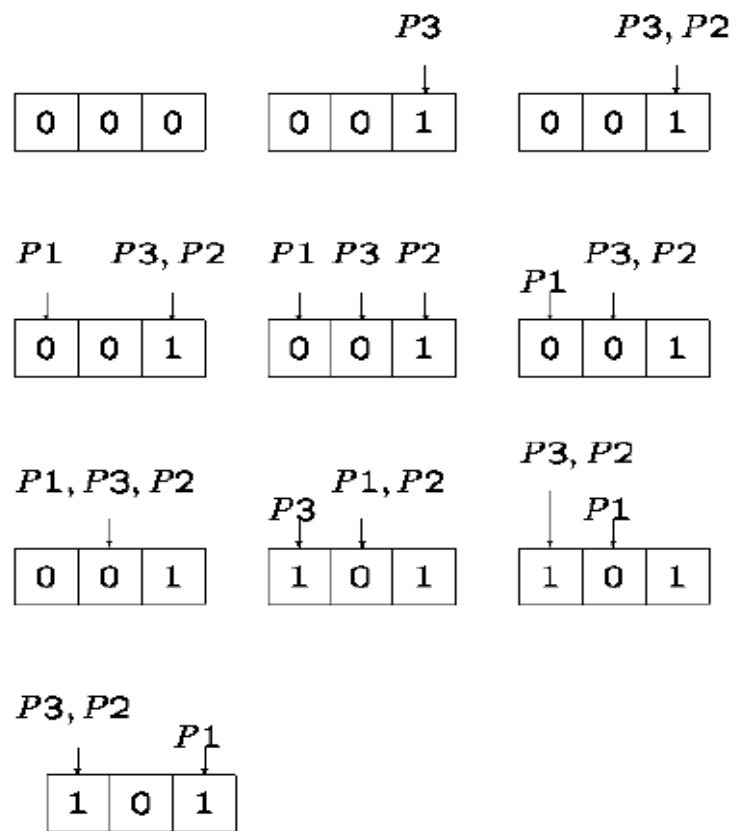
- Prozess 1: (x: integer)
begin x ← 0; x ← x+1; stop; end
- Prozess 2: (x, y: integer)
begin y ← 0; y ← x+1; stop; end
- Voraussetzung: Memory Platz x ist dergleiche als Memory Platz y
- Werte von x, y an Ende?

Prozess Synchronisierung Puzzle 3

- Wert von der Variabel z ist 1, falls es existieren 20 Blöcke freiverfügbarem Speicher;
- ...ist 2, falls es $\dots < 20$ Blöcke....
- Wert von z ist 1
- Prozess 1 braucht 15 Blöcke, Prozess 2 auch
- Beide lesen z gleichzeitig
- Was passiert?

Prozess Synchronisierung Puzzle 4

- Programm 1 und Programm 2 lesen Variabel `turn`
- `turn` könnte von Programm 3 geschrieben werden
- `turn` hat 3 Bits
- `turn = 001` bedeutet, Prog 1 kann den Drucker benutzen
- `Turn = 101` bedeutet, Prog 2



Puzzle 4: Lösung

- Sicherstellen, dass nur ein Prozess Anschluss an die Variabel zu einer Zeit hat
- Dies heisst: **mutex (mutual exclusion)**
- Problemlösung stammt von Edsger Dijkstra
- Turing-Preissieger, Designer des THE Betriebssystems (Eindhoven, 1968)

Semaphore

- Vorteil: mutex sichergestellt
- Nachteil: ineffizient
- Allerdings muss man nur sicherstellen, dass nicht gelesen wird wenn geschrieben wird
- P1 und P2 könnten ohne Gefahr gleichzeitig lesen; nur wenn und dass P3 nicht schreibt

Semaphore: Übung

- Wie kann die effizientere Lösung mit Hilfe von Semaphoren implementiert werden?
- Wie könnten Semaphoren mit Hilfe von Interrupt-Masking programmiert werden?

Semaphoren

- Alle Prozessen können gleichzeitig versuchen, den Semaphor zu holen
- Nur ein Prozess könnte den Semaphore "bekommen"
- Die anderen müssen *warten* (z.B. auf eine Warteschlange) bis der Prozess fertig ist
- Was passiert, wenn der Prozess scheitert?

Semaphoren

- Technische ausgesehen ist ein Semaphor ein *Shared Variable* , deren Anschluss kontrolliert ist
- Zwei Operationen nur: *holen* und *freigeben*
- Ein *Interlock* , der verhindert, dass ein Prozess *in seinen Critical Section* hineingeht, wenn der Semaphor schon gesetzt wird

Semaphoren

- Wie ein einziges Bit
- Gesetzt: P ("passeren")
- Freigegeben: V ("vrijgeven")
- Nur ein Prozess kann zu einem Zeitpunkt eine Operation ausführen
- Andere sind blockiert bis $V(S)$ ausgeführt wird
- d.h., P und V sind *atomäre Operationen (atomic Operations)*

Semaphoren

- Semaphoren und andere atomäre Operationen werden normalerweise im Betriebssystem implementiert

Einfache Mutex

- `Mask (Interrupts) ;`
`Critical Section ;`
`Unmask (Interrupts)`
- Ineffizient
- z.B., P1 und P2 möchten Drucker1 benutzen
P3 und P4 möchten Drucker2 benutzen
- P1 muss nur P2 ausschliessen, P2 nur P1,
P3 nur P4, und P4 nur P3
- P1 könnte gegen P3 ausgetauscht werden, usw

Programmierung mit Semaphoren

- `var integer x,y = 0;`
`semaphore sem = 1;`

```
cobegin
  loop P(sem); CS1; V(sem) endloop
[]
  loop P(sem); CS2; V(sem) endloop
coend
```

Notation

- cobegin ... [] coend
- Die zwei Hälften laufen gleichzeitig
- Es könnte mehrere Klauseln geben
- cobegin ... [] ... [] ... coend
- Notation von Dijkstra
- Prozedurale Sprache - man könnte die *Zustände* des Programms nicht beschreiben

Vorlesung 9.2: Specification



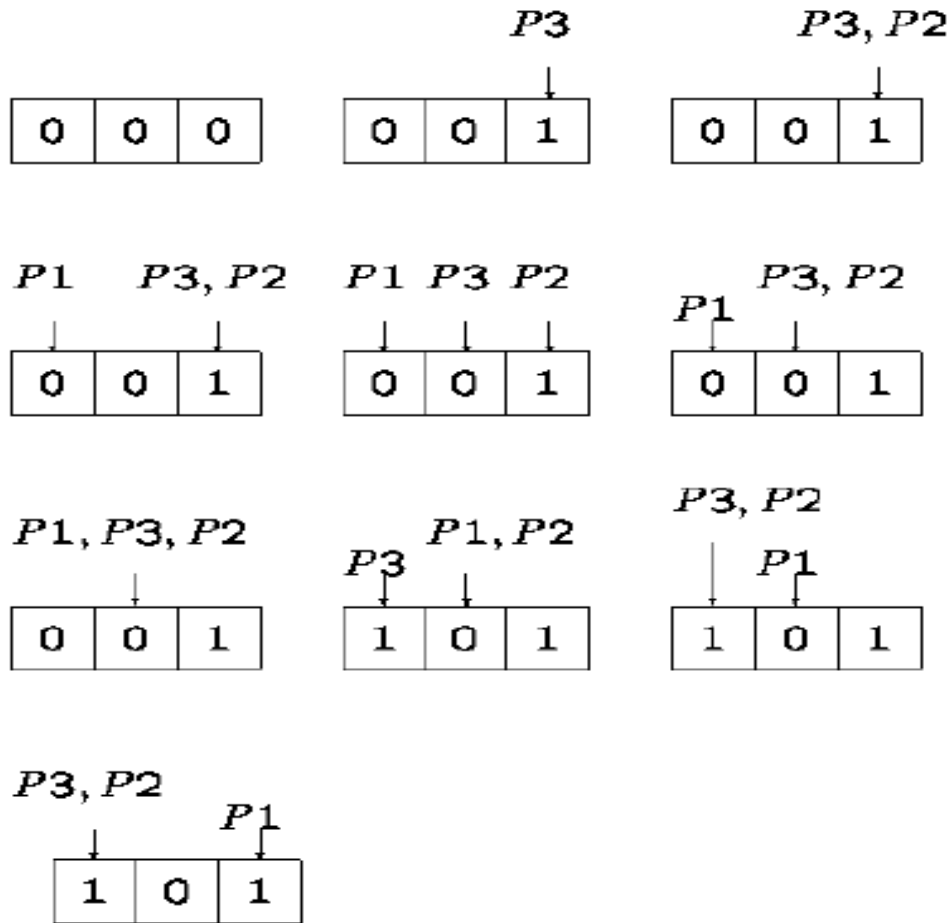
Wintersemester 2001/2002

Peter B. Ladkin

ladkin@rvs.uni-bielefeld.de

Beschreibung 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



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

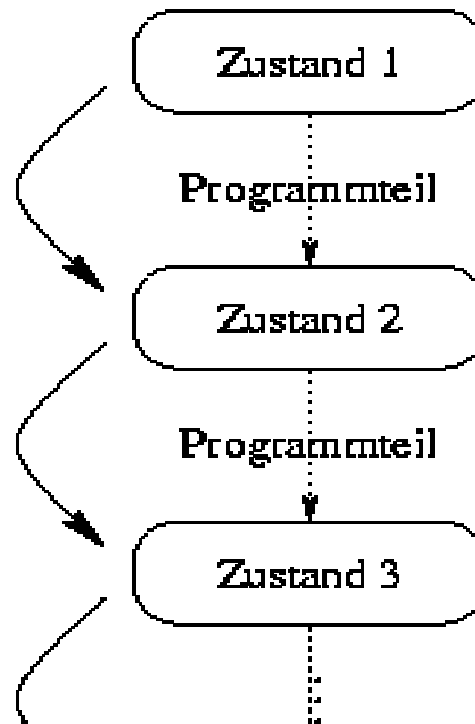
Wert(en) der Programm-Variablen

- { $\langle B0,1 \rangle$, $\langle B1,0 \rangle$, $\langle B11,1 \rangle$,
 $\langle PC1,B11 \rangle$, $\langle PC2,B0 \rangle$, $\langle PC3,B0/B1 \rangle$ }
- 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.....

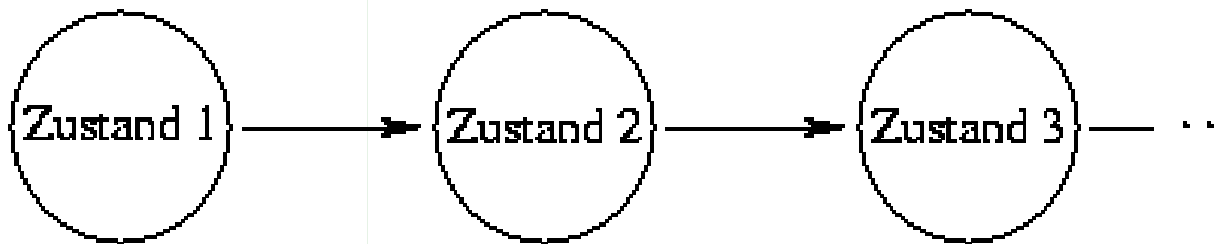
Alles umbenennen

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

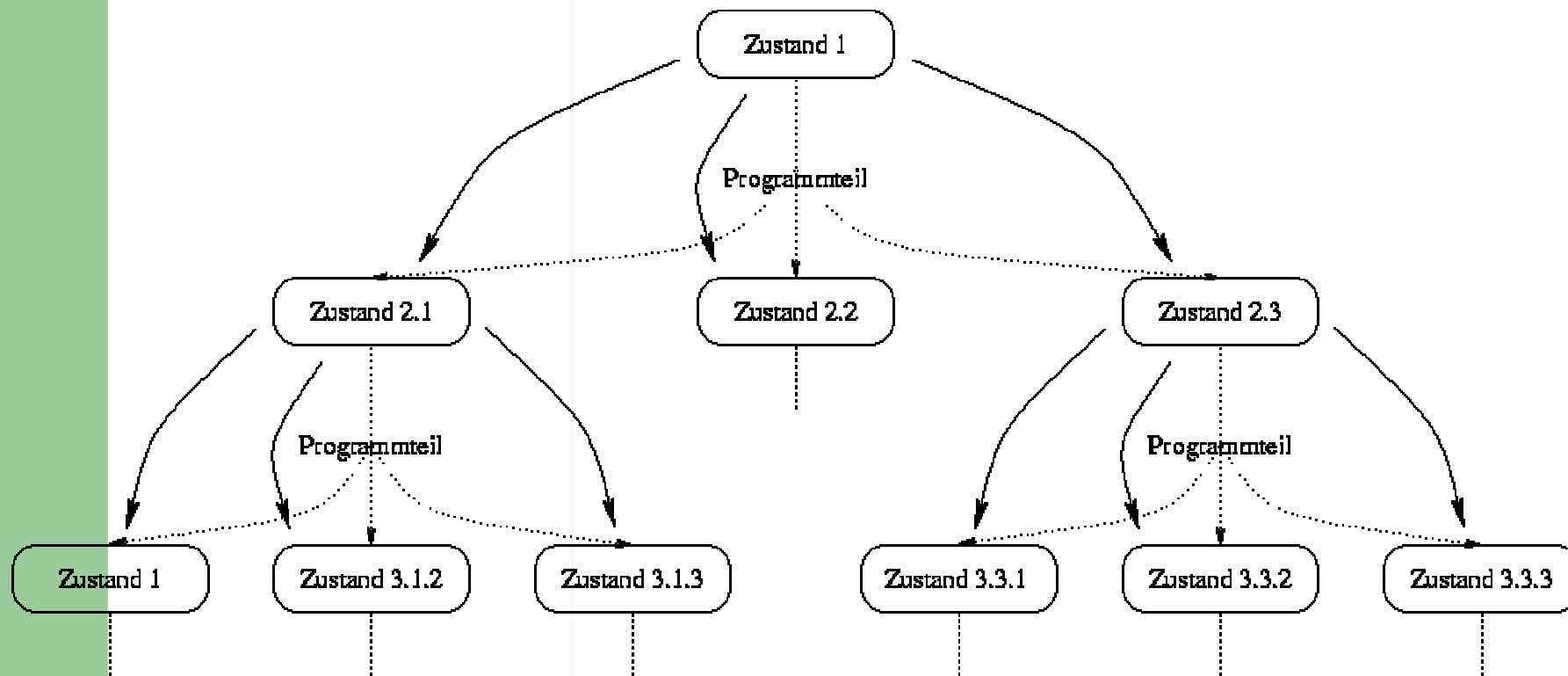
Programmmlauf



Programmmlauf anders'rum



Wie ein Programm laufen kann



Wie ein Programm laufen kann

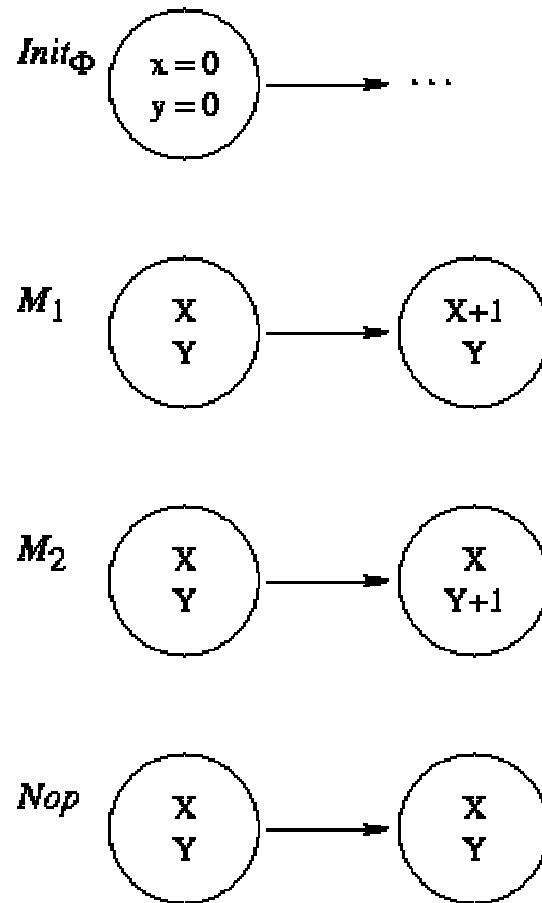
- Wird exponentiell gross, wenn so bezeichnet wie dargestellt
- Wir müssen eine Methode benutzen, die die entsprechenden Eigenschaften aller (oder genügender) "nächsten Zustände" gleichzeitig beschreiben kann

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 $x \leftarrow x+1$ ist die Operation, in der die Wert von x im Nachfolgerzustand ist gleich dem Wert von x im Vorgängerzustand + 1
- Wir schreiben $x' = x+1$ statt $x \leftarrow x+1$

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  $\wedge$  y' = y
  []
    y' = y+1  $\wedge$  x' = x
  coend
endloop
```

Zustand-Operation Notation

- Oder `var integer.....`

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

Zustand-Operation Notation

- Oder

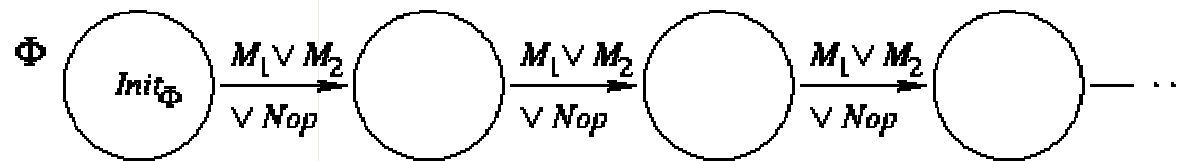
Auf jedem Programmschritt:

$M1 \vee M2$

wird aufgeführt

- d.h., immer: $M1 \vee M2$
- d.h., $\square (M1 \vee M2)$

Zustand-Operation Notation



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

"für immer"

Zustand-Operation Notation

- Allerdings:

$$\wedge x = 0$$

$$\wedge y = 0$$

$$\wedge \square (\vee \wedge x' = x+1$$

$$\wedge y' = y$$

$$\vee \wedge y' = y+1$$

$$\wedge x' = x$$

$$\vee \text{NoOp})$$

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 \square

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

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

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 Nächster Teil

Lösung

- NächsterTeil ::= \wedge PC3 = B11
 \wedge PC3' = B0
 \wedge B0' = B0
 \wedge B1' = B1

Lösung: Das Programm

- $\text{MutexSchreiben} ::= \wedge \text{Schreiben} \wedge \text{NichtLesen}$
- $\text{Program} ::=$
 - $\wedge B0 = B1 = B11 = 0$
 - $\wedge \square (\vee \text{Mutexschreiben}$
 - $\vee P1\text{-Lesen}$
 - $\vee P2\text{-Lesen}$
 - $\vee \text{NoOp})$