Übersicht

Übersicht

Modellierung nebenläufiger Systeme

Systeme, in welchen Komponenten parallel arbeiten:

1. Technische Systeme
2. Softwaresysteme
3. ...

Komponenten arbeiten teilweise

- unabhängig
- abhängig

voneinander.

Beispiel

Einfaches Programm mit boolescher Variable s:

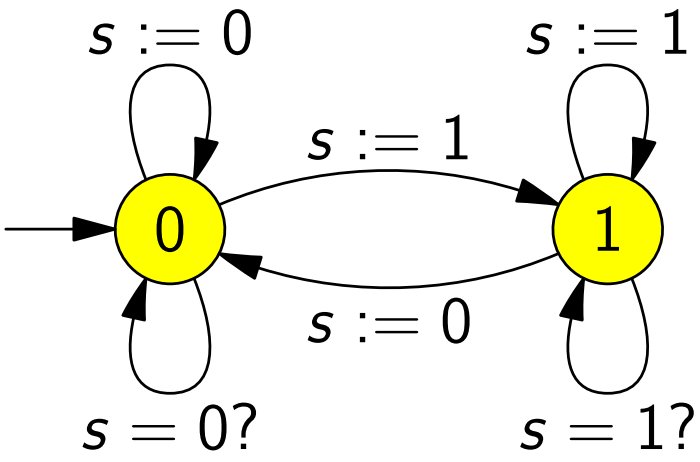
```
s := 0;
s := 1;
if (s=0) print;
else exit;
```

Zerlegen in Komponenten:

- Kontrollstruktur (welche Anweisung wird ausgeführt?)
- Inhalt der Variablen

Interaktion, falls Folgeanweisung vom Variableninhalt abhängt.

Modellierung einer booleschen Variable:

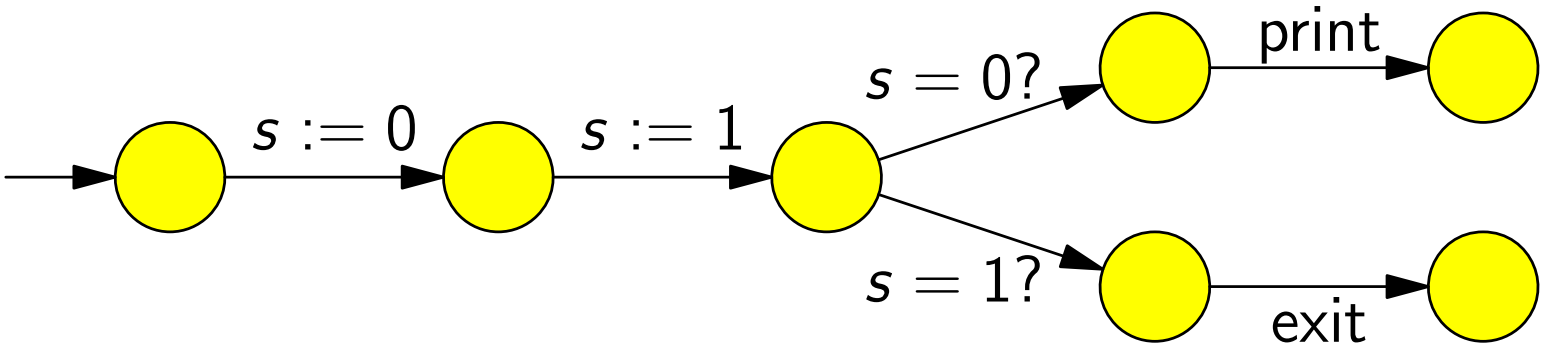


Mögliche Aktionen:

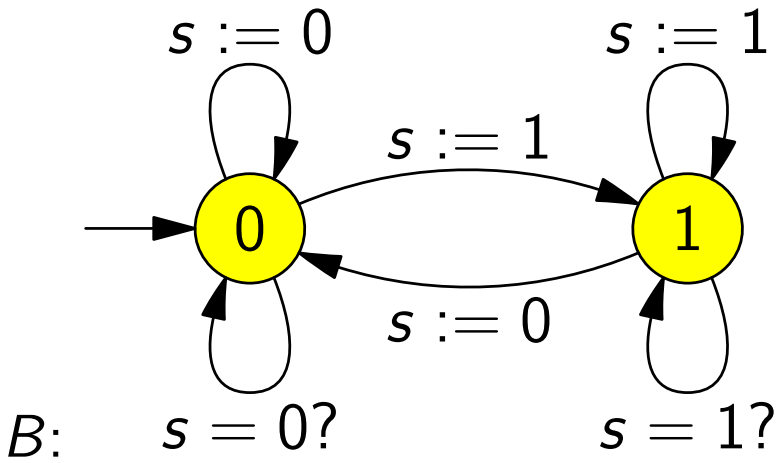
$s := 0$	Variable auf 0 setzen
$s := 1$	Variable auf 1 setzen
$s = 0?$	Bestätige, dass Variable 0 ist
$s = 1?$	Bestätige, dass Variable 1 ist

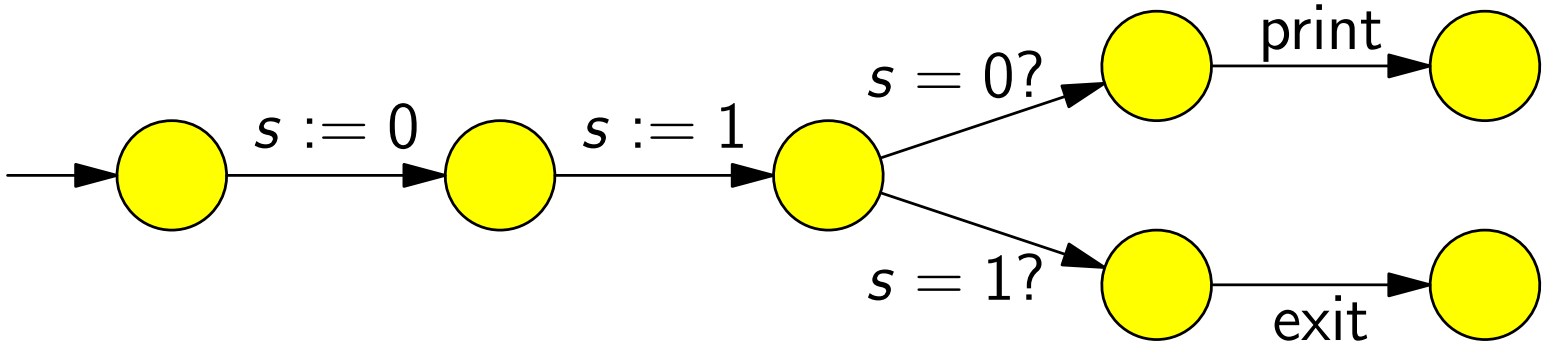
```
s := 0;
s := 1;
if (s=0) print;
else exit;
```

Modellierung der Kontrollstruktur:

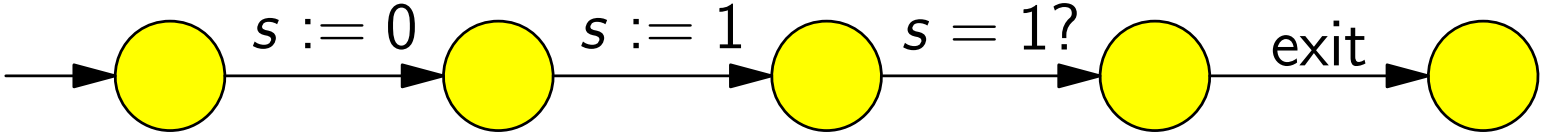


Synchronisiertes Produkt





$B \circ K$:



Synchronisiertes Produkt

Definition 5.1.1

Es seien $M' = (\Sigma', Q', \delta', q_0', F')$ *und* $M'' = (\Sigma'', Q'', \delta'', q_0'', F'')$ *zwei NFA's.*

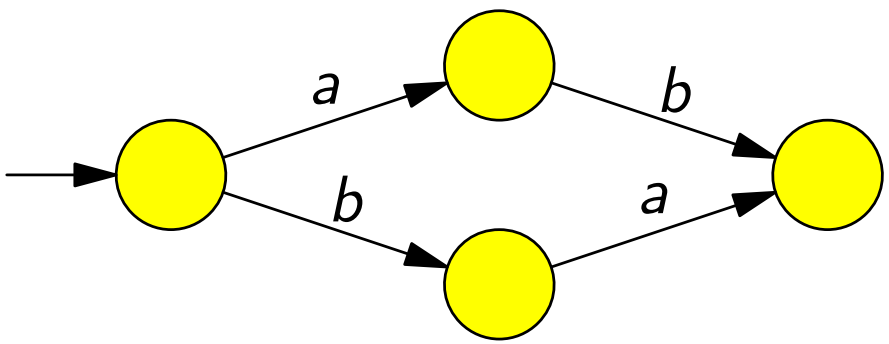
Wir definieren das synchronisierte Produkt $M = M' \circ M''$*:*

$M = (\Sigma, Q' \times Q'', \delta, (q_0', q_0''), F' \times F'')$ *mit* $\Sigma = \Sigma' \cup \Sigma''$ *und*

- $(q', p') \in \delta((q, p), a)$ *falls* $a \in \Sigma' \cap \Sigma''$ *und* $q' \in \delta'(q, a)$, $p' \in \delta''(p, a)$.
- $(q', p) \in \delta((q, p), a)$ *falls* $a \in \Sigma' \setminus \Sigma''$ *und* $q' \in \delta'(q, a)$.
- $(q, p') \in \delta((q, p), a)$ *falls* $a \in \Sigma'' \setminus \Sigma'$ *und* $p' \in \delta''(p, a)$.

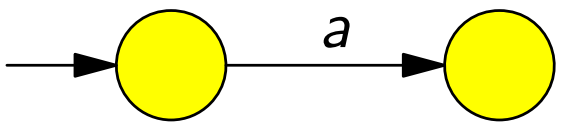
Beispiel

Was ist das synchronisierte Produkt?

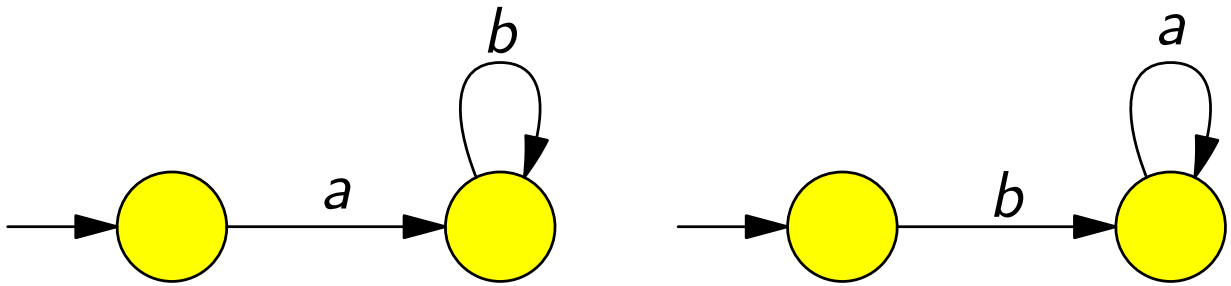


Beispiel

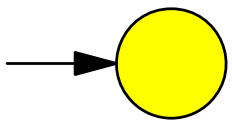
Was ist das synchronisierte Produkt?



Beispiel



Was ist das synchronisierte Produkt?



Unsynchronisiertes Produkt

Definition 5.1.2

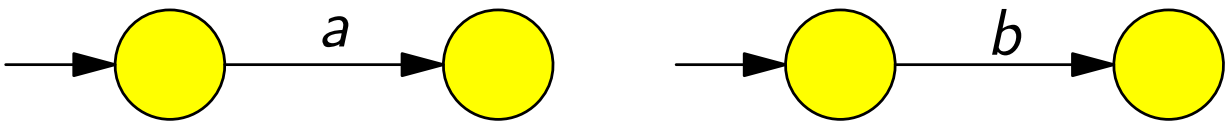
Es seien $M' = (\Sigma', Q', \delta', q_0', F')$ *und* $M'' = (\Sigma'', Q'', \delta'', q_0'', F'')$ *zwei NFA's.*

Wir definieren das unsynchronisierte Produkt $M = M' \sqcup M''$*:*

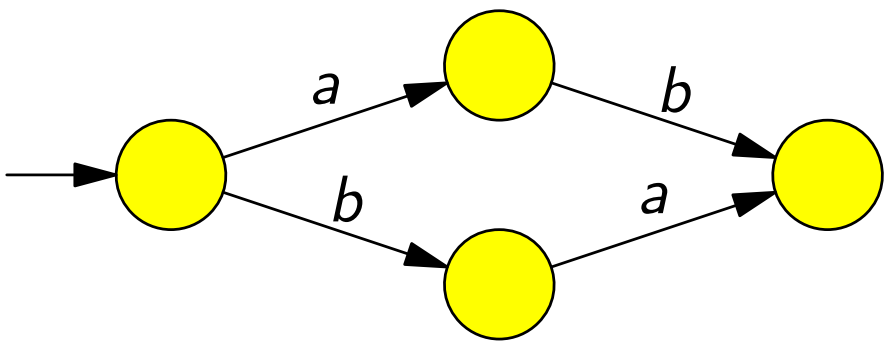
$M = (\Sigma, Q' \times Q'', \delta, (q_0', q_0''), F' \times F'')$ *mit* $\Sigma = \Sigma' \cup \Sigma''$ *und*

- $(q', p) \in \delta((q, p), a)$ *falls* $a \in \Sigma'$ *und* $q' \in \delta'(q, a)$.
- $(q, p') \in \delta((q, p), a)$ *falls* $a \in \Sigma''$ *und* $p' \in \delta''(p, a)$.

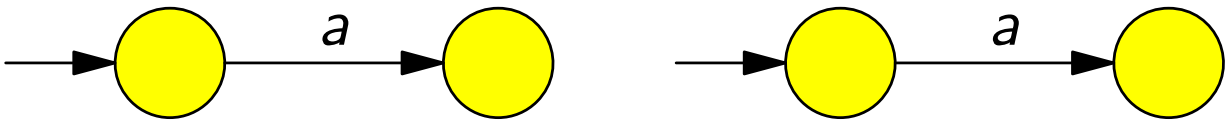
Beispiel



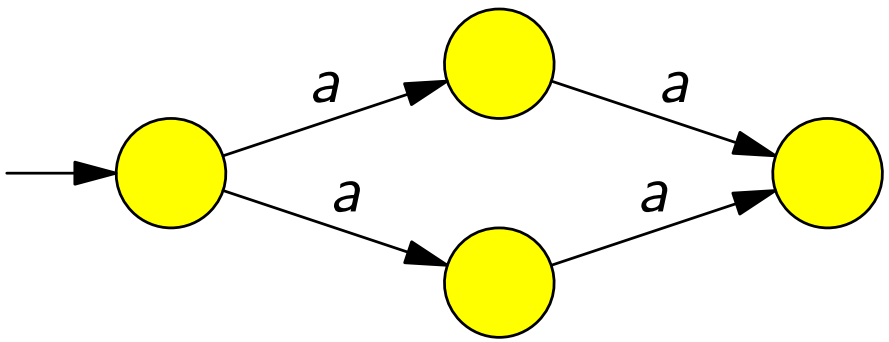
Was ist das unsynchronisierte Produkt?



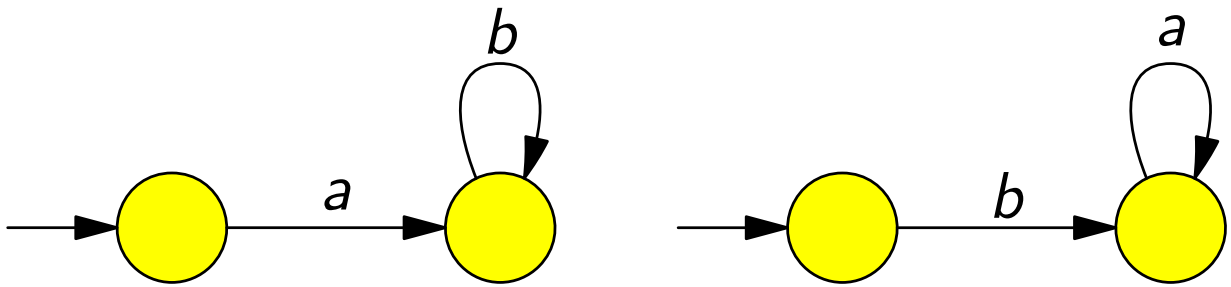
Beispiel



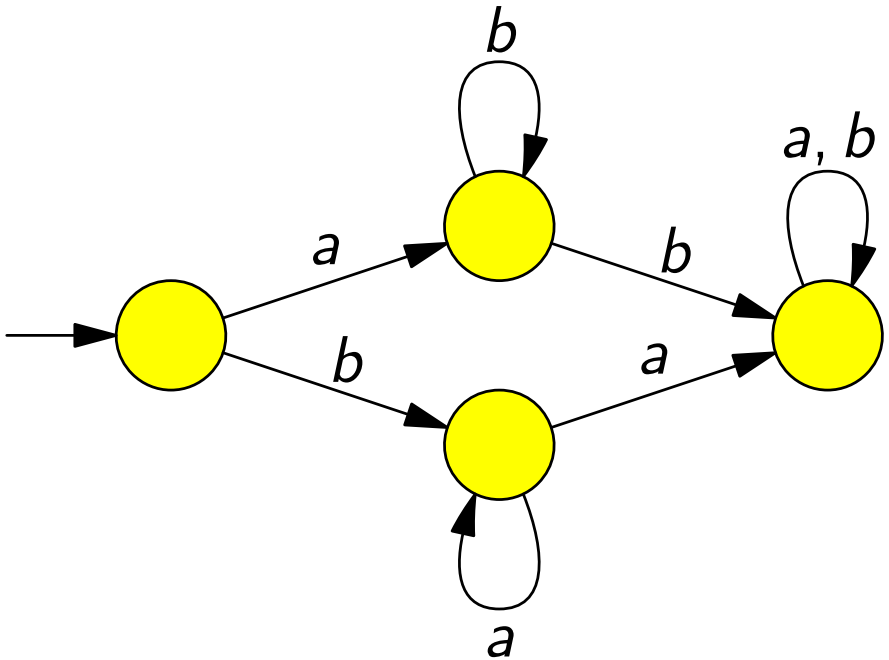
Was ist das unsynchronisierte Produkt?



Beispiel



Was ist das unsynchronisierte Produkt?



Das Mutual-Exclusion-Problem (Mutex)

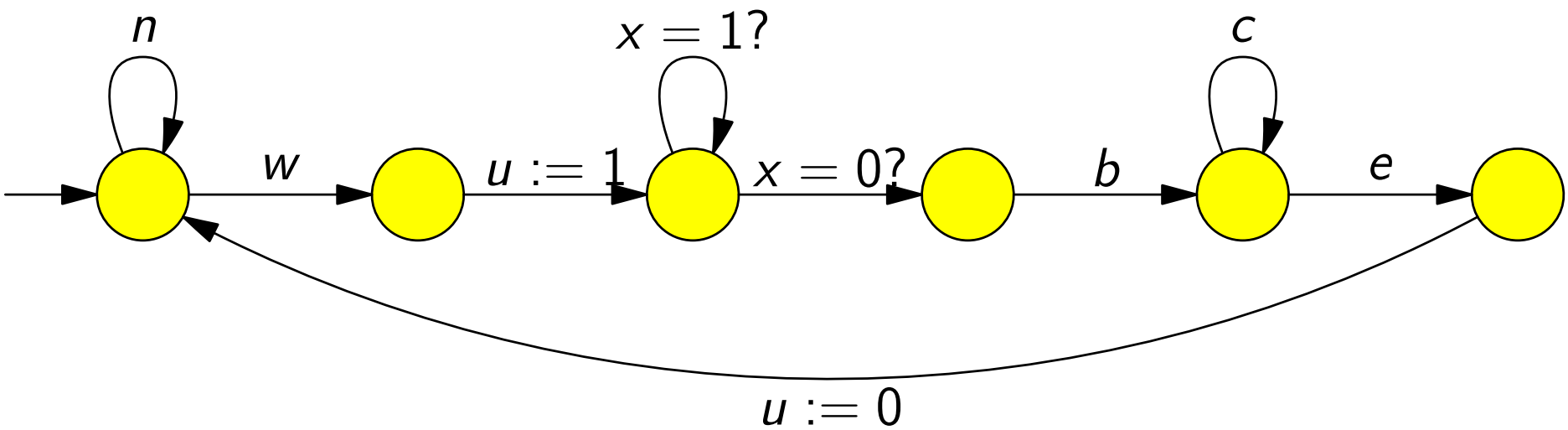
```
while(true) {
  /* non critical */
  u:=1;
  while(x=1) ;
  /* critical section */
  u:=0;
}
```

```
while(true) {
  /* non critical */
  x:=1;
  while(u=1) ;
  /* critical section */
  x:=0;
}
```

Nur ein Programm darf sich in der *critical section* befinden.

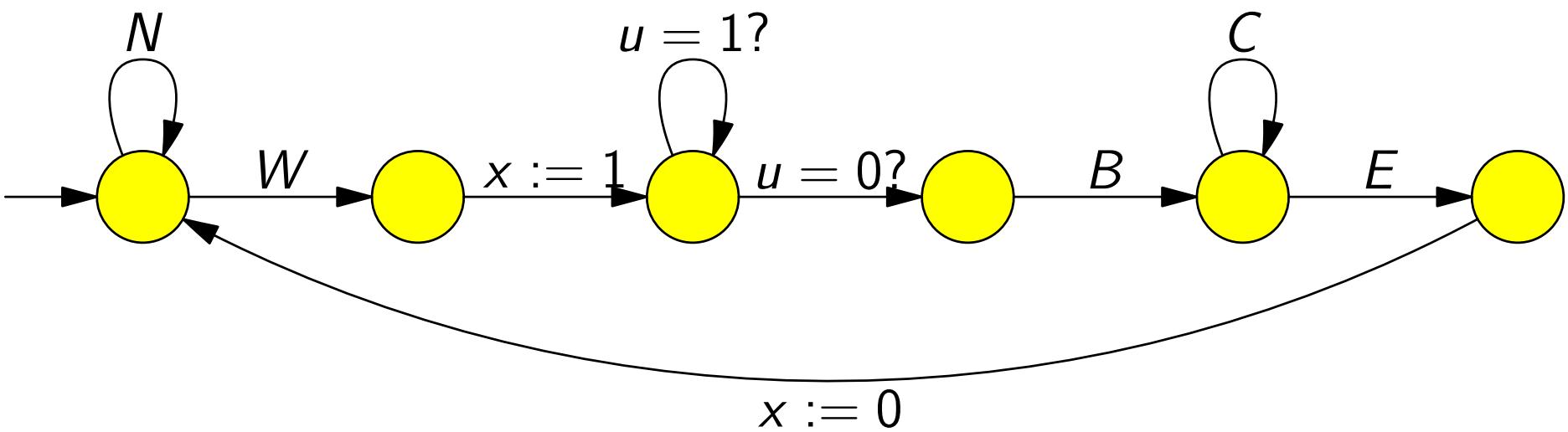
Modellierung des Kontrollflusses von Prozess P_0:

```
while(true) {
  /* non critical */
  u:=1;
  while(x=1) ;
  /* critical section */
  u:=0;
}
```

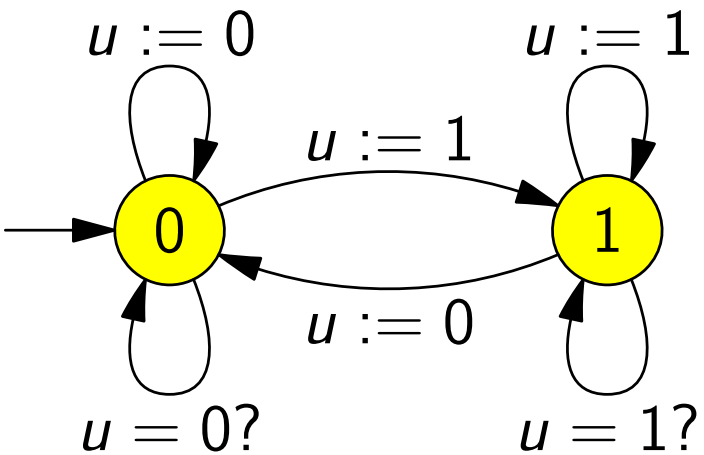


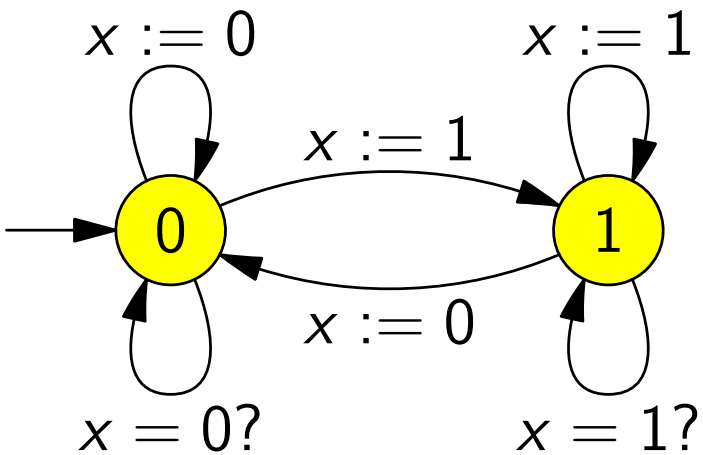
Modellierung des Kontrollflusses von Prozess P_1:

```
while(true) {
  /* non critical */
  x:=1;
  while(u=1) ;
  /* critical section */
  x:=0;
}
```



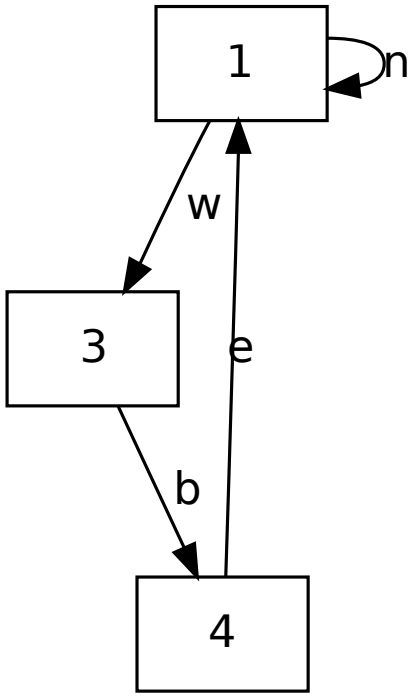
Modellierung der booleschen Variablen (B_u und B_x):

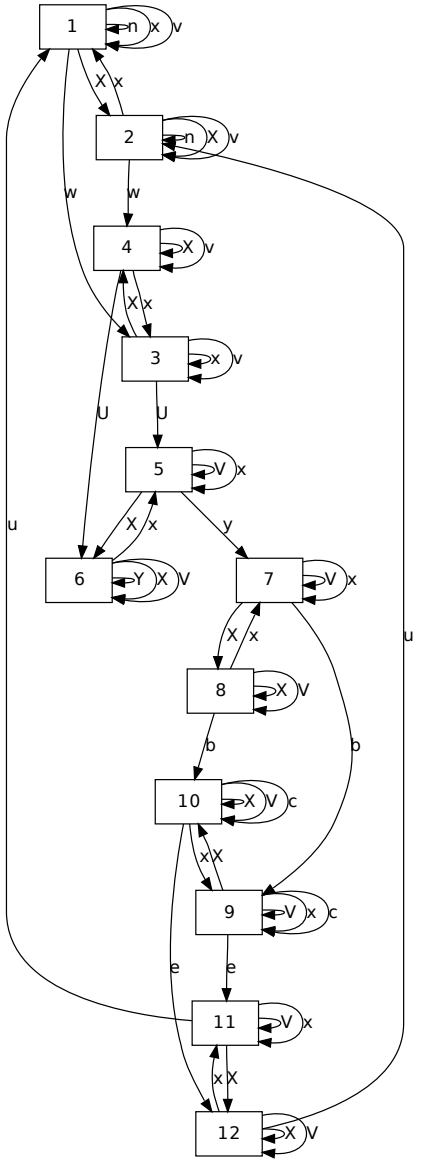




Was passiert, wenn nur P_0 läuft?

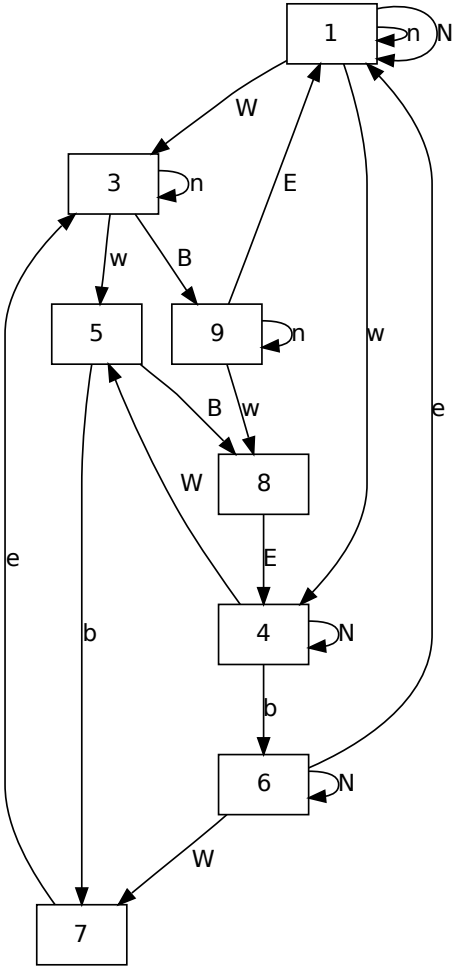
Betrachte $P_0 \circ B_u \circ B_x$.

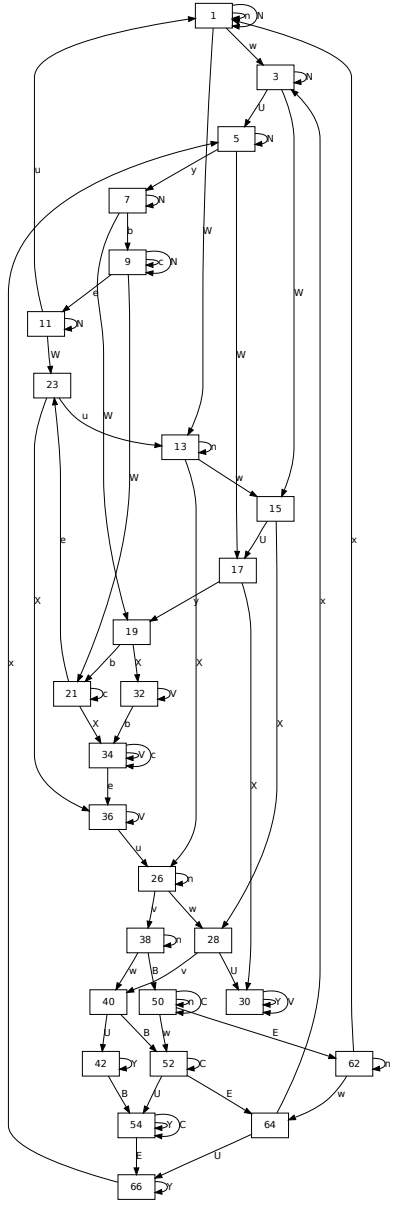


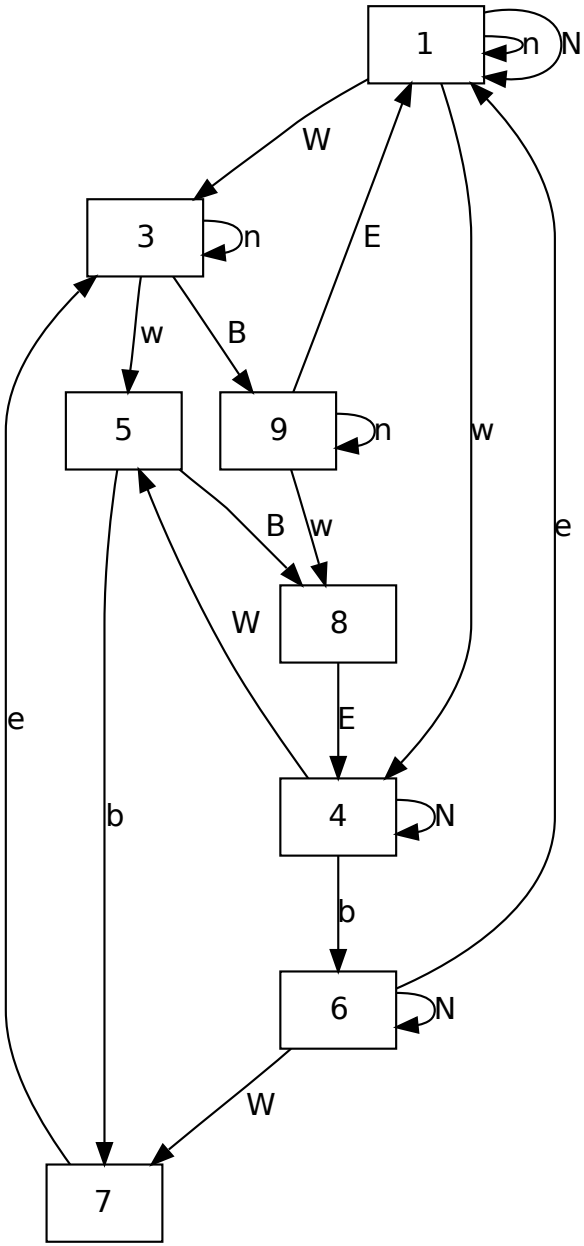


Was passiert, wenn P_0 und P_1 laufen?

Betrachte $(P_0 \sqcup\!\sqcup P_1) \circ B_u \circ B_x$.







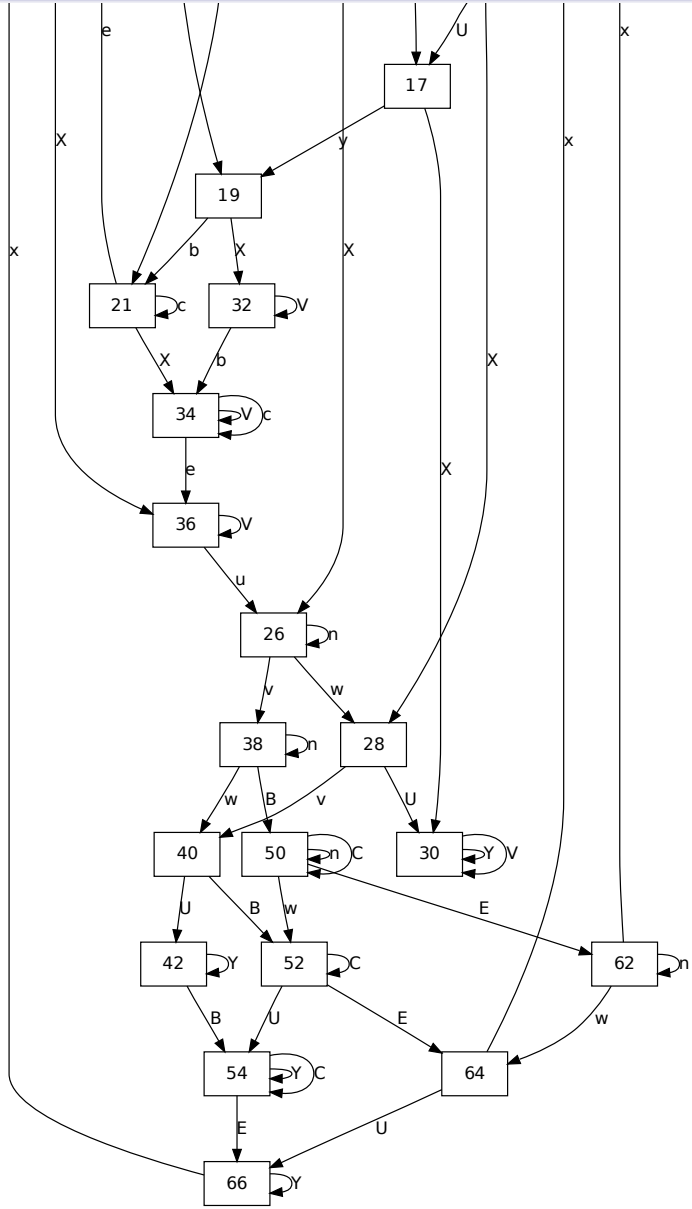
h ist ein Homomorphismus, der alle Symbole außer W, B, E, N, w, b, e und n löscht.

$L' = h(L(M))$ mit
$M = (P_0 \sqcup\!\sqcup P_1) \circ B_u \circ B_x$.

$$L' \cap \Sigma^* b(\Sigma \setminus \{e\})^* B \Sigma^* = \varnothing$$

$$L' \cap \Sigma^* B(\Sigma \setminus \{E\})^* b \Sigma^* = \varnothing$$

Also kann sich nur ein Prozess im kritischen Bereich befinden.



Problem: Deadlock!

Das Verfahren von Peterson

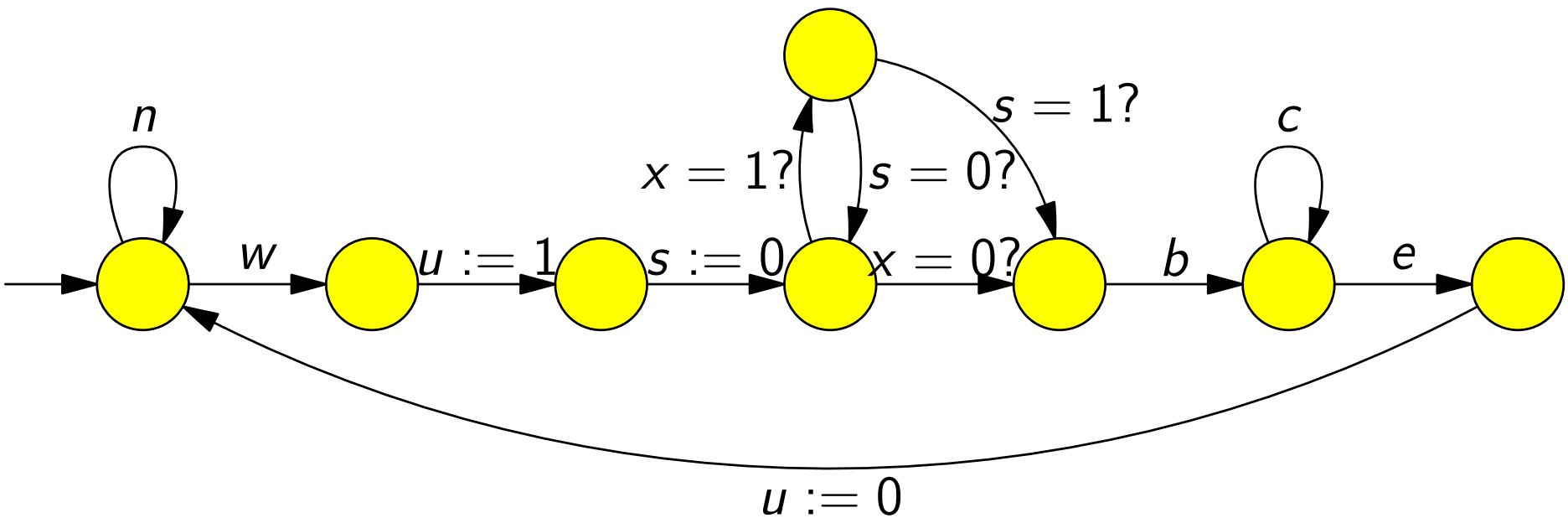
```
while(true) {
  /* non critical */
  u:=1;
  s:=0;
  while(x=1 ∧ s=0) ;
  /* critical section */
  u:=0;
}
```

```
while(true) {
  /* non critical */
  x:=1;
  s:=1;
  while(u=1 ∧ s=1) ;
  /* critical section */
  x:=0;
}
```

Nur ein Programm darf sich in der *critical section* befinden.

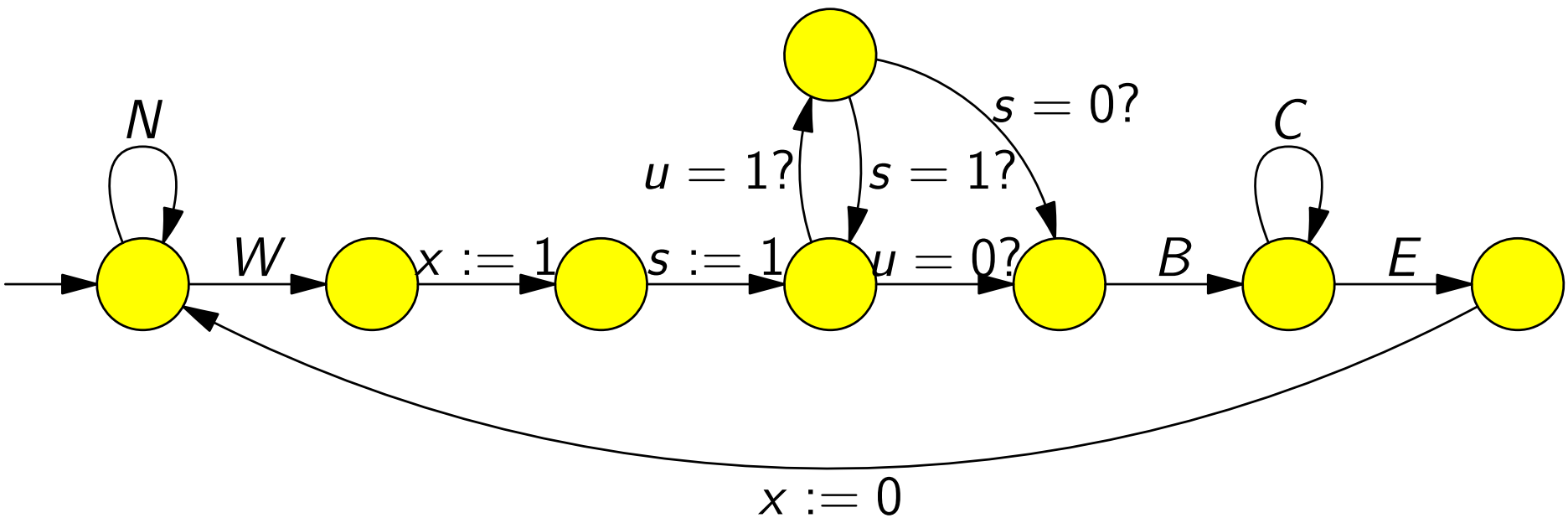
Modellierung des Kontrollflusses von Prozess P_0:

```
while(true) {
  /* non critical */
  u:=1;
  s:=0;
  while(x=1 ∧ s=0) ;
  /* critical section */
  u:=0;
}
```



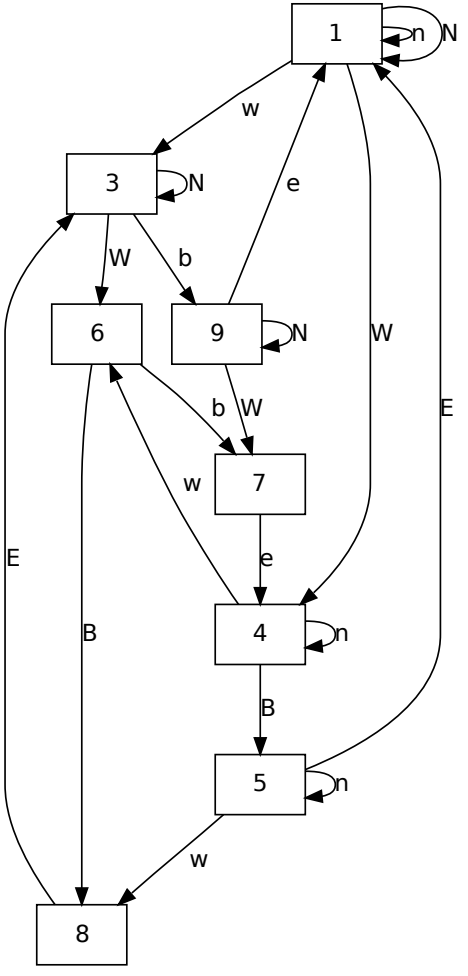
Modellierung des Kontrollflusses von Prozess P_1:

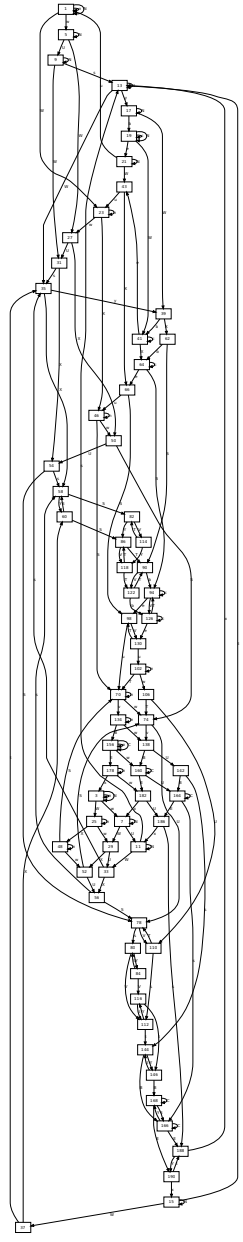
```
while(true) {
  /* non critical */
  x:=1;
  s:=1;
  while(u=1 ∧ s=1) ;
  /* critical section */
  x:=0;
}
```

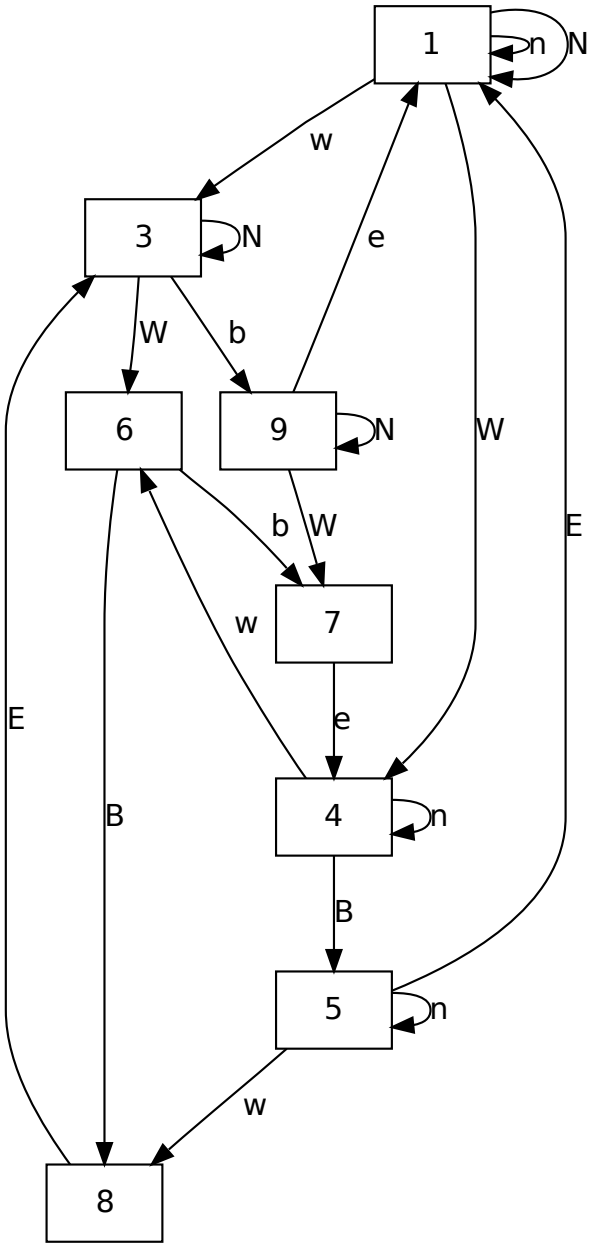


Was passiert, wenn P_0 und P_1 laufen?

Betrachte $(P_0 \sqcup\!\sqcup P_1) \circ B_u \circ B_x \circ B_s$.







h ist wieder ein Homomorphismus, der alle Symbole außer W, B, E, N, w, b, e und n löscht.

$L' = h(L(M))$ mit
$M = (P_0 \sqcup\!\sqcup P_1) \circ B_u \circ B_x \circ B_s$.

$$L' \cap \Sigma^* b(\Sigma \setminus \{e\})^* B \Sigma^* = \varnothing$$

$$L' \cap \Sigma^* B(\Sigma \setminus \{E\})^* b \Sigma^* = \varnothing$$

Also kann sich wieder nur ein Prozess im kritischen Bereich befinden.