Unsynchronisiertes Produkt

Definition

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

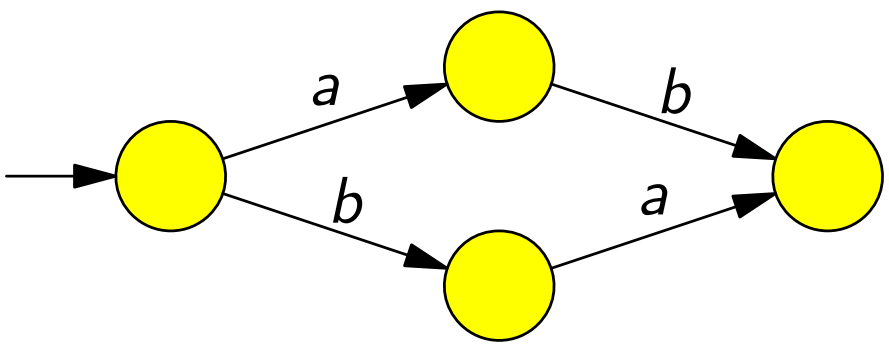
Wir definieren das *unsynchronisierte Produkt* $M = M' \sqcup\!\sqcup M''$:

$M = (\Sigma, Q' \times Q'', \delta, (q_0', q_0''), F' \times F'')$ 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 $p' \in \delta''(p, a)$.

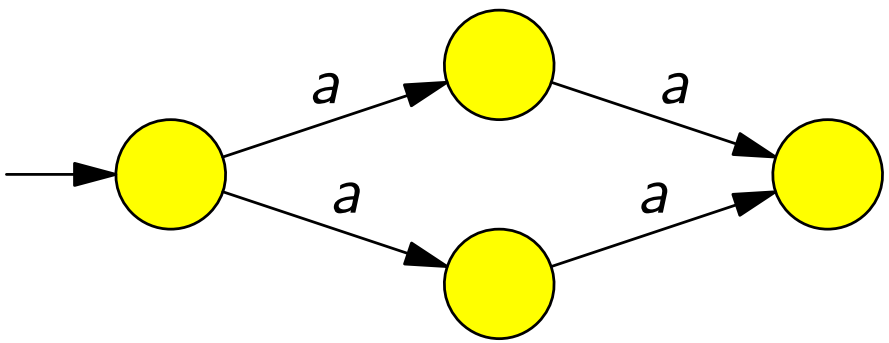
Beispiel

Was ist das unsynchronisierte Produkt?

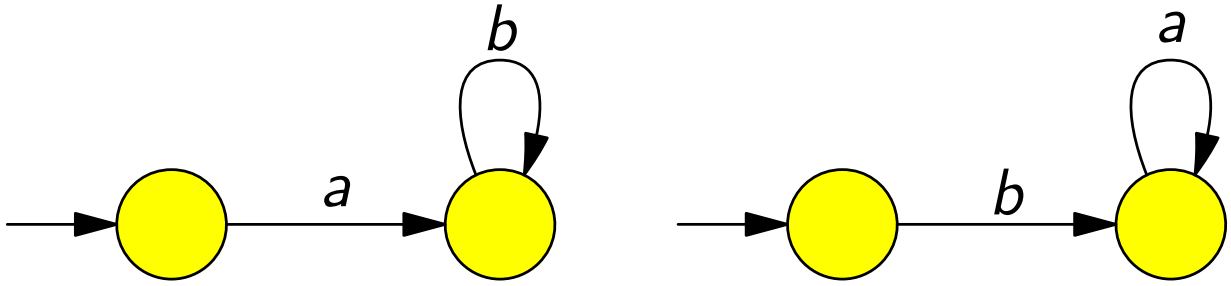


Beispiel

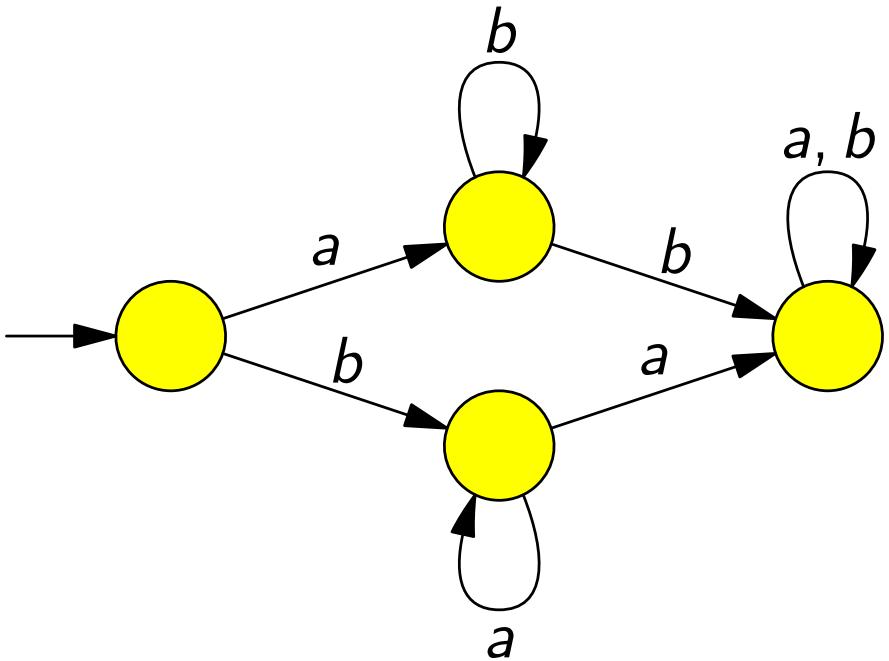
Was ist das unsynchronisierte Produkt?



Beispiel



Was ist das unsynchronisierte Produkt?



Das Mutual-Exclusion-Problem (Mutex)

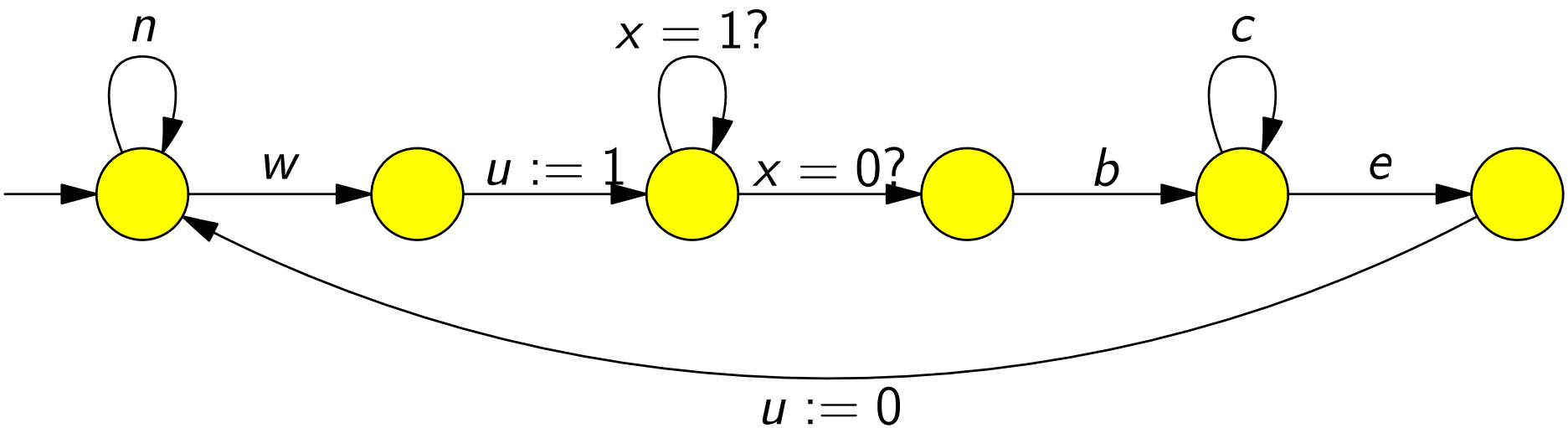
```
while(true) {
  /* non critical */
  u:=1;
  while(x=1) ;
  /* critical section */
  u:=0;
}
```

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

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

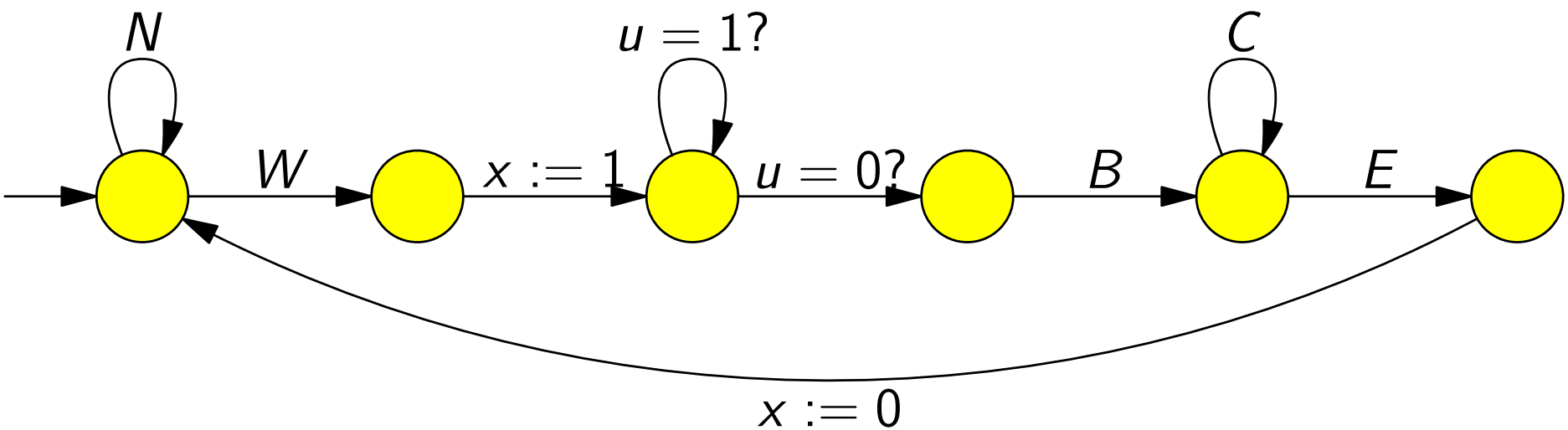
Modellierung des Prozesses P_0:

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

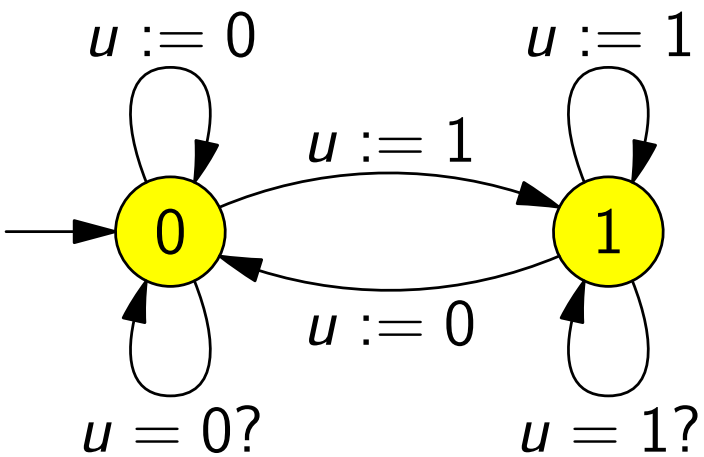


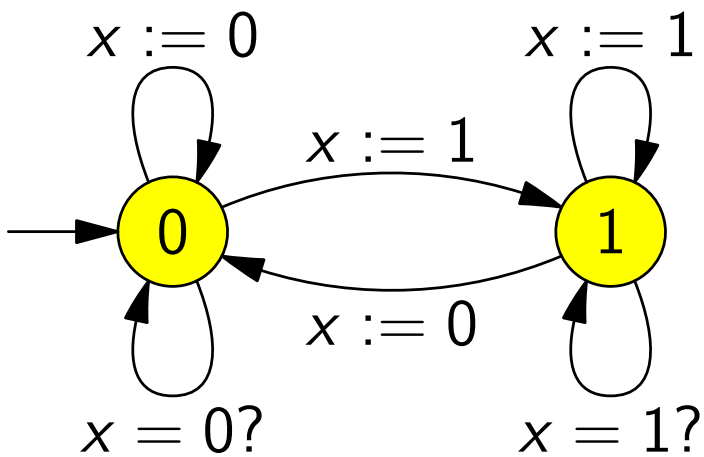
Modellierung des Prozesses P_1:

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



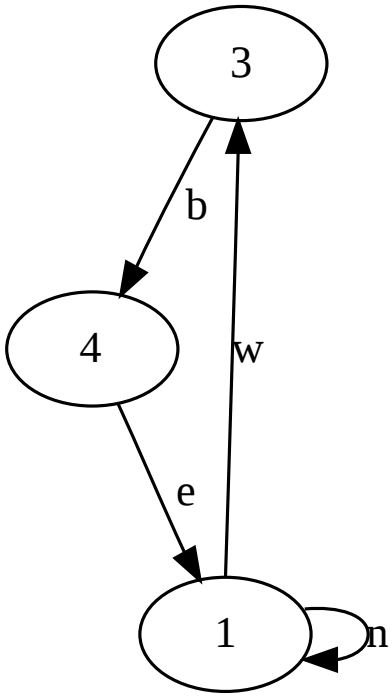
Modellierung der booleschen Variablen (B_u und B_x):

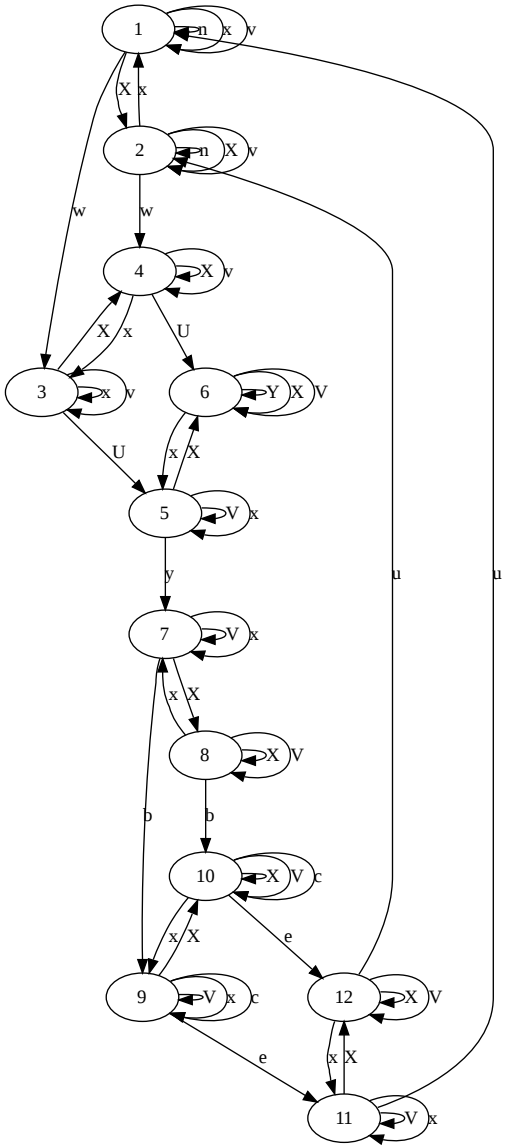




Was passiert, wenn nur P_0 läuft?

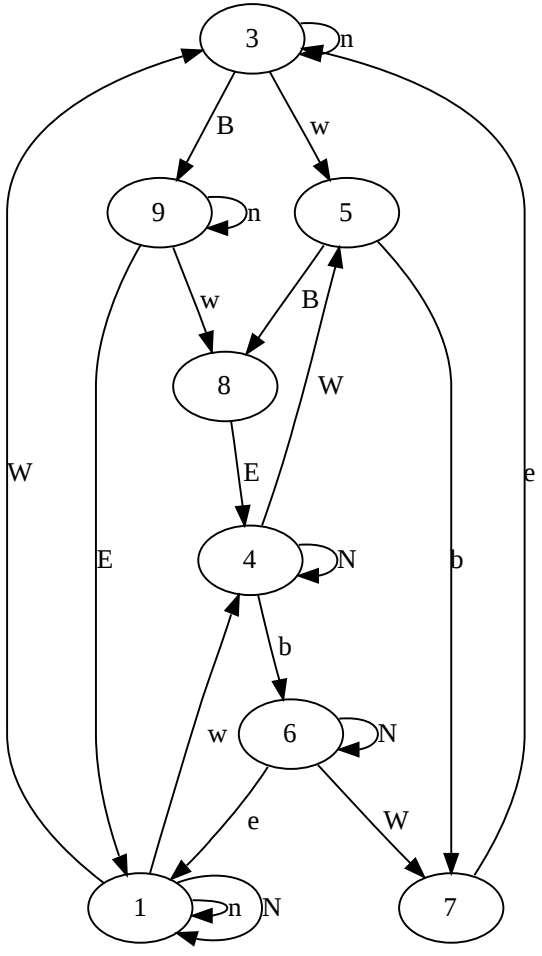
Betrachte $P_0 \circ B_u \circ B_x$.

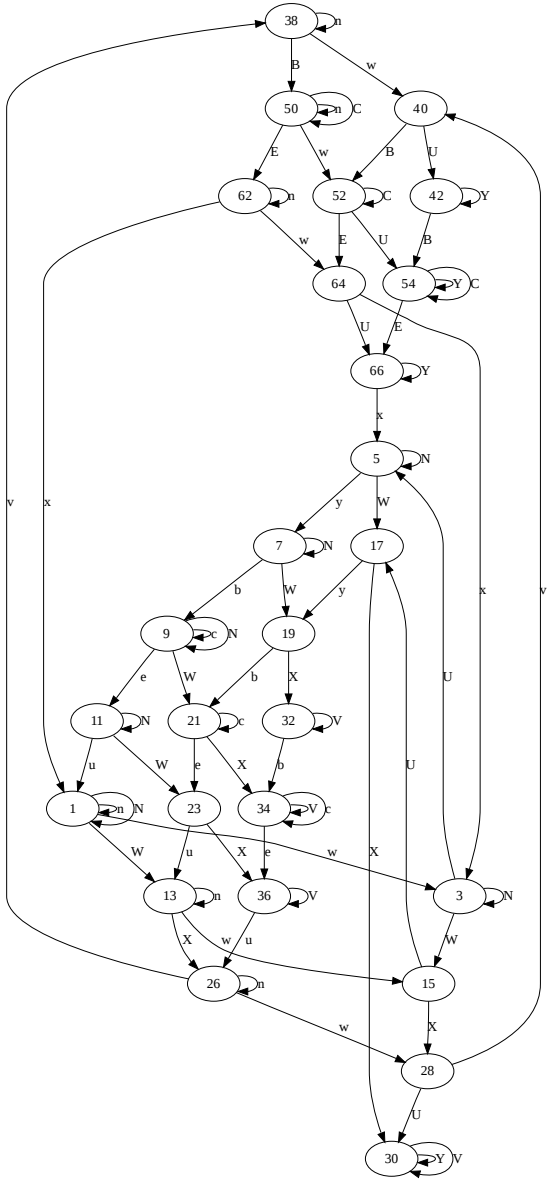


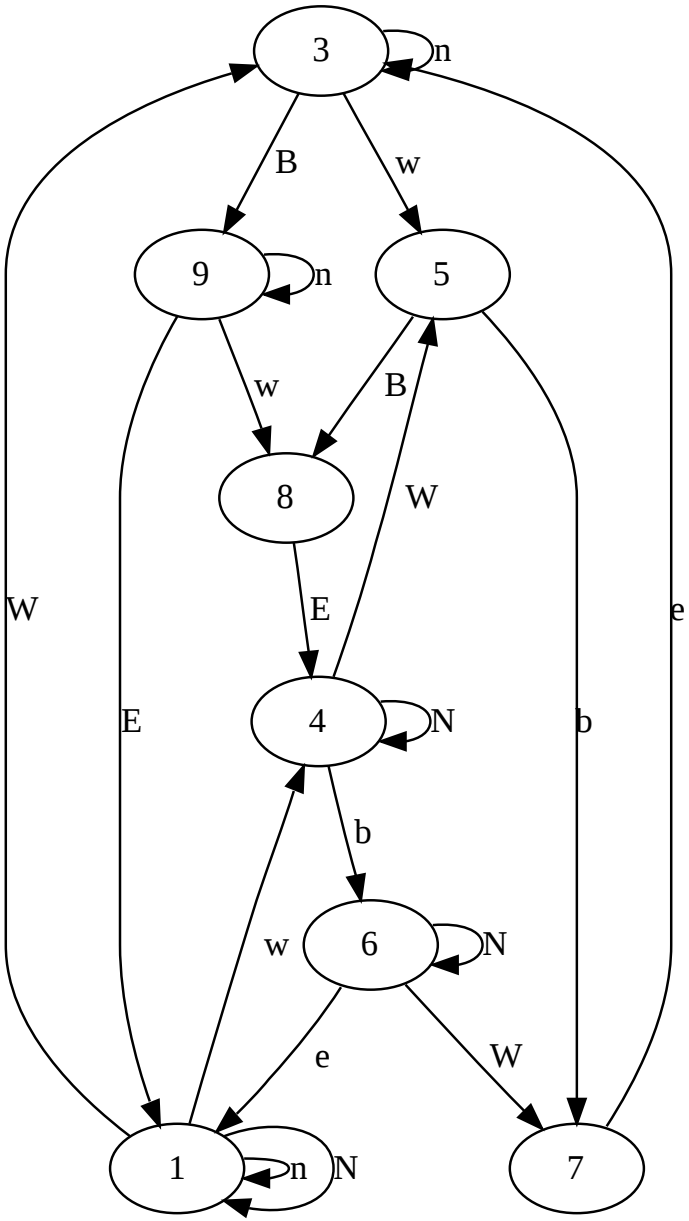


Was passiert, wenn P_0 und P_1 laufen?

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







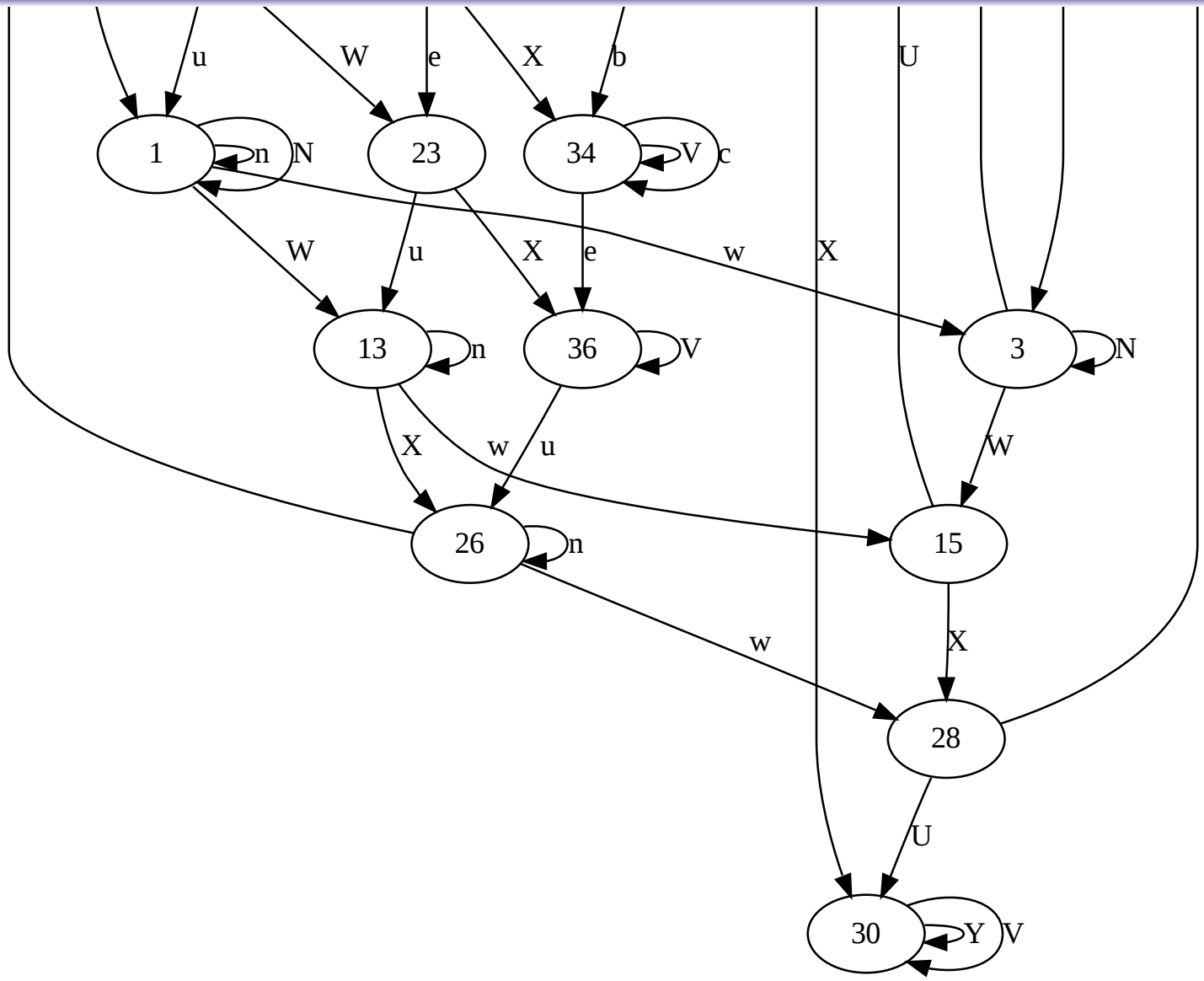
h ist ein Homomorphismus, der alle Symbole außer W, B, E, N, w, b, e und n löscht.

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

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

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

Also kann sich nur ein Prozeß im kritischen Bereich befinden.



Problem: Deadlock!

Das Verfahren von Peterson

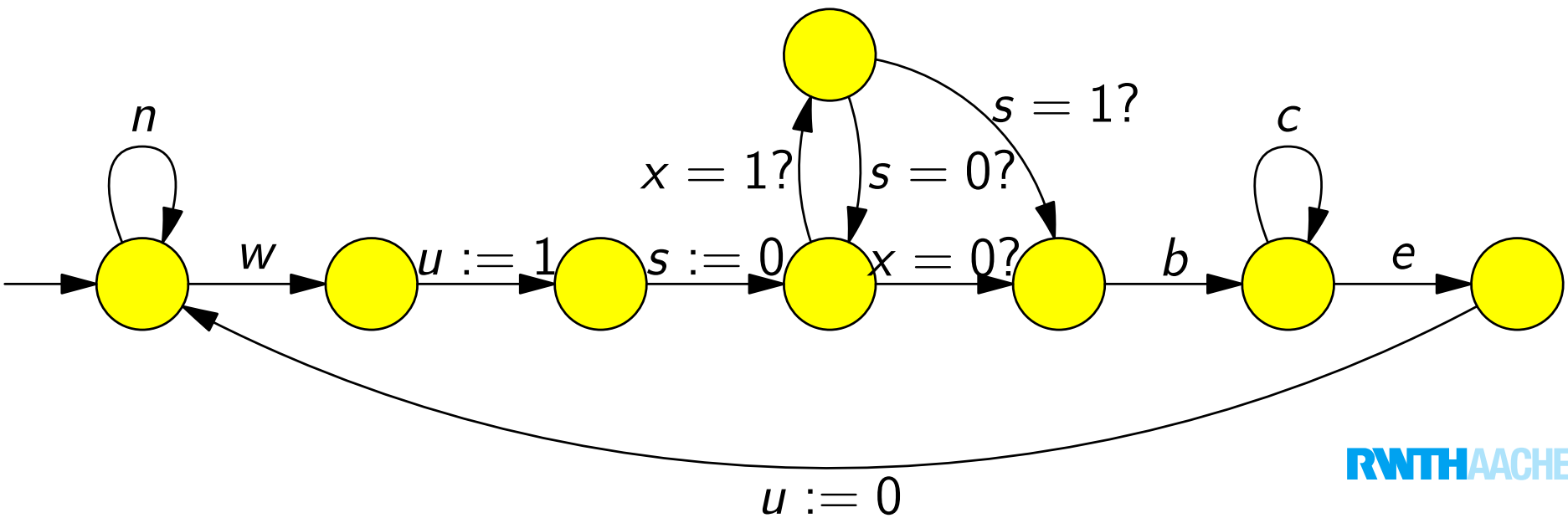
```
while(true) {
  /* non critical */
  u:=1;
  s:=0;
  while(x=1 ∧ s=0) ;
  /* critical section */
  u:=0;
}
```

```
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.

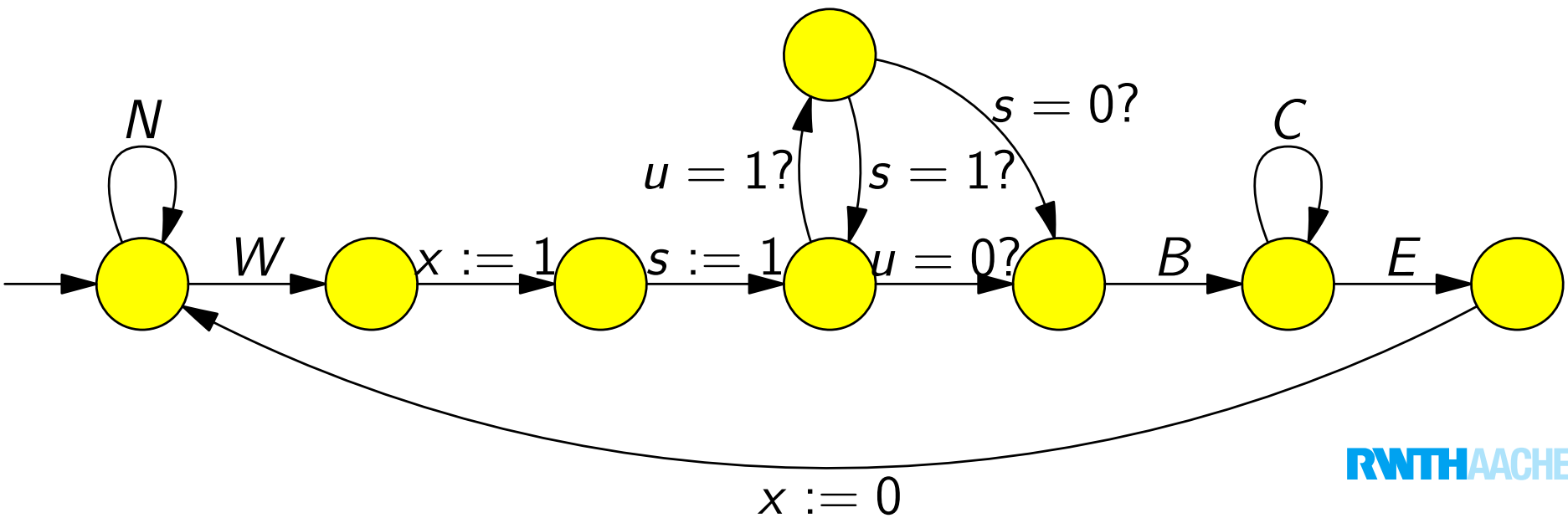
Modellierung des Prozesses P_0:

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



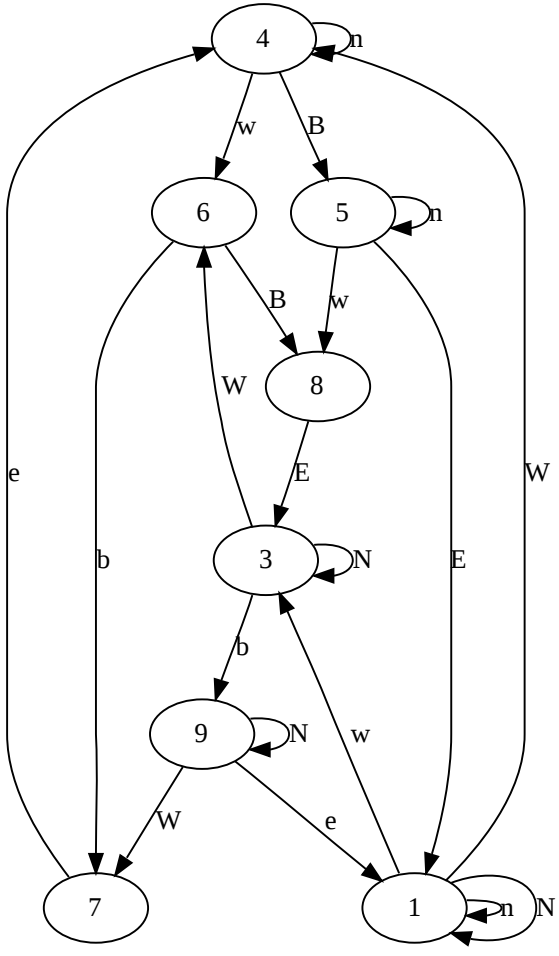
Modellierung des Prozesses P_1:

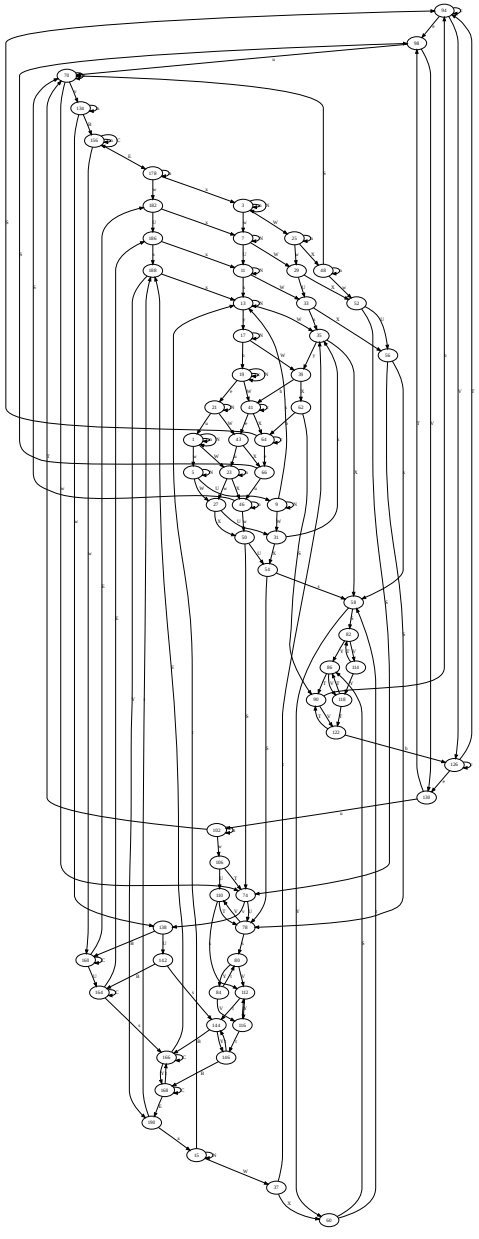
```
while(true) {
  /* non critical */
  x:=1;
  s:=1;
  while(u=1 ∧ s=1) ;
  /* critical section */
  x:=0;
}
```

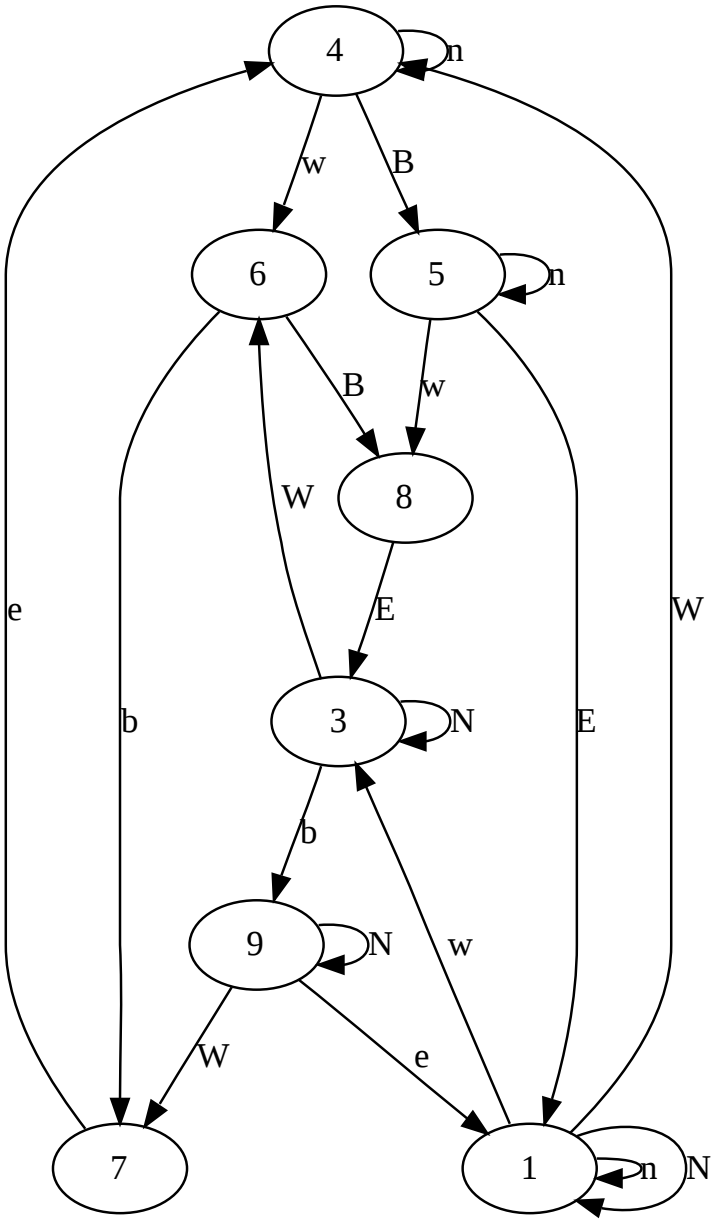


Was passiert, wenn P_0 und P_1 laufen?

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

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

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

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

Aber: Problem des Verhungerns!