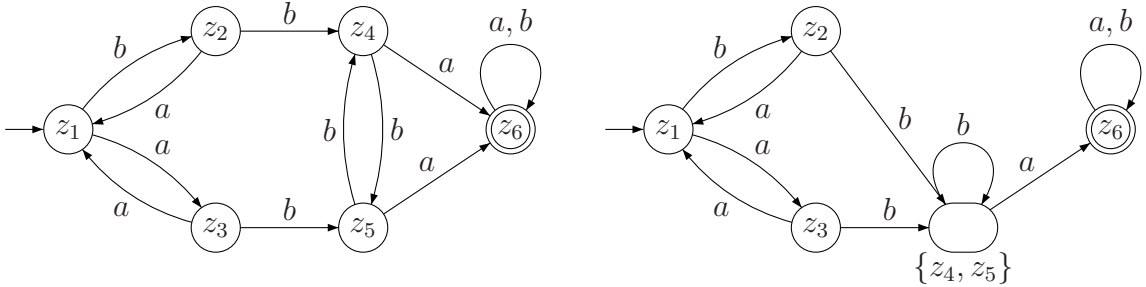
2.5 Minimale Automaten

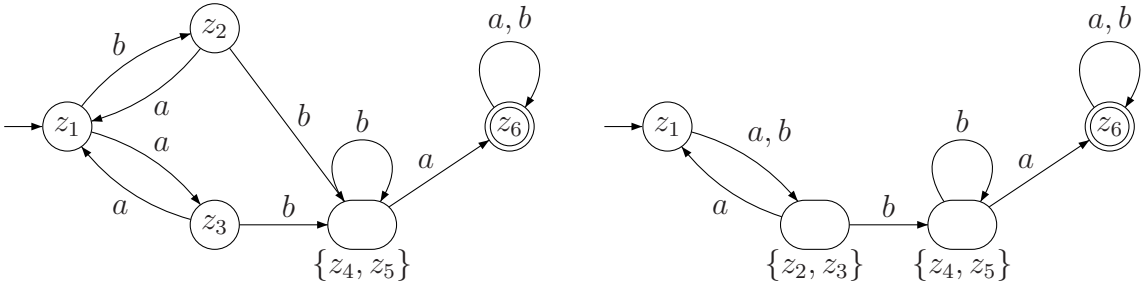
Beispiel 2.17.



Betrachte linken DFA M_1. Dann gilt für alle Wörter $w \in \{a, b\}^*$:

$$\delta(z_4, w) \in F \iff |w|_a > 0 \iff \delta(z_5, w) \in F$$

Es ist also egal, ob der Automat sich im Zustand z_4 oder im Zustand z_5 befindet. Daher akzeptiert M_1 dieselbe Sprache wie der rechte DFA M_2, in dem wir die „äquivalenten“ Zustände z_4 und z_5 „verschmolzen“ haben.



In diesem DFA gilt für alle $w \in \{a, b\}^* \setminus \{\varepsilon\}$: $\delta(z_2, w) = \delta(z_3, w)$ und daher

$$\delta(z_2, w) \in F \iff \delta(z_3, w) \in F\,.$$

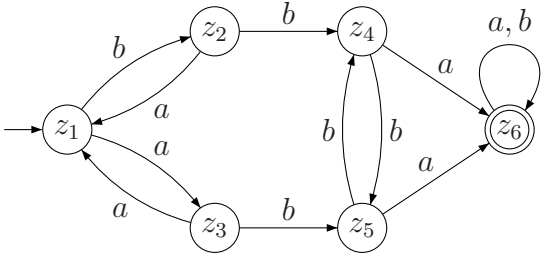
Es ist also egal, ob der Automat sich im Zustand z_2 oder im Zustand z_3 befindet. Daher akzeptiert M_2 dieselbe Sprache wie der rechte DFA M_3, in dem wir die „äquivalenten“ Zustände z_2 und z_3 „verschmolzen“ haben. In diesem DFA sind keine 2 Zustände mehr „äquivalent“, er kann also nicht weiter verkleinert werden.

Systematisches Vorgehen Um festzustellen, daß 2 Zustände z und z' „äquivalent“ sind, muß man überprüfen, ob $\delta(z,w) \in F \iff \delta(z',w) \in F$ für alle Wörter w gilt. Da dies eine unendliche Bedingung ist, ist nicht klar, wie sie algorithmisch gelöst werden kann.
Wir gehen hierzu wie folgt vor:
Eingabe: DFA $M = (Z, \Sigma, \iota, \delta, F)$, in dem alle Zustände von ι aus erreichbar sind
Ausgabe: Menge der Paare (z, z') von Zuständen aus Z mit
$\delta(z,w) \in F \iff \delta(z',w) \in F$ für alle $w \in \Sigma^*$

1. Stelle eine Tabelle aller ungeordneten Zustandspaare $\{z, z'\}$ mit $z \neq z'$ auf.
2. Markiere alle Paare $\{z, z'\}$ mit $z \in F$ und $z' \notin F$.
3. Markiere ein unmarkiertes Paar $\{z, z'\}$, für das es ein $a \in \Sigma$ gibt, so daß das Paar $\{\delta(z,a), \delta(z',a)\}$ bereits markiert ist.
4. Wiederhole den vorherigen Schritt, bis keine neuen Markierungen entstehen.
5. Gib alle Paare $\{z, z'\}$ aus, die nicht markiert sind.

Der DFA M' wird jetzt berechnet, indem die ausgegebenen Zustandspaare „verschmolzen“ werden.

Beispiel 2.17 (Fortsetzung)



z_2	9				
z_3	8				
z_4	7	2	5		
z_5	6	4	3		
z_6	1	1	1	1	1
	z_1	z_2	z_3	z_4	z_5

Ausgegeben werden also die Paare $\{z_2, z_3\}$ und $\{z_4, z_5\}$. Das „Verschmelzen“ dieser Zustände im DFA M_1 liefert den DFA M_3 aus Beispiel 2.17.

Satz 2.18. *Sei M ein DFA und M' der nach obigem Algorithmus berechnete DFA.*

1. *$L(M) = L(M')$*
2. *Es gibt keinen DFA M'' mit weniger Zuständen als M' und $L(M) = L(M'')$.*
3. *Jeder DFA M'' mit $L(M) = L(M'')$ und genauso vielen Zuständen wie M' ist isomorph zu M', d.h. „gleich bis auf Umbenennung der Zustände“.*

Daher heißt der berechnete DFA M' auch *Minimalautomat*.

Um festzustellen, ob zwei DFAs M_1 und M_2 dieselbe Sprache erkennen, können wir also wie folgt vorgehen:

1. berechne die Minimalautomaton M_1' und M_2'
2. teste, ob diese isomorph sind, d.h. „gleich bis auf Umbenennung der Zustände“

2.6 Analyse regulärer Sprachen

Wir diskutieren Verfahren, die die folgenden Fragestellungen bzw. Probleme für reguläre Sprachen entscheiden. Dabei nehmen wir an, daß reguläre Sprachen als DFAs, NFAs oder reguläre Ausdrücke gegeben sind.

- *Wortproblem:* Gilt $w \in L$ für eine gegebene reguläre Sprache L und $w \in \Sigma^*$?
- *Leerheitsproblem:* Gilt $L = \emptyset$ für eine gegebene reguläre Sprache L?
- *Endlichkeitsproblem:* Ist eine gegebene reguläre Sprache L endlich?
- *Disjunktheitsproblem:* Gilt $L_1 \cap L_2 = \emptyset$ für gegebene reguläre L_1, L_2?
- *Inklusionsproblem:* Gilt $L_1 \subseteq L_2$ für gegebene reguläre L_1, L_2?
- *Äquivalenzproblem:* Gilt $L_1 = L_2$ für gegebene reguläre L_1, L_2?

Wortproblem (DFA)

Eingabe: DFA $M = (Z, \Sigma, \iota, \delta, F)$ und $w \in \Sigma^*$
Frage: $w \in L(M)$?

Verfahren:
Sei $w = a_1 a_2 \cdots a_n$ mit $a_i \in \Sigma$.
Verfolge die Zustandsübergänge von M, die durch die Symbole $a_1, \ldots, a_n$ vorgegeben sind:

$z := \iota$
for $i := 1$ **to** n **do**
  $z := \delta(z, a_i)$
endfor
if $z \in F$ **then return**(JA) **else return**(NEIN)

Zeitbedarf: polynomiell in M und w

Wortproblem (NFA)

Eingabe: NFA $M = (Z, \Sigma, I, \delta, F)$ und $w \in \Sigma^*$
Frage: $w \in L(M)$?

Verfahren:
man könnte aus dem NFA M einen äquivalenten DFA M' berechnen und dann obiges Verfahren anwenden. Der 1. Schritt verlange Zeit exponentiell in M.

alternatives Verfahren:
Sei $w = a_1 a_2 \cdots a_n$ mit $a_i \in \Sigma$.
Berechne induktiv die Menge S von Zuständen, in denen sich der NFA nach Lesen von $a_1 a_2 \cdots a_i$ befinden kann:

$S := I$
for $i := 1$ **to** n **do**
  $S := \{z' \in Z \mid \exists z \in S \colon z' \in \delta(z, a_i)\}$
endfor
if $S \cap F \neq \emptyset$ **then return**(JA) **else return**(NEIN)

Zeitbedarf: polynomiell in M und w

Leerheitsproblem

Eingabe: NFA $M = (Z, \Sigma, I, \delta, F)$.
Frage: $L(M) = \emptyset$?

Verfahren:
Sei $G = (Z, \to)$ der gerichtete Graph mit

$$z \to z' \iff \exists a \in \Sigma : z' \in \delta(z, a)\,.$$

Dann gilt: $L(M) \neq \emptyset$ genau dann, wenn es in dem Graphen G einen (evtl. leeren) Pfad von einem Knoten aus I zu einem Knoten aus F gibt.
Dies kann z.B. mit dem Algorithmus' von Dijkstra entschieden werden.

Zeitbedarf: polynomiell in M

Endlichkeitsproblem

Eingabe: NFA $M = (Z, \Sigma, I, \delta, F)$.
Frage: Ist $L(M)$ endlich?

Verfahren:
Sei $G = (Z, \to)$ wieder der gerichtete Graph mit

$$z \to z' \iff \exists a \in \Sigma : z' \in \delta(z, a)\,.$$

Dann gilt: $L(M)$ ist genau dann unendlich, wenn es $z \in Z$, $z_0 \in I$ und $z_1 \in F$ gibt mit $z_0 \to^* z \to^+ z \to^* z_1$, d.h. z liegt auf einem Zyklus, ist von einem Startzustand aus erreichbar und von z kann ein Endzustand erreicht werden.
Dies kann wieder mit dem Algorithmus von Dijkstra entschieden werden.

Zeitbedarf: polynomiell in M

Disjunktheitsproblem

Eingabe: NFAs M_1 und M_2
Frage: Gilt $L(M_1) \cap L(M_2) = \emptyset$?

Verfahren:
Konstruiere aus M_1 und M_2 einen NFA M mit $L(M) = L(M_1) \cap L(M_2)$ (mgl. nach Folgerung 2.6 in polynomieller Zeit mittels Kreuzproduktkonstruktion).
Teste, ob $L(M) = \emptyset$ gilt.

Zeitbedarf: polynomiell in M_1 und M_2, da M nur polynomiell groß sein kann

Inklusionsproblem (NFAs)

Eingabe: NFAs M_1 und M_2.
Frage: Gilt $L(M_1) \subseteq L(M_2)$.

Verfahren:

	Zeitbedarf
1. berechne DFA M_2' mit $L(M_2') = L(M_2)$	exponentiell in M_2
2. berechne DFA M_2'' mit $L(M_2'') = \Sigma^* \setminus L(M_2')$	polynomiell in M_2'
3. berechne NFA M mit $L(M) = L(M_2'') \cap L(M_1)$	polynomiell in M_2'' und M_1
4. entscheide, ob $L(M) = \emptyset$	polynomiell in M

Dieses Verfahren ist korrekt, denn es gelten

$$L(M_1) \subseteq L(M_2) \iff L(M_1) \setminus L(M_2) = \emptyset$$

und

$$L(M_1) \setminus L(M_2) = L(M_1) \cap \Sigma^* \setminus L(M_2) = L(M_2'') \cap L(M_1) = L(M)$$

Zeitbedarf: exponentiell in M_2 und polynomiell in M_1

Inklusionsproblem (NFA, DFA)

Eingabe: NFA M_1 und DFA M_2.
Frage: Gilt $L(M_1) \subseteq L(M_2)$.

Verfahren:
verwende obiges Verfahren, wobei wir $M_2' = M_2$ setzen können

Zeitbedarf: polynomiell in M_1 und M_2

Äquivalenzproblem (NFAs)

Eingabe: NFAs M_1 und M_2.
Frage: Gilt $L(M_1) = L(M_2)$?

Verfahren 1:
Es gilt: $L(M_1) = L(M_2)$ genau dann, wenn $L(M_1) \subseteq L(M_2)$ und $L(M_2) \subseteq L(M_1)$.

Verfahren 2:

1. berechne DFAs mit $L(M_1') = L(M_1)$ und $L(M_2') = L(M_2)$
2. berechne minimale DFAs mit $L(M_1'') = L(M_1')$ und $L(M_2'') = L(M_2')$
3. teste, ob M_1'' und M_2'' isomorph sind, d.h. durch Umbenennung der Zustände ineinander überführt werden können

Dieses Verfahren ist korrekt nach Satz 2.18.

Zeitbedarf für beide Verfahren: exponentiell in M_1 und in M_2
polynomiell, wenn M_1 und M_2 DFAs sind

2.7 Eine Anwendung: Verifikation

Ein abschließendes ausführliches Beispiel:

- Wir betrachten zwei Prozesse P_1 und P_2, die auf eine gemeinsame Ressource zugreifen wollen.
- Jeder Prozeß hat einen sogenannten kritischen Bereich, in dem auf die Ressource zugegriffen wird. Es darf sich also jeweils nur ein Prozeß im kritischen Bereich befinden.
- Es stehen gemeinsame Variable zur Verfügung, über die sich die Prozesse synchronisieren können. Diese Variablen sind jedoch keine Semaphore, d.h. eine atomare Operation, bei der gleichzeitig gelesen und geschrieben wird, ist nicht möglich.

Wir möchten zeigen, daß der wechselseitige Ausschluß gewährleistet ist und daß gewisse Fairneßbedingungen (jeder Prozeß kommt irgendwann an die Reihe) eingehalten werden.

Was hat das mit formalen Sprachen zu tun?

- Jeder Ablauf eines Prozesses ist ein Wort, die Menge der Abläufe ist also eine Sprache.
- Ebenso ist die Menge der Abläufe des Gesamtsystems *Sys* eine Sprache L_{Sys}.
- Und auch die Menge der erlaubten bzw. verbotenen Abläufe ist eine Sprache L_{Spec}.

Damit ist also das Inklusionsproblem „$L_{Sys} \subseteq L_{Spec}$?“ bzw. das Disjunktheitsproblem „$L_{Sys} \cap L_{Spec} = \emptyset$?“ zu lösen.

Wir haben gesehen: Sind beide Sprachen regulär, so ist dies algorithmisch machbar!

2.7.1 Erster Versuch:

Die Prozesse P_1, P_2 verwenden eine gemeinsame Boolesche Variable `f`, die mit `false` initialisiert wird.

Programmcode für P_1, P_2

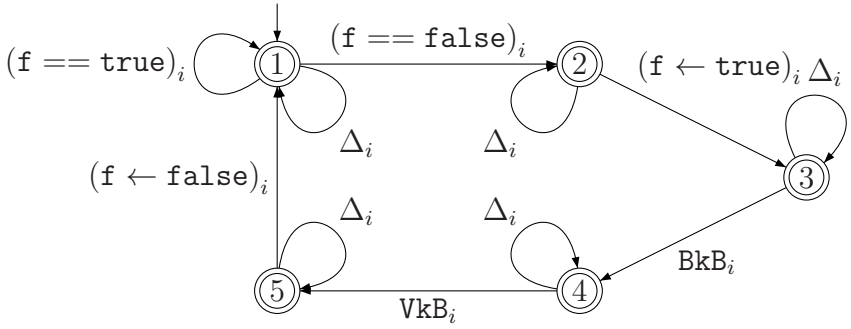
```
   while true do
1: if (f == false) then do
     begin
2:     f ← true
3:     [Betrete kritischen Bereich]
4:     [Verlasse kritischen Bereich]
5:     f ← false
     endif
   enddo
```

Das folgende Alphabet Σ besteht aus den Programm-Befehlen und den Abfragen der Booleschen Variablen:

$$\begin{aligned}
&\{(\texttt{f} \leftarrow \texttt{true})_i, (\texttt{f} \leftarrow \texttt{false})_i\} \\
\cup\ &\{\texttt{BkB}_i, \texttt{VkB}_i \mid i \in \{1,2\}\} \\
&\qquad \text{(Prozeß } i \text{ betritt/verläßt kritischen Bereich)} \\
&\{(\texttt{f} == \texttt{false})_i, (\texttt{f} == \texttt{true})_i \mid i \in \{1,2\}\} \\
&\qquad \text{(Synchronisation von Prozeß } i \text{ mit Variable } \texttt{f})
\end{aligned}$$

Der Index $i \in \{1,2\}$ gibt an, ob die jeweilige Aktion vom ersten oder vom zweiten Prozeß ausgeführt wird.

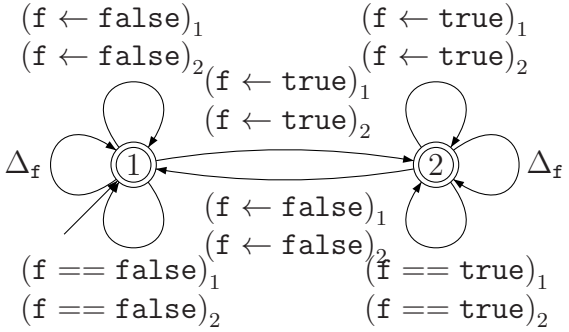
Beschreibung der Abläufe des Prozesses i als NFA P_i:
(mit $j = 3-i$ und $\Delta_i = \{(\texttt{f} \leftarrow \texttt{true})_j, (\texttt{f} \leftarrow \texttt{false})_j, (\texttt{f} == \texttt{true})_j, (\texttt{f} == \texttt{false})_j, \texttt{BkB}_j, \texttt{VkB}_j\}$)



Bemerkungen:

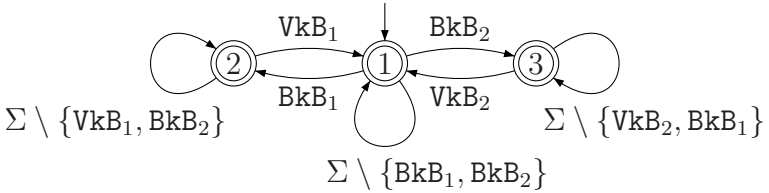
- Zustände 1, 2, 3, 4, 5 entsprechen den entsprechend markierten Programmzeilen
- Bedeutung der Schleifen mit Alphabetsymbolen aus Δ_i: Der Prozeß i interessiert sich nicht für die Aktionen des anderen Prozesses, ändert bei diesen also seinen Zustand nicht. Sie werden also einfach „mitgehört“ und „ignoriert“.

Beschreibung der Booleschen Variable `f` durch einen NFA F:
(mit $\Delta_{\mathtt{f}} = \{\mathtt{BkB}_1, \mathtt{VkB}_1, \mathtt{BkB}_2, \mathtt{VkB}_2\}$):



Die Sprache aller Abläufe des Gesamtsystems ist $L(P_1) \cap L(P_2) \cap L(F)$.

Der NFA *WA*, der diejenigen Abläufe beschreibt, die den wechselseitigen Ausschluß erfüllen (höchstens ein Prozeß ist im kritischen Bereich) sieht folgendermaßen aus:



Damit ist $L(P_1) \cap L(P_2) \cap L(F) \subseteq L(WA)$ zu zeigen.

Es stellt sich heraus, daß $L(P_1) \cap L(P_2) \cap L(F) \subseteq L(WA)$ nicht gilt, d.h. daß unsere Implementierung den gegenseitigen Ausschluß nicht sichert.

Die meisten Werkzeuge, die das Inklusionsproblem automatisch lösen, liefern im Mißerfolgsfall ein Gegenbeispiel. In unserem Fall ist dies z.B. das Wort

$$(\texttt{f} == \texttt{false})_2\ (\texttt{f} == \texttt{false})_1\ (\texttt{f} \leftarrow \texttt{true})_2\ \texttt{BkB}_2\ (\texttt{f} \leftarrow \texttt{true})_1\ \texttt{BkB}_1.$$

Daraus kann ein Grund für die Verletzung des wechselseitigen Ausschlusses abgelesen werden: Die beiden Prozesse können nacheinander die Variable auslesen, anschließend setzen beide die Variable und betreten den kritischen Bereich.

Jedenfalls ist der Algorithmus unbrauchbar!

2.7.2 Zweiter Versuch:

Wir betrachten nun das Verfahren zum wechselseitigen Ausschluß von Leslie Lamport (1978).
Dabei betrachten wir zwei Prozesse P_1 und P_2 mit unterschiedlichem Programmcode und zwei Boolesche Variable `f1` und `f2` (initialisiert mit `false`).

Prozeß P_1

```
   while true do
1:   f1 ← true;                (#)
2:   while (f2 == true)
       do skip enddo;
3:   [Betrete krit. Bereich];
4:   [Verlasse krit. Bereich];
5:   f1 ← false
   enddo;
```

`skip`: Null-Operation
(hat keine Auswirkungen)

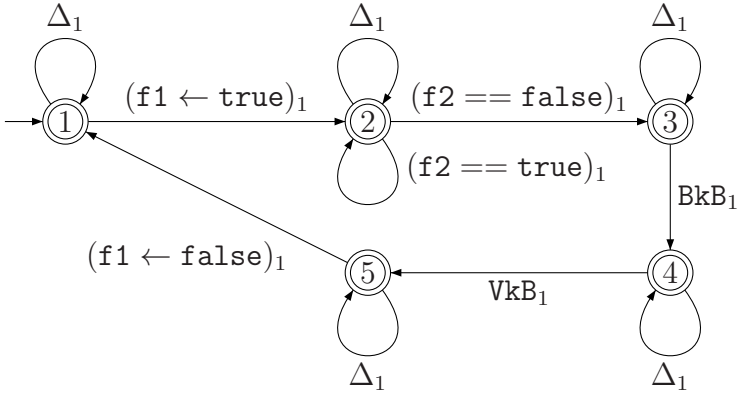
Prozeß P_2

```
   while true do
1:   f2 ← true;                (#)
2:   if (f1 == true) then do
3:     f2 ← false;
4:     while (f1 == true)
         do skip enddo;
       goto 1
     endif;
5:   [Betrete krit. Bereich];
6:   [Verlasse krit. Bereich];
7:   f2 ← false
   enddo;
```

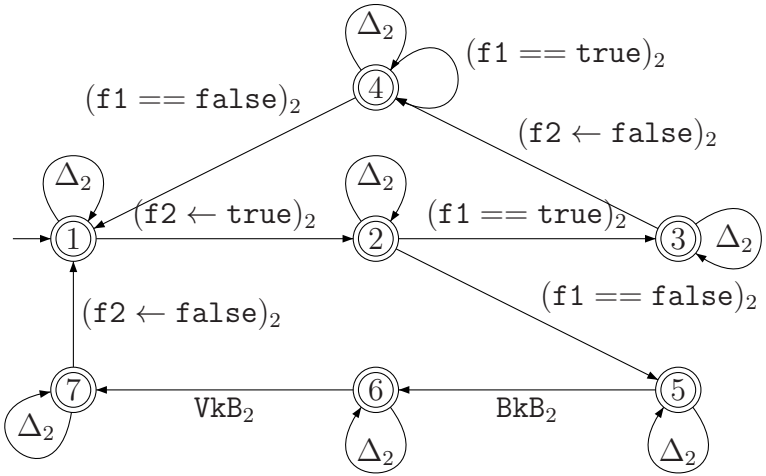
In diesem Fall betrachten wir folgendes Alphabet Σ:

$$\begin{aligned}\Sigma \;=\; & \{(\texttt{f1} \leftarrow \texttt{true})_1, (\texttt{f1} \leftarrow \texttt{false})_1, (\texttt{f1} == \texttt{true})_2, (\texttt{f1} == \texttt{false})_2,\\ & (\texttt{f2} \leftarrow \texttt{true})_2, (\texttt{f2} \leftarrow \texttt{false})_2, (\texttt{f2} == \texttt{true})_1, (\texttt{f2} == \texttt{false})_1,\\ & \texttt{BkB}_1, \texttt{VkB}_1, \texttt{BkB}_2, \texttt{VkB}_2\}.\end{aligned}$$

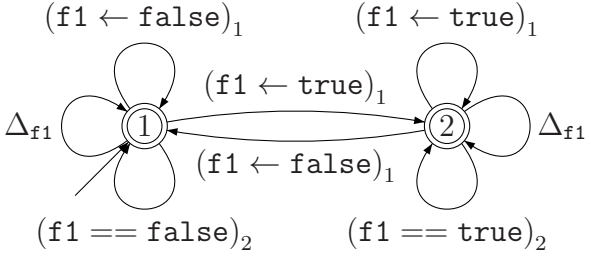
NFA für den Prozeß P_1:
(mit $\Delta_1 = \{(\texttt{f2} \leftarrow \texttt{true})_2, (\texttt{f2} \leftarrow \texttt{false})_2, (\texttt{f1} == \texttt{true})_2, (\texttt{f1} == \texttt{false})_2, \texttt{BkB}_2, \texttt{VkB}_2\}$):



NFA für den Prozeß P_2:
(mit $\Delta_2 = \{(\texttt{f1} \leftarrow \texttt{true})_1, (\texttt{f1} \leftarrow \texttt{false})_1, (\texttt{f2} == \texttt{true})_1, (\texttt{f2} == \texttt{false})_1, \texttt{BkB}_1, \texttt{VkB}_1\}$)



NFA V_1 für die Variable `f1`:
(mit $\Delta_{\texttt{f1}} = \{(\texttt{f2} \leftarrow \texttt{true})_2, (\texttt{f2} \leftarrow \texttt{false})_2, (\texttt{f2} == \texttt{true})_1, (\texttt{f2} == \texttt{false})_1, \texttt{BkB}_1, \texttt{BkB}_2, \texttt{VkB}_1, \texttt{VkB}_2\}$)

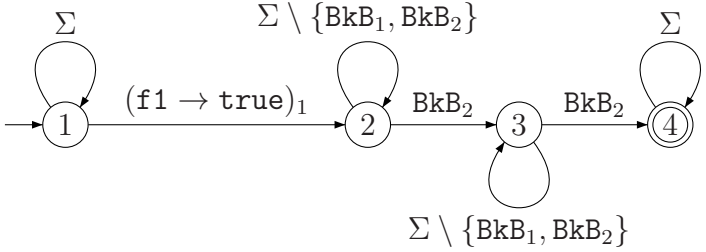


Analog sieht der NFA V_2 für die Variable `f2` aus.
In diesem Fall ist der wechselseitige Ausschluß erfüllt, d.h. es gilt $L(P_1) \cap L(P_2) \cap L(V_1) \cap L(V_2) \subseteq L(WA)$.

Neben dem wechselseitigen Ausschluß soll noch folgende Fairneß-Bedingung für jeden Prozeß i überprüft werden:

(F_i) „Sobald Prozeß i seine Bereitschaft bekundet hat, den kritischen Bereich zu betreten, indem er die Anweisung `(#)` ausführt, kann der andere Prozeß j nicht zweimal hintereinander den kritischen Bereich betreten, ohne daß Prozeß i zwischendurch den kritischen Bereich betritt."

NFA NF_1 für die Abläufe, die (F_1) *nicht* erfüllen:



Man kann zeigen (bzw. einen Computer zeigen lassen), daß $L(P_1) \cap L(P_2) \cap L(V_1) \cap L(V_2) \cap L(NF_1) = \emptyset$. Damit ist Fairneß also für den Prozeß 1 erfüllt.

Hingegen gilt $L(P_1) \cap L(P_2) \cap L(V_1) \cap L(V_2) \cap L(NF_2) \neq \emptyset$, d.h. Fairneß für den Prozeß 2 ist nicht erfüllt. Die automatische (fehlschlagende) Überprüfung liefert auch gleich ein Gegenbeispiel: Das Wort

$$\begin{array}{llll} (\texttt{f2} \leftarrow \texttt{true})_2 & (\texttt{f1} \leftarrow \texttt{true})_1 & (\texttt{f1} == \texttt{true})_2 & (\texttt{f2} \leftarrow \texttt{false})_2 \\ (\texttt{f2} == \texttt{false})_1 & \texttt{BkB}_1 & \texttt{VkB}_1 & (\texttt{f1} \leftarrow \texttt{false})_1 \quad (\texttt{f1} \leftarrow \texttt{true})_1 \\ (\texttt{f2} == \texttt{false})_1 & \texttt{BkB}_1 & & \end{array}$$

ist im Schnitt enthalten.

Die Interpretation dieses Gegenbeispiels ist Ihnen überlassen.

2.7.3 Zusammenfassung:

- Wir haben mit Hilfe von endlichen Automaten zwei Protokolle modelliert, die wechselseitigen Ausschluß realisieren sollen.
- Mit Hilfe der Lösungsverfahren für das Inklusions- bzw. Disjunktheitsproblem haben wir überprüft, ob diese Protokolle tatsächlich wechselseitigen Ausschluß und Fairneß realisieren.

Das bedeutet: die vorgestellten Verfahren können zur Programmverifikation eingesetzt werden.

Bemerkung: Bei realen Programmen hat man allerdings noch damit zu kämpfen, daß die Zustandsmenge des Programms unendlich ist. Damit wird vieles algorithmisch unlösbar („unentscheidbar“) und muß durch approximative Verfahren gelöst werden (vgl. Master-Vorlesung „Verifikation“).

Zusammenfassung reguläre Sprachen

- äquivalente Beschreibungsformen: DFAs, NFAs, reguläre Ausdrücke
- Abschlußeigenschaften: Vereinigung, Produkt, Iteration, Schnitt, Komplement
- Nicht-Regularitätsbeweise: Pumping-Lemma
- minimale DFAs
- Algorithmen für Wort-, Leerheits-, Endlichkeits-, Disjunktheits-, Inklusions- und Äquivalenzproblem
- Anwendung in Verifikation