Automaten und Formale Sprachen
8. Vorlesung

Prof. Dr. Dietrich Kuske

FG Automaten und Logik, TU Ilmenau

Wintersemester 2022/23

Berechnung des minimalen DFA

Zwischenbilanz

Ist $M = (Z, \Sigma, z_0, \delta, E)$ reduzierter DFA, so ist $M_{\equiv}$ der minimale DFA für die reguläre Sprache $L(M)$. Um diesen zu berechnen, benötigen wir einen Algorithmus, der für zwei Zustände $z, z' \in Z$ entscheidet, ob $z \equiv z'$ gilt.

Problem

Es gilt $z \equiv z'$ gdw. $L(M_z) = L(M_{z'})$, d.h. wenn

$$\widehat{\delta}(z, w) \in E \iff \widehat{\delta}(z', w) \in E$$

für alle Wörter w gilt - hiervon gibt es aber unendlich viele.

Um herauszufinden, welche Zustände erkennungsäquivalent sind, markieren wir alle Zustandspaare $\{z, z'\}$, die nicht erkennungsäquivalent sind.

Zunächst gilt $z \not\equiv z'$ für alle $z \in E$ und $z' \notin E$ (Lemma auf Folie 7.13), diese Paare markieren wir zu Beginn.

Angenommen für ein Paar $\{z, z'\}$ existiert ein $a \in \Sigma$, so daß $\delta(z, a) \not\equiv \delta(z', a)$.

Dann gilt auch $z \not\equiv z'$ nach dem Lemma auf Folie 7.13.

Diese Beobachtung erlaubt es uns, weitere Paare als nicht erkennungsäquivalent zu markieren.

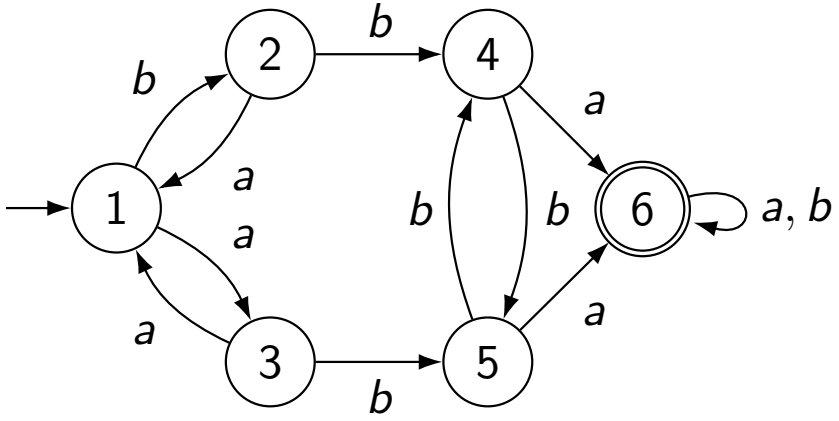
Algorithmus Minimalautomat

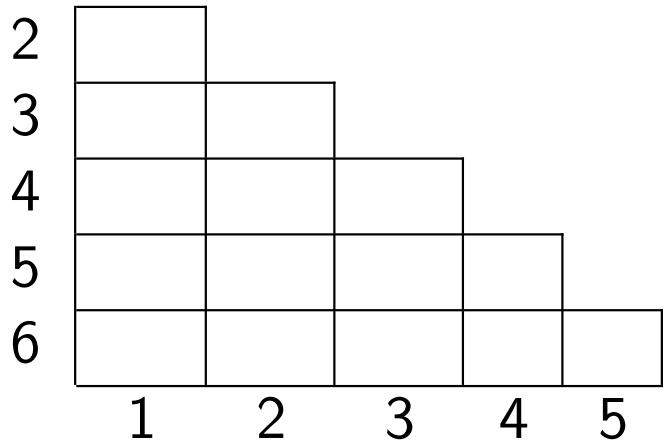
Eingabe: reduzierter DFA M

Ausgabe: Menge der Paare erkennungsäquivalenter Zustände

1. Stelle eine Tabelle aller ungeordneten Zustandspaare $\{z, z'\}$ mit $z \neq z'$ auf.
2. Markiere alle Paare $\{z, z'\}$ mit $z \in E$ und $z' \notin E$.
3. Markiere ein beliebiges unmarkiertes Paar $\{z, z'\}$, für das es ein $a \in \Sigma$ gibt, so daß $\{\delta(z, a), \delta(z', a)\}$ bereits markiert ist (falls dies möglich ist).
4. Wiederhole den vorherigen Schritt, bis sich keine Änderung in der Tabelle mehr ergibt.

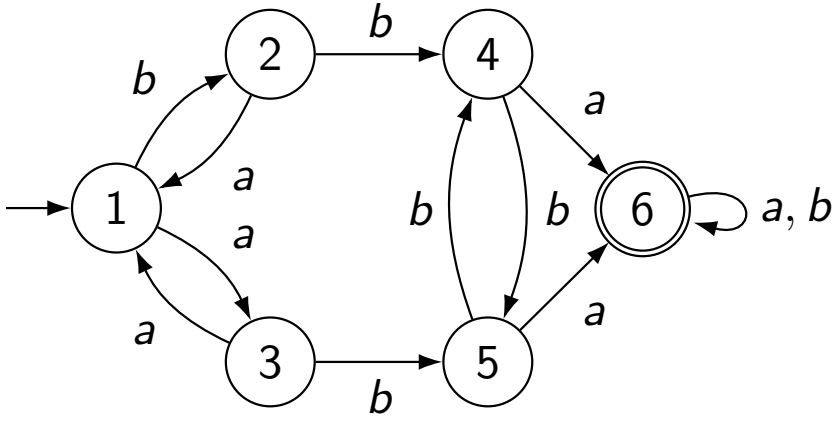
Beispiel für Durchführung des Minimierungsalgorithmus:

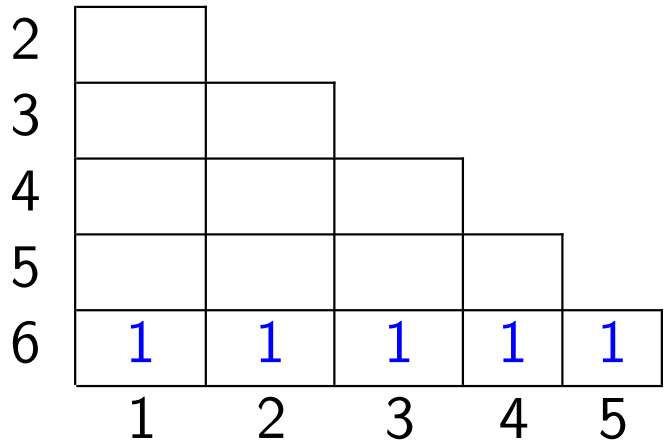




Bereite eine Tabelle aller ungeordneten Zustandspaare vor.

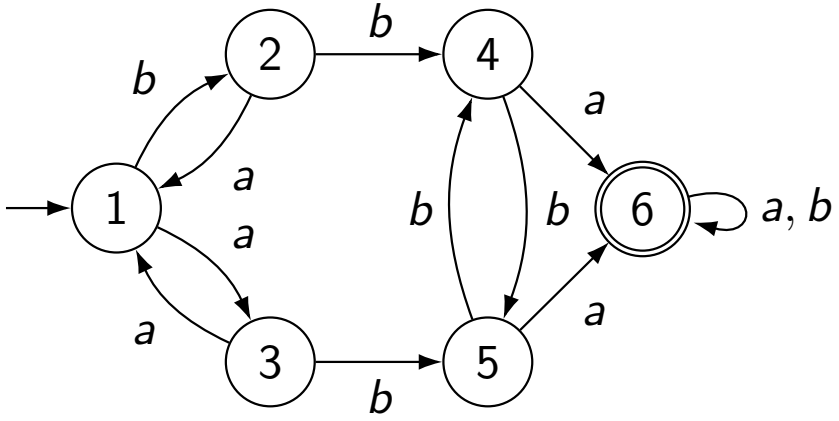
Beispiel für Durchführung des Minimierungsalgorithmus:

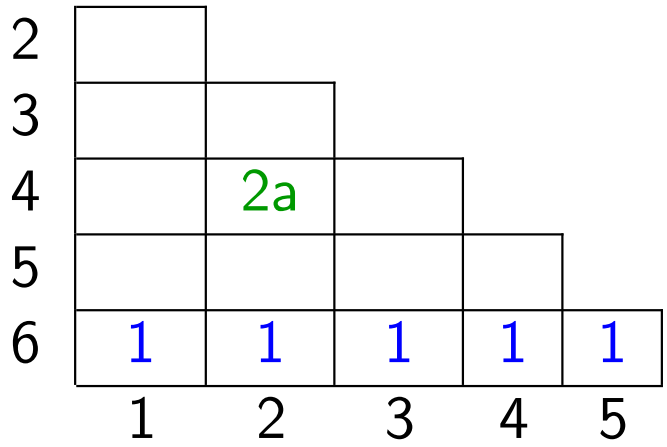




(1) Markiere Paare von Endzuständen und Nicht-Endzuständen.

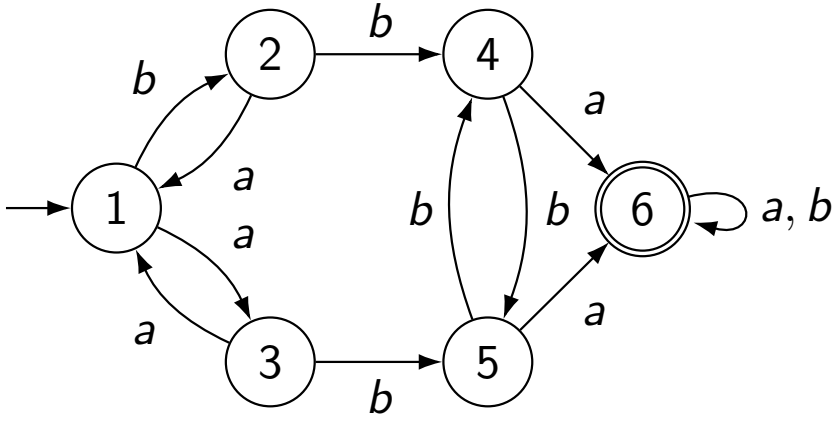
Beispiel für Durchführung des Minimierungsalgorithmus:

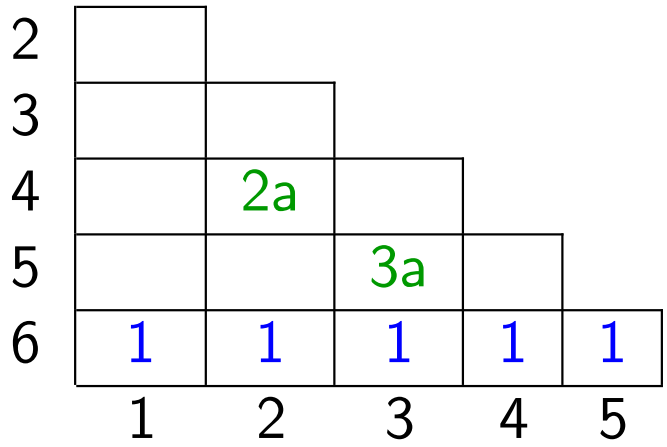




(2) Markiere $\{2,4\}$ da $\{\delta(2,a),\delta(4,a)\} = \{1,6\}$ markiert ist.

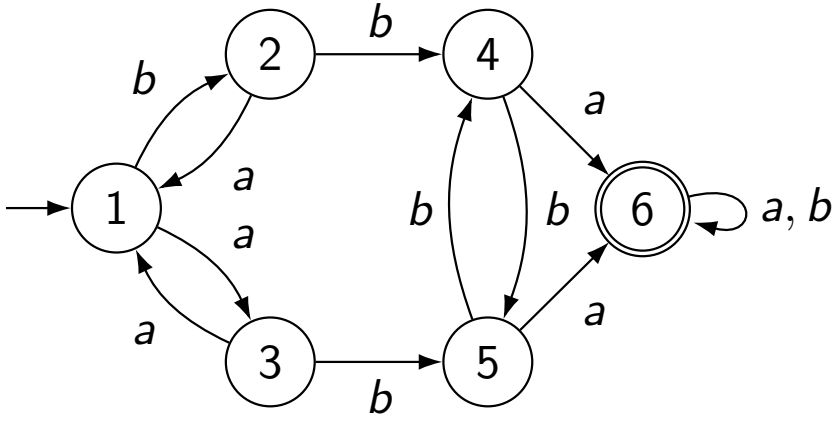
Beispiel für Durchführung des Minimierungsalgorithmus:

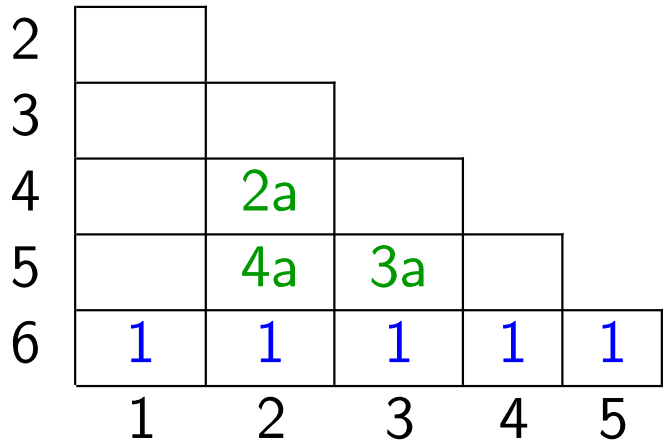




(3) Markiere $\{3,5\}$ da $\{\delta(3,a),\delta(5,a)\} = \{1,6\}$ markiert ist.

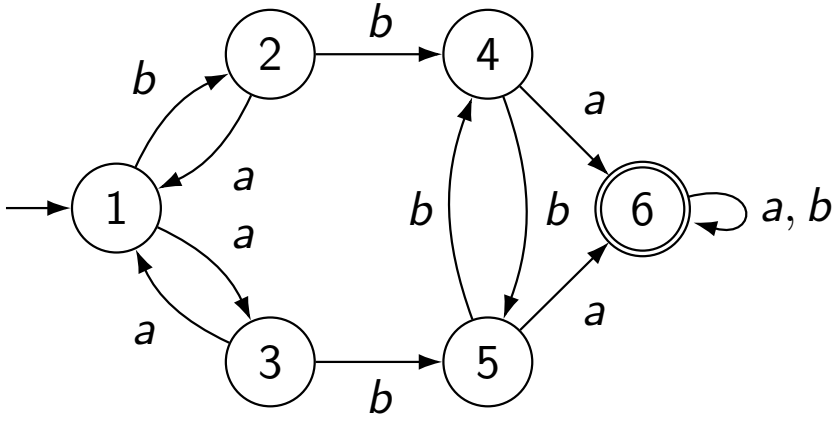
Beispiel für Durchführung des Minimierungsalgorithmus:





(4) Markiere $\{2,5\}$ da $\{\delta(2,a),\delta(5,a)\} = \{1,6\}$ markiert ist.

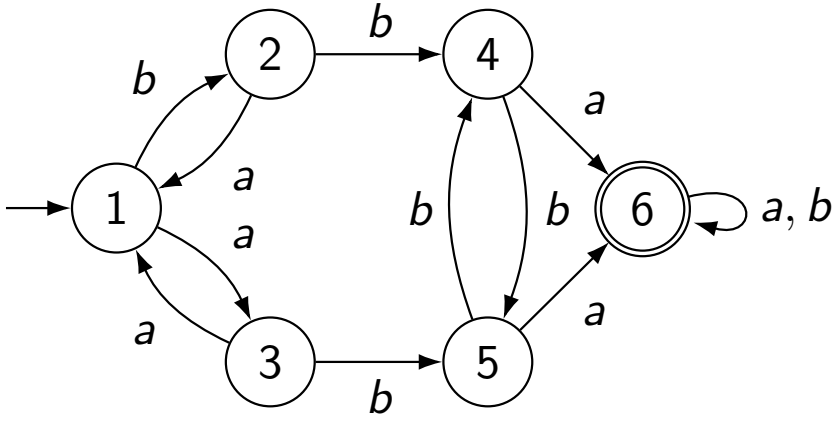
Beispiel für Durchführung des Minimierungsalgorithmus:



2					
3					
4		2a	5a		
5		4a	3a		
6	1	1	1	1	1
	1	2	3	4	5

(5) Markiere $\{3,4\}$ da $\{\delta(3,a),\delta(4,a)\} = \{1,6\}$ markiert ist.

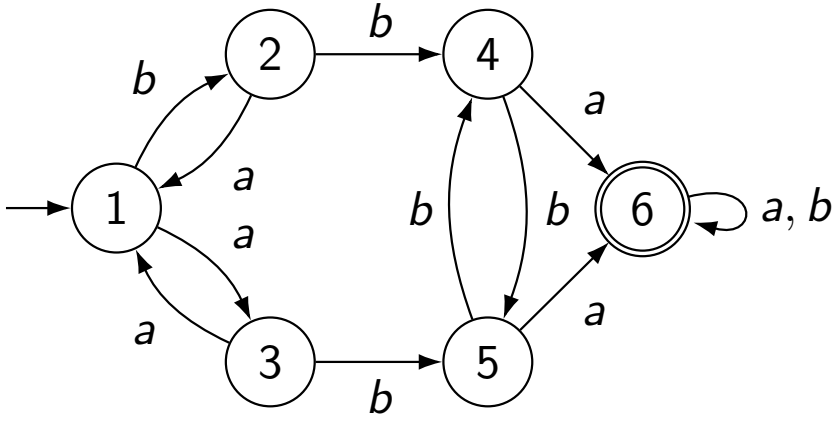
Beispiel für Durchführung des Minimierungsalgorithmus:



2					
3					
4		2a	5a		
5	6a	4a	3a		
6	1	1	1	1	1
	1	2	3	4	5

(6) Markiere $\{1,5\}$ da $\{\delta(1,a),\delta(5,a)\} = \{3,6\}$ markiert ist.

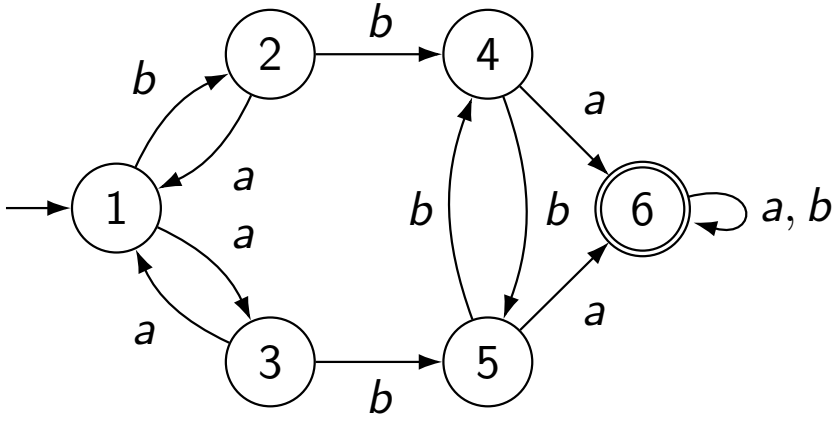
Beispiel für Durchführung des Minimierungsalgorithmus:



2					
3					
4	7a	2a	5a		
5	6a	4a	3a		
6	1	1	1	1	1
	1	2	3	4	5

(7) Markiere $\{1,4\}$ da $\{\delta(1,a),\delta(4,a)\} = \{3,6\}$ markiert ist.

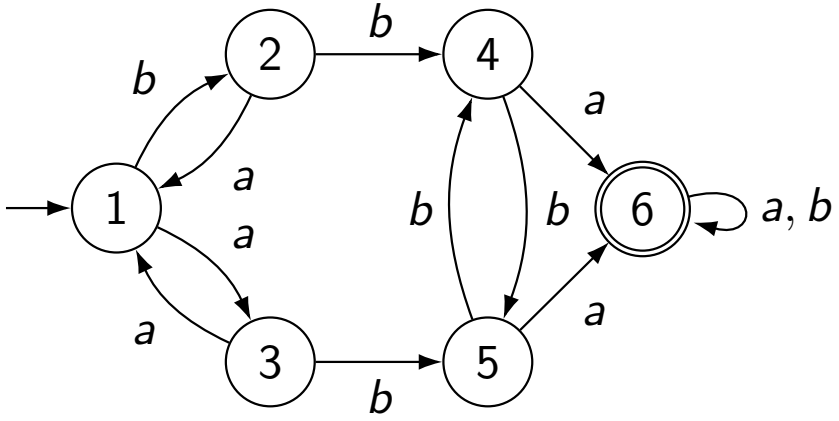
Beispiel für Durchführung des Minimierungsalgorithmus:



2					
3	8b				
4	7a	2a	5a		
5	6a	4a	3a		
6	1	1	1	1	1
	1	2	3	4	5

(8) Markiere $\{1,3\}$ da $\{\delta(1,b),\delta(3,b)\} = \{2,5\}$ markiert ist.

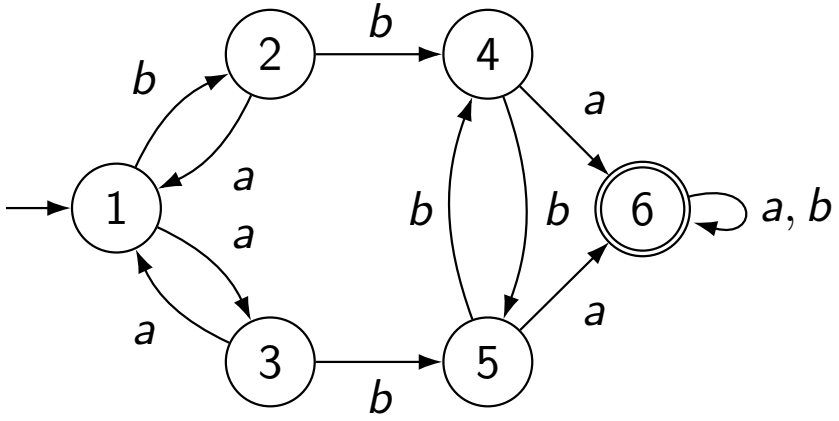
Beispiel für Durchführung des Minimierungsalgorithmus:



2	9b				
3	8b				
4	7a	2a	5a		
5	6a	4a	3a		
6	1	1	1	1	1
	1	2	3	4	5

(9) Markiere $\{1,2\}$ da $\{\delta(1,b),\delta(2,b)\} = \{2,4\}$ markiert ist.

Beispiel für Durchführung des Minimierungsalgorithmus:



2	9b				
3	8b				
4	7a	2a	5a		
5	6a	4a	3a		
6	1	1	1	1	1
	1	2	3	4	5

Die verbleibenden Zustandspaare $\{2,3\}$ und $\{4,5\}$ können nicht mehr markiert werden. $\rightsquigarrow$ Sie sind erkennungsäquivalent.

Satz

Für einen reduzierten DFA $M = (Z, \Sigma, z_0, \delta, E)$ wird ein Paar $\{z, z'\} \subseteq Z$ mit $z \neq z'$ genau dann durch den Markierungsalgo. markiert, wenn $z \not\equiv z'$.

Beweis:

„$\Longrightarrow$“ z.z.: Falls $\{z, z'\}$ markiert wird, so gilt $z \not\equiv z'$.

Beweis durch Induktion über den Zeitpunkt, zu dem $\{z, z'\}$ markiert wird.

IA: $\{z, z'\}$ wird zu Beginn markiert, weil $z \in E$ und $z' \notin E$.

$\Longrightarrow$ $z \not\equiv z'$ (nach Lemma auf Folie 7.13).

IS: $\{z, z'\}$ wird irgendwann markiert, weil ein $a \in \Sigma$ existiert, so daß $\{\delta(z, a), \delta(z', a)\}$ zu einem **früheren** Zeitpunkt markiert wurde.

$\overset{\text{IV}}{\Longrightarrow}$ $\delta(z, a) \not\equiv \delta(z', a)$

$\Longrightarrow$ $z \not\equiv z'$ (nach Lemma auf Folie 7.13).

Damit ist die Implikation „$\Longrightarrow$“ gezeigt.

„$\Longleftarrow$“ z.z.: Wenn $z \not\equiv z'$ gilt, dann wird $\{z, z'\}$ irgendwann markiert.

Gelte $z \not\equiv z'$. Dann gibt es also ein Wort, das von M_z akzeptiert und von $M_{z'}$ nicht akzeptiert wird (oder umgekehrt).

Sei $\lambda(z, z')$ die **minimale** Länge eines Wortes w mit $\widehat{\delta}(z, w) \in E$, $\widehat{\delta}(z', w) \notin E$ (oder umgekehrt).

Wir zeigen durch Induktion über $\lambda(z, z')$, daß $\{z, z'\}$ markiert wird.

IA: $\lambda(z, z') = 0$

$\Longrightarrow$ $z \in E$ und $z' \notin E$ (oder umgekehrt)

$\Longrightarrow$ $\{z, z'\}$ wird zu Beginn markiert.

IS: Sei $\lambda(z, z') > 0$.

$\Longrightarrow$ Es gibt ein Wort au ($a \in \Sigma$ und $u \in \Sigma^*$) mit $|au| = \lambda(z, z')$, so daß

- $E \ni \widehat{\delta}(z, au) = \widehat{\delta}\big(\delta(z, a), u\big)$ und
- $E \not\ni \widehat{\delta}(z', au) = \widehat{\delta}\big(\delta(z', a), u\big)$ (oder umgekehrt).

$\Longrightarrow \delta(z, a) \not\equiv \delta(z', a)$ und $\lambda\big(\delta(z, a), \delta(z', a)\big) \leq |u| < \lambda(z, z')$

$\overset{\text{IV}}{\Longrightarrow}$ $\big\{\delta(z, a), \delta(z', a)\big\}$ wird irgendwann markiert.

$\Longrightarrow$ $\{z, z'\}$ wird markiert.

Damit ist auch die Implikation „$\Longleftarrow$“ gezeigt. □

Hinweise für die Durchführung des Minimierungsalgorithmus:

- Die Tabelle möglichst so aufstellen, daß jedes Paar nur genau einmal vorkommt! Also bei Zustandsmenge $\{1, \ldots, n\}$ Zeilen $2, \ldots, n$ und Spalten $1, \ldots, n-1$
- Bitte angeben, welche Zustände in welcher Reihenfolge und warum markiert wurden!

Entscheidbarkeit

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

Probleme

- 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?
- Schnittproblem: 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?

Das Wortproblem

Eingabe: DFA $M = (Z, \Sigma, z_0, \delta, E)$ 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:

$z := z_0$
for $i := 1$ **to** n **do**
  $z := \delta(z, a_i)$
endfor
if $z \in E$ **then return**(JA) **else return**(NEIN)

Das Leerheitsproblem:

Eingabe: NFA $M = (Z, \Sigma, S, \delta, E)$.

Frage: $L(M) = \emptyset$?

Verfahren:

Sei $G = (Z, \rightarrow)$ der gerichtete Graph mit

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

Dann gilt: $L(M) \neq \emptyset$ genau dann, wenn es in dem Graphen G einen (evtl. leeren) Pfad von einem Knoten aus S zu einem Knoten aus E gibt.

Dies kann z.B. mit dem Algorithmus von Dijkstra entschieden werden.

Das Endlichkeitsproblem:

Eingabe: NFA $M = (Z, \Sigma, S, \delta, E)$.

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 $z \in Z$, $z_0 \in S$ und $z_1 \in E$ 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.

Das Schnittproblem:

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 $L(M) = L(M_1) \cap L(M_2)$, siehe Folie 4.10.

Teste, ob $L(M) = \emptyset$ gilt.

Das Inklusionsproblem:

Eingabe: NFAs M_1 und M_2.

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

Verfahren: Aus M_1 und M_2 können wir einen NFA M mit $L(M) = \overline{L(M_2)} \cap L(M_1)$ konstruieren (siehe Folien 4.6 und 4.10).

Es gilt $L(M_1) \subseteq L(M_2)$ genau dann, wenn $L(M) = \emptyset$.

Das Äquivalenzproblem:

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:

Bestimme zu M_i ($i \in \{1, 2\}$) den äquivalenten minimalen DFA N_i.

Dann gilt $L(M_1) = L(M_2)$ genau dann, wenn N_1 und N_2 isomorph sind (d.h. sie können durch Umbenennung der Zustände ineinander überführt werden).

Effizienzbetrachtungen:

Je nachdem, in welcher Darstellung eine reguläre Sprache L gegeben ist, kann die Komplexität der oben beschriebenen Verfahren sehr unterschiedlich ausfallen.

Beispiel: Äquivalenzproblem $L_1 = L_2$:

- L_1 und L_2 gegeben als DFAs

  $\rightsquigarrow$ Zeitaufwand $O(n^2)$

- L_1 und L_2 gegeben als Grammatiken, reguläre Ausdrücke oder NFAs

  $\rightsquigarrow$ exponentieller Zeitaufwand,
  genauer: „PSPACE-vollständig“

  Das bedeutet unter anderem: Es ist kein polynomieller Algorithmus bekannt, und es ist „sehr unwahrscheinlich“, daß einer existiert.

  Mehr zur Komplexitätsklasse PSPACE und verwandten Fragestellungen im abschließenden Kapitel der Vorlesung „Berechenbarkeit und Komplexität“.

Übersicht

Bei Eingabe der regulären Sprache als NFA bzw. DFA ergeben sich die folgenden Zeitschranken:

-problem	NFA	DFA
Wort$\sim$	polynomiell	linear
Leerheits$\sim$	polynomiell	polynomiell
Endlichkeits$\sim$	polynomiell	polynomiell
Schnitt$\sim$	polynomiell	polynomiell
Inklusions$\sim$	exponentiell	polynomiell
Äquivalenz$\sim$	exponentiell	polynomiell

Es spricht viel dafür, daß die exponentiellen Zeitschranken nicht durch polynomielle ersetzt werden können.

Anwendung: Verifikation

Ein abschließendes Anwendungsbeispiel:

- 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 Schnittproblem „$L_{Sys} \cap L_{Spec} = \emptyset$?“ zu lösen.

Wir haben gesehen: Sind beide Sprachen regulär, so gibt es für diese Fragen algorithmische Verfahren!

Versuch 1: Die Prozesse P_1, P_2 verwenden eine gemeinsame Boolesche Variable `f`, die mit `false` initialisiert wird.

Programmcode für P_1, P_2

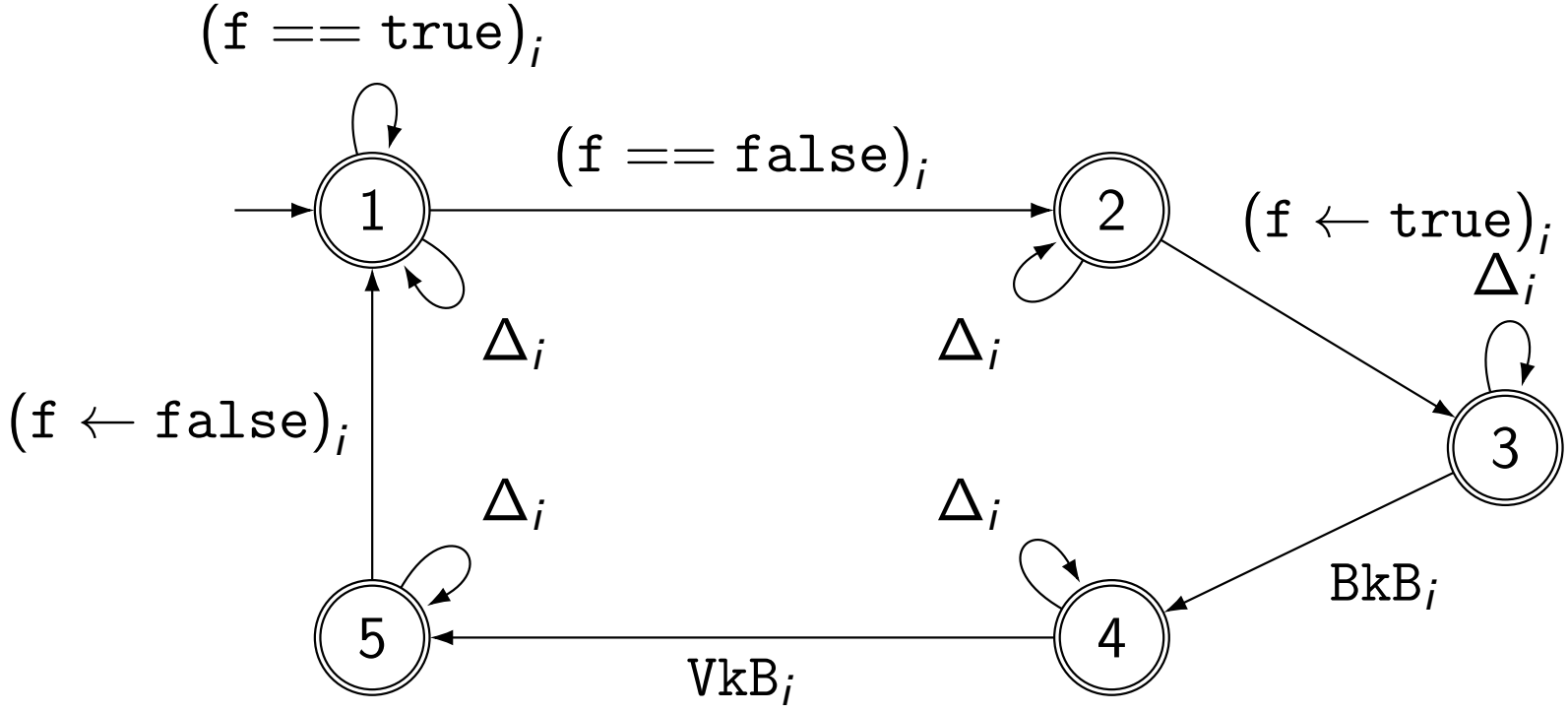
```
   while true do
1: if (f == false) then do
     begin
2:     f ← true
3:     [Betrete kritischen Bereich]
4:     [Verlasse kritischen Bereich]
5:     f ← false
     endif
   enddo
```

Wir verwenden folgendes Alphabet, bestehend aus den Programm-Befehlen und den Abfragen der Booleschen Variablen:

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

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 endlicher Automat P_i:

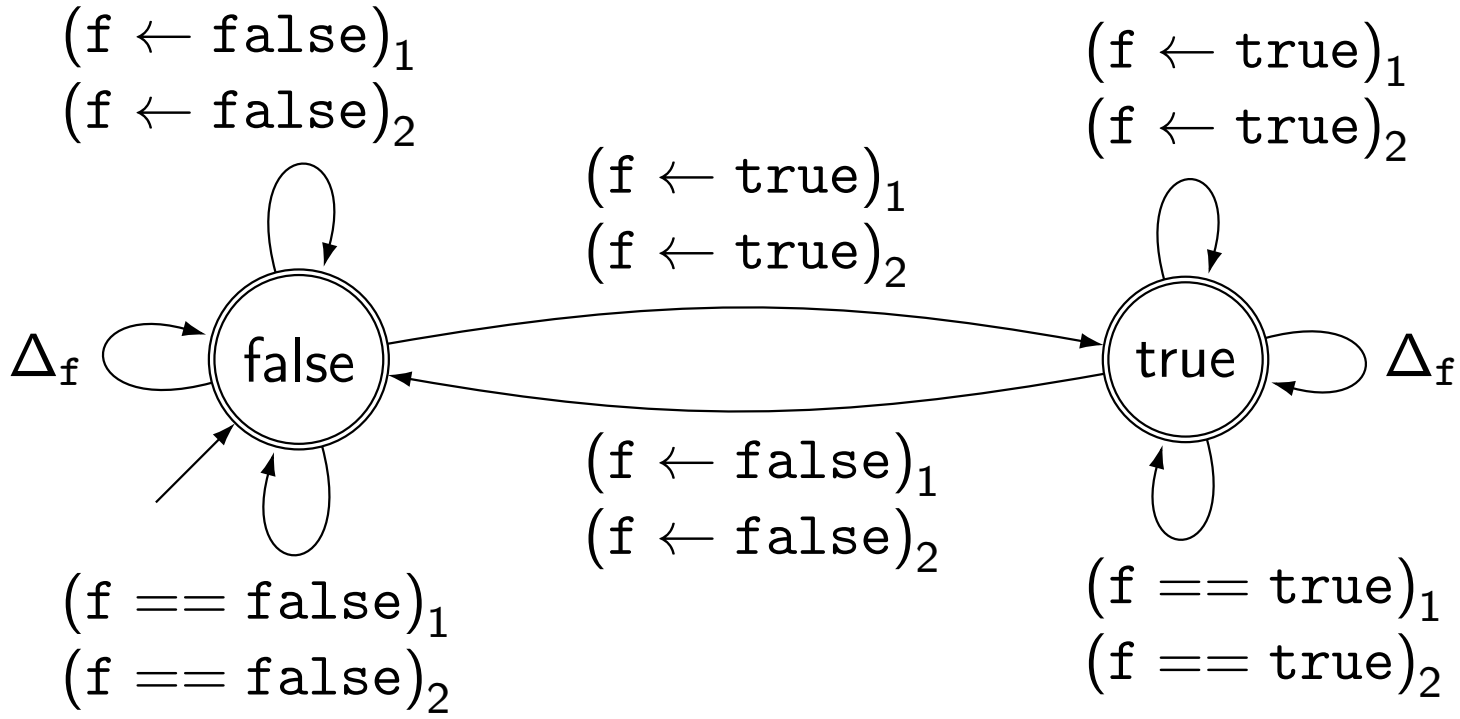


mit $\Delta_i =$
$\{(\texttt{f} \leftarrow \texttt{true})_j, (\texttt{f} \leftarrow \texttt{false})_j, (\texttt{f} == \texttt{true})_j, (\texttt{f} == \texttt{false})_j, \texttt{BkB}_j, \texttt{VkB}_j\}$
wobei $j = 3 - i$.

Bemerkungen:

- Bedeutung der Zustände 1, 2, 3, 4, 5: diese 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“.

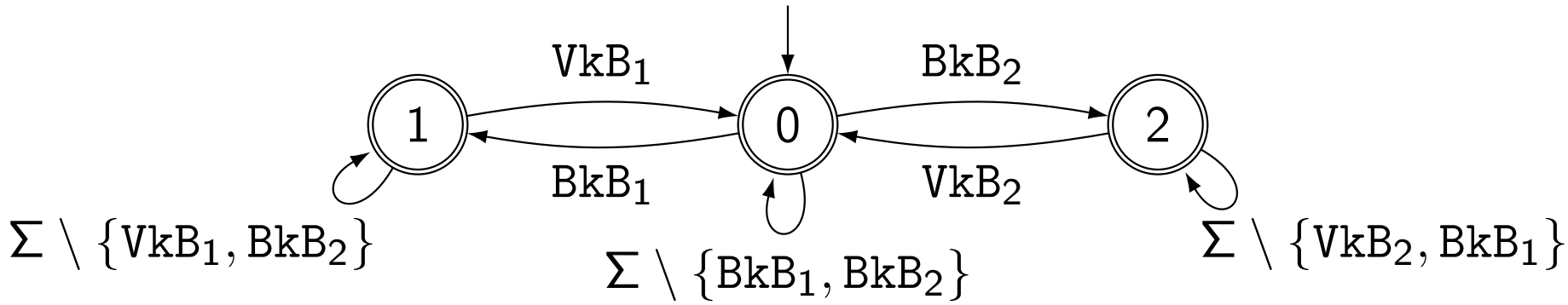
Beschreibung der Booleschen Variable f durch einen Automaten F:



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 Automat 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 das Verfahren unbrauchbar!

Zusatzmaterial auf Folien 8.30 ff.: Verfahren von 1978 von Leslie Lamport, Beweis der Korrektheit, der Fairneß (für einen Prozeß) und Nicht-Fairneß (für anderen Prozeß)

Zusammenfassung 8. Vorlesung

- Algorithmus für die Minimierung eines DFA
- Algorithmen für Leerheits- u.a. Probleme für reguläre Sprachen
- Modellierung eines Protokolls und Identifikation eines Fehlers mit Hilfe des Lösungsverfahrens für das Inklusionsproblem.
- Zusatzmaterial: Verifikation eines alternativen Protokolls unter Verfendung des Lösungsverfahren für Inklusions- und Schnittproblem

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 unentscheidbar (vgl. Vorlesung „Berechenbarkeit und Komplexität“) und muß durch approximative Verfahren gelöst werden (vgl. Master-Vorlesung „Verifikation“).

Zusammenfassung rechtslineare Sprachen

- äquivalente Beschreibungsformen: Typ-3-Grammatiken, DFAs, NFAs, reguläre Ausdrücke
- Abschlußeigenschaften: Vereinigung, Produkt, Iteration, Schnitt, Komplement
- Nicht-Regularitätsbeweise: Pumping-Lemma und Myhill-Nerode
- minimale DFAs
- Algorithmen für Wort-, Leerheits-, Endlichkeits-, Schnitt-, Inklusions- und Äquivalenzproblem
- Anwendung in Verifikation

Zusatzmaterial

Versuch 2: 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)

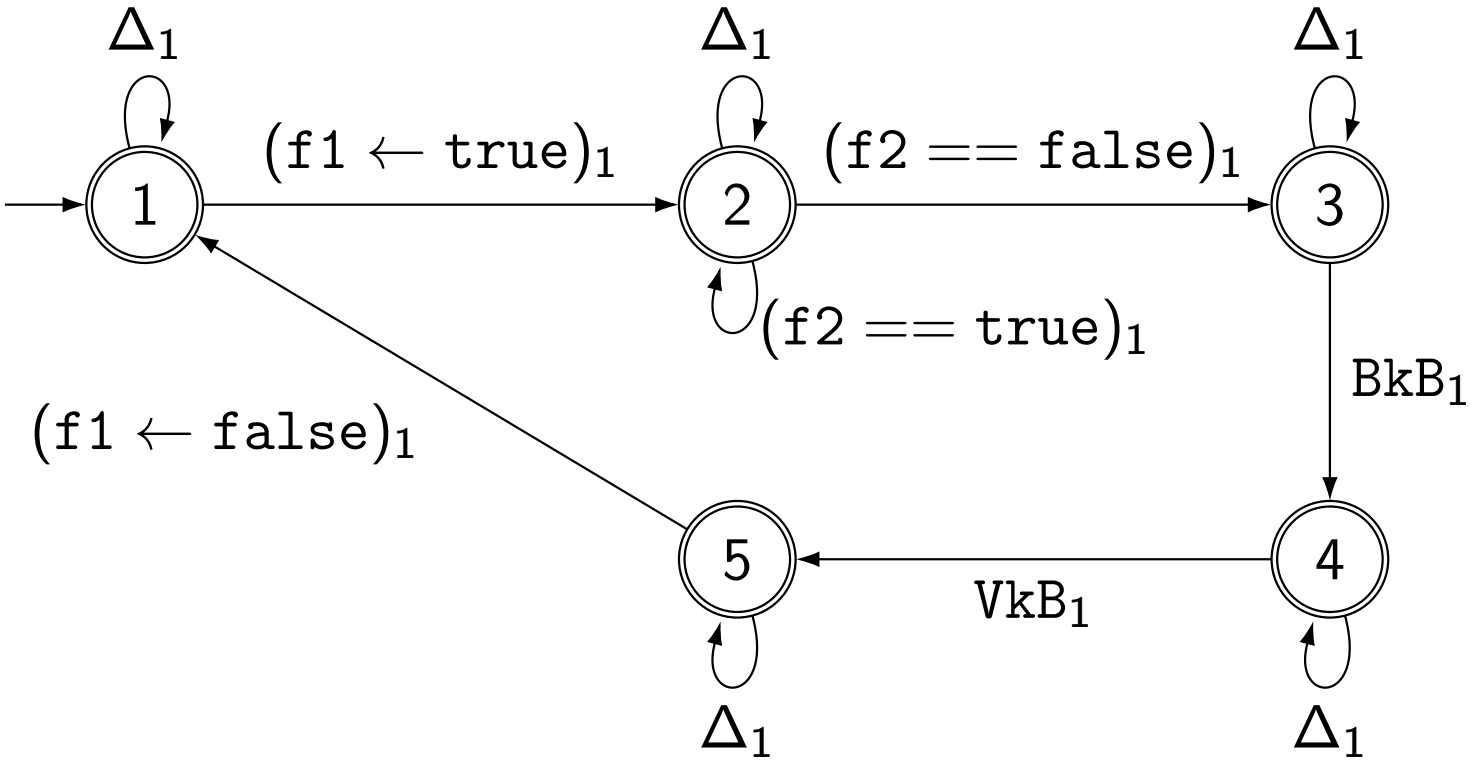
Prozeß P_2

```
   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:

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

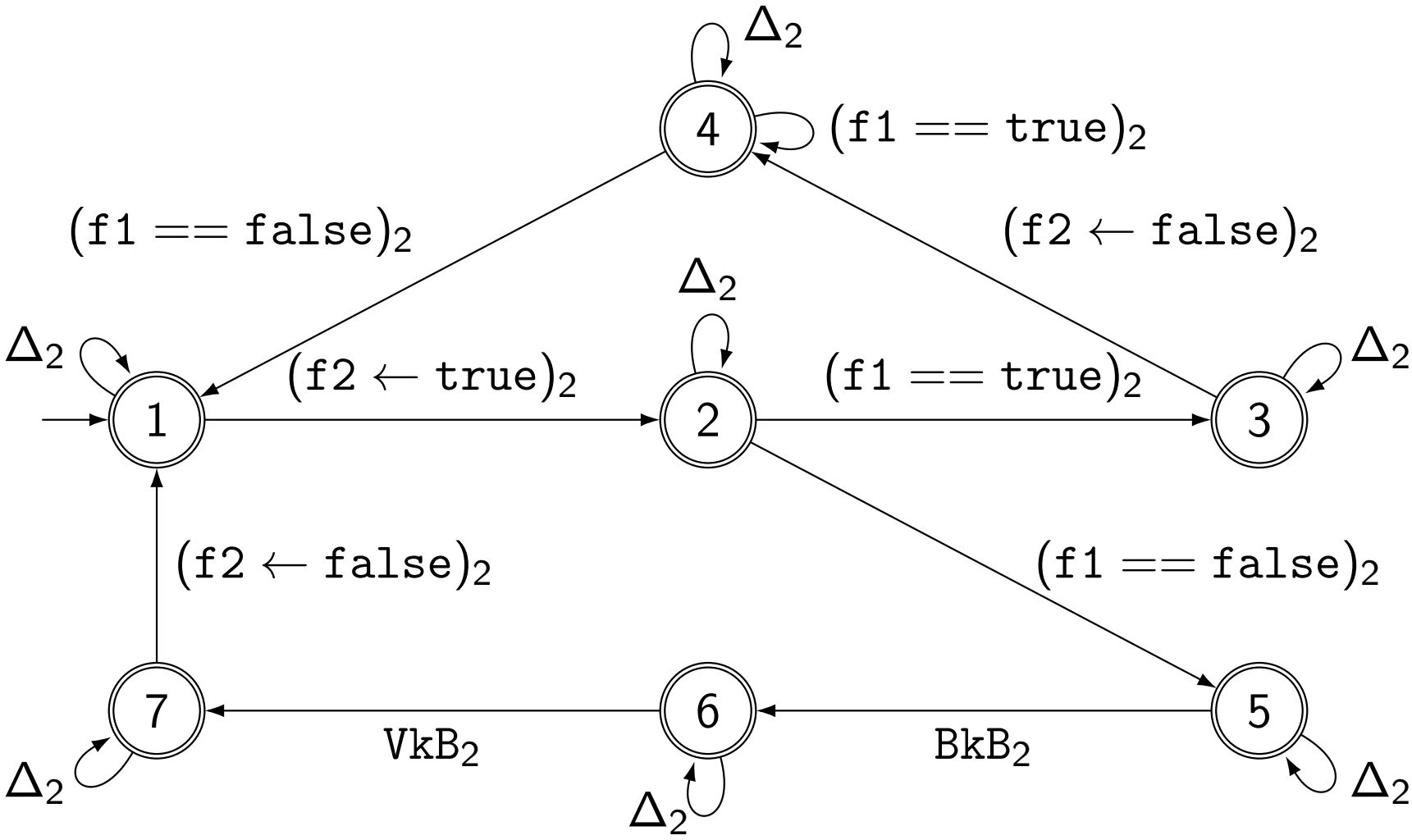
Automat für den Prozeß P_1:



Dabei gilt für die „mitgehörten“ Alphabetsymbole:

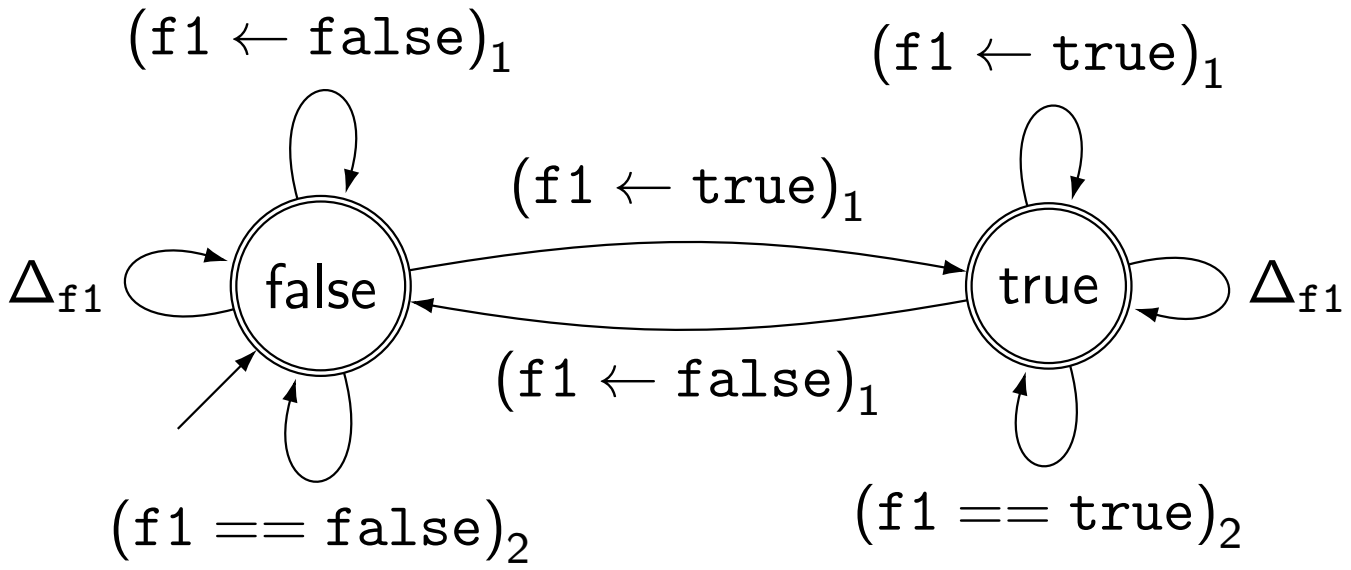
$$\begin{aligned}\Delta_1 \quad = \quad & \{(\texttt{f2} \leftarrow \texttt{true})_2, (\texttt{f2} \leftarrow \texttt{false})_2, (\texttt{f1} == \texttt{true})_2, \\ & (\texttt{f1} == \texttt{false})_2, \texttt{BkB}_2, \texttt{VkB}_2\}\end{aligned}$$

Automat für den Prozeß P_2:



$$\Delta_2 = \{(\texttt{f1} \leftarrow \texttt{true})_1, (\texttt{f1} \leftarrow \texttt{false})_1, (\texttt{f2} == \texttt{true})_1, (\texttt{f2} == \texttt{false})_1, \texttt{BkB}_1, \texttt{VkB}_1\}$$

Automat V_1 für die Variable f1:



$$\Delta_{\mathtt{f1}} = \{(\mathtt{f2} \leftarrow \mathtt{true})_2, (\mathtt{f2} \leftarrow \mathtt{false})_2, (\mathtt{f2} == \mathtt{true})_1, (\mathtt{f2} == \mathtt{false})_1, \mathtt{BkB}_1, \mathtt{BkB}_2, \mathtt{VkB}_1, \mathtt{VkB}_2\}$$

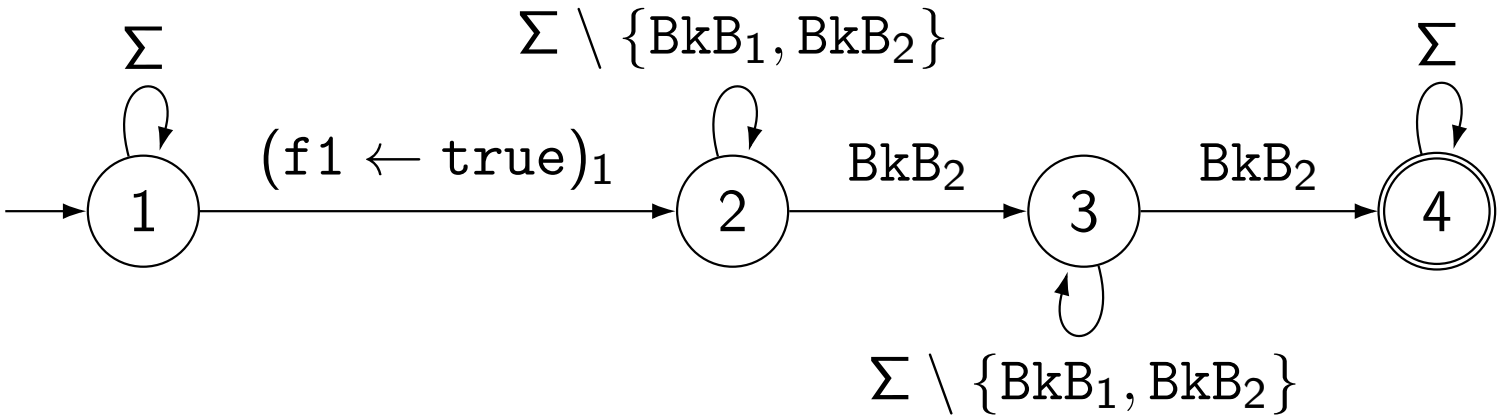
Analog sieht der Automat V_2 für die Variable f2 aus.

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."

Automat NF_1, der genau die Abläufe erkennt, 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

$$(\texttt{f2} \leftarrow \texttt{true})_2 \quad (\texttt{f1} \leftarrow \texttt{true})_1 \quad (\texttt{f1} == \texttt{true})_2 \quad (\texttt{f2} \leftarrow \texttt{false})_2$$
$$(\texttt{f2} == \texttt{false})_1 \quad \texttt{BkB}_1 \quad \texttt{VkB}_1 \quad (\texttt{f1} \leftarrow \texttt{false})_1 \quad (\texttt{f1} \leftarrow \texttt{true})_1$$
$$(\texttt{f2} == \texttt{false})_1 \quad \texttt{BkB}_1$$

ist im Schnitt enthalten.

Die Interpretation dieses Gegenbeispiels ist Ihnen überlassen.