Endliche Automaten und reguläre Sprachen

Endliche Automaten

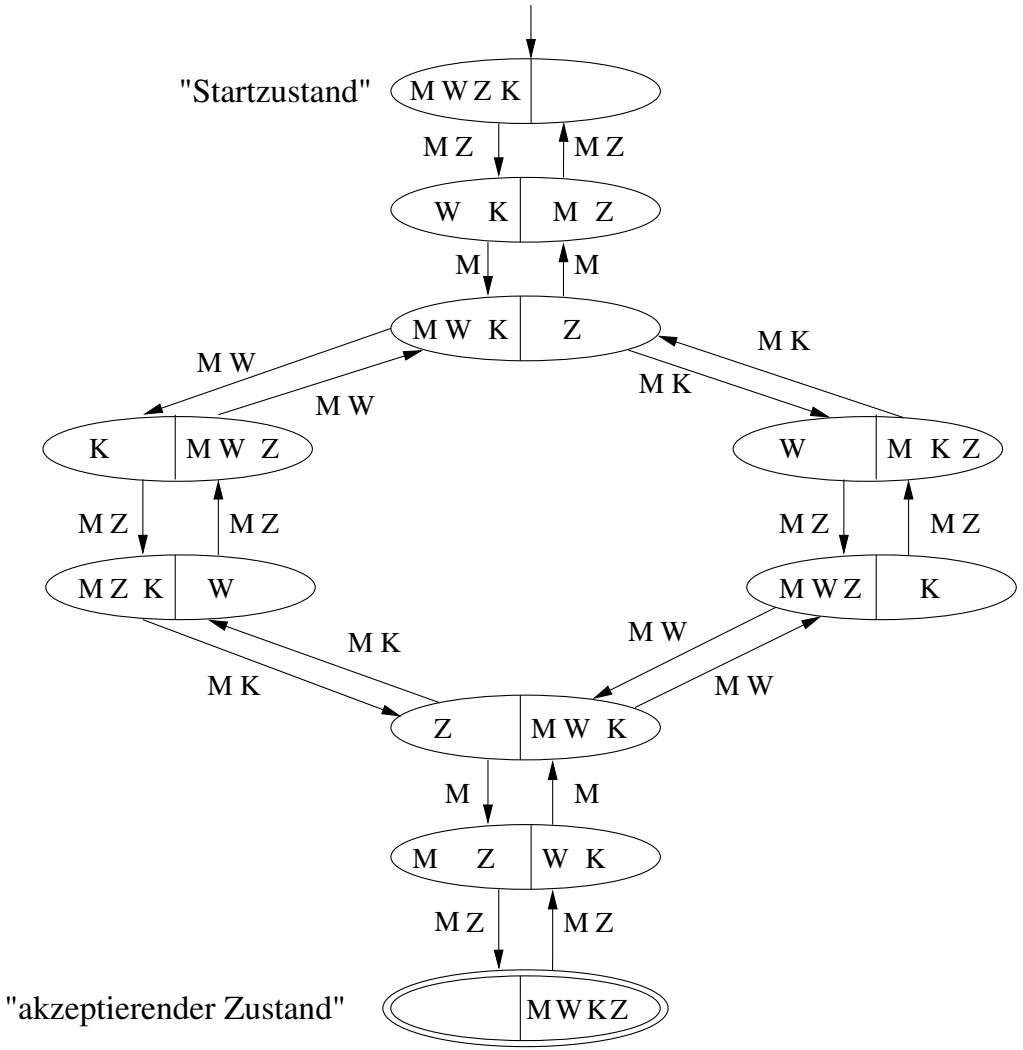
Endliche Automaten erlauben eine **Beschreibung von Handlungsabläufen**:

> Wie ändert sich ein Systemzustand in Abhängigkeit von veränderten Umgebungsbedingungen?

Vielfältiges Einsatzgebiet, nämlich:

- in der Definition der **regulären Sprachen**
  > also der Menge aller Folgen von Ereignissen,
  > die von einem Startzustand in einen gewünschten Zustand führen,
- in der **Entwicklung digitaler Schaltungen**
- in der **Softwaretechnik** (z. B. in der Modellierung des Applikationsverhaltens)
- in der **Compilierung**: Lexikalische Analyse
- im **Algorithmenentwurf** für String Probleme
- in der **Abstraktion tatsächlicher Automaten** (wie Bank- und Getränkeautomaten, Fahrstühle etc.)

Das Problem der Flußüberquerung: Transitionssysteme



Dieser endliche Automat „akzeptiert“ genau diejenigen Folgen von einzelnen Flussüberquerungen, die vom Startzustand in den akzeptierenden Zustand führen.

Lexikalische Analyse

Zu Beginn liest der Compiler Anweisung nach Anweisung und bricht Anweisungen in Tokenklassen auf.

Betrachte zum Beispiel die Anweisung

```
if distance >= rate * (end - start)
then distance = maxdistance;
```

mit den Tokenklassen:

- Keywords (für `if` und `then`),
- Variablen (für `distance`, `rate`, `end`, `start`, `maxdistance`),
- Operatoren (für `*`, `-`, `=`) und Vergleichsoperatoren (für `>=`),
- Klammern (für `(` und `)`) und Semikolon (für `;`).

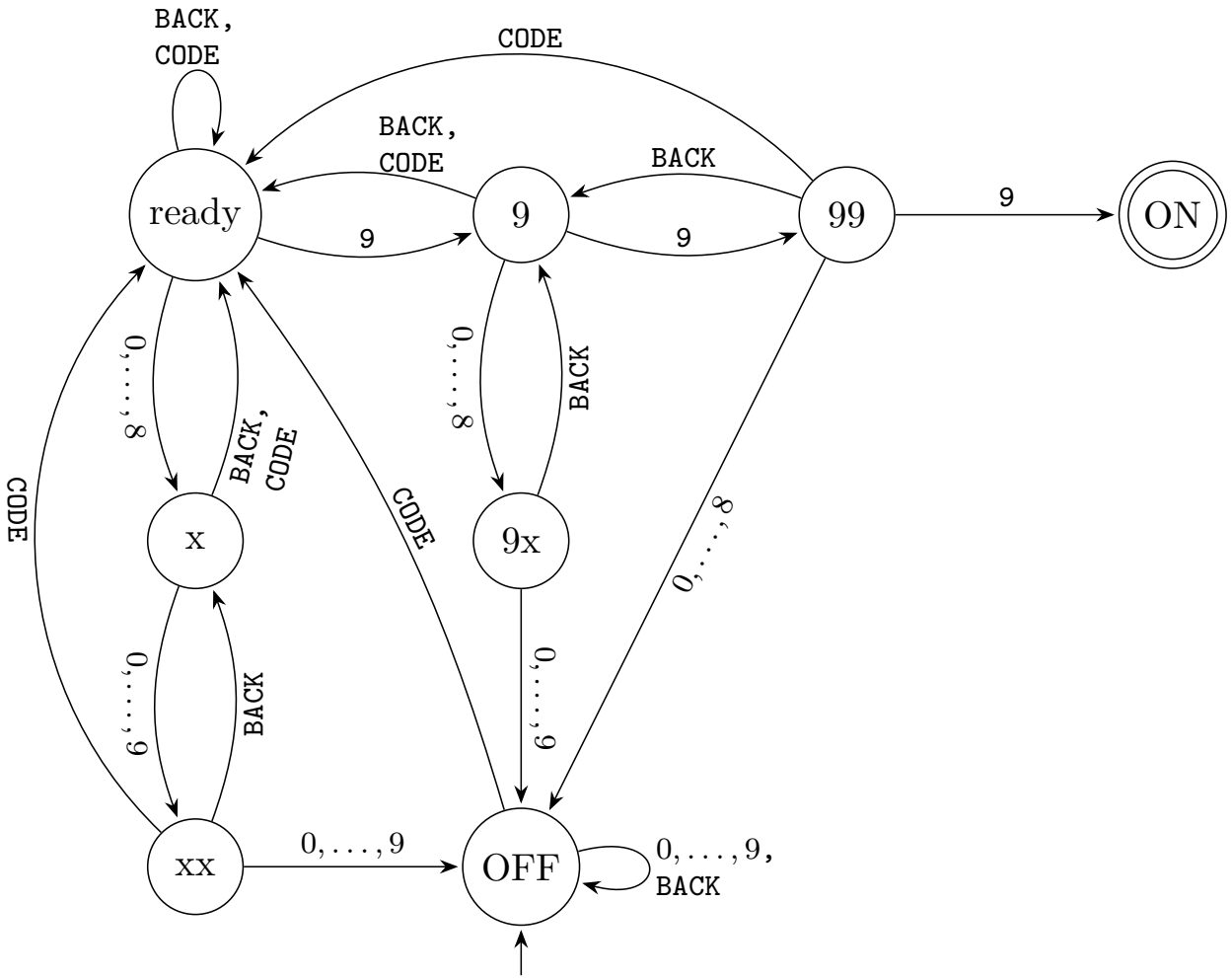
Die lexikalische Analyse benutzt reguläre Grammatiken, bzw nichtdeterminische endliche Automaten.

Freischaltung eines Fernsehers (1/2)

- Um die Kindersicherung des Fernsehers über die Fernbedienung freizuschalten, muss ein dreistelliger Code korrekt eingegeben werden. Dabei sind die folgenden Tasten relevant:
  - Die Tasten $0, \ldots, 9$,
  - die Taste `CODE` sowie
  - die Taste `BACK`.
- Die Taste `CODE` muss vor Eingabe des Codes gedrückt werden.
- Wird `CODE` während der Codeeingabe nochmals gedrückt, so wird die Eingabe neu begonnen.
- Wird `BACK` gedrückt, so wird die zuletzt eingegebene Zahl zurückgenommen.

Der Code zum Entsperren ist 999.

Freischaltung eines Fernsehers (2/2)



Der Automat „akzeptiert“ alle Folgen von Bedienoperationen, die vom Zustand „**ready**“ in den Zustand „**ON**“ führen.

Was ist ein endlicher Automat?

Ein **deterministischer endlicher Automat** (engl. deterministic finite automaton, kurz **DFA**)

$$A = (\Sigma, Q, \delta, q_0, F)$$

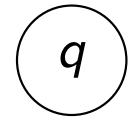
besteht aus:

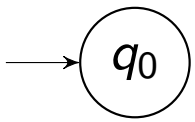
- einer endlichen Menge Σ, dem **Eingabealphabet**,
- einer endlichen Menge Q, der **Zustandsmenge** (die Elemente aus Q werden Zustände genannt),
- einer Funktion δ von $Q \times \Sigma$ nach Q, der **Übergangsfunktion** (oder Überführungsfunktion),
- einem Zustand $q_0 \in Q$, dem **Startzustand**,
- sowie einer Menge $F \subseteq Q$ von **Endzuständen** bzw. akzeptierenden Zuständen. (Der Buchstabe F steht für „final states“, also „Endzustände“).

Das Zustandsdiagramm, bzw. der Automatengraph, die grafische Darstellung eines DFA

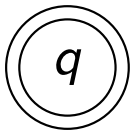
Das Zustandsdiagramm, bzw. der Automatengraph

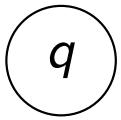
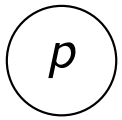
$A = (\Sigma, Q, \delta, q_0, F)$ sei ein DFA.

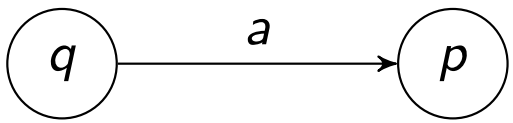
- Für jeden Zustand $q \in Q$ gibt es einen durch  dargestellten Knoten.
- Der Startzustand q_0 wird durch einen in ihn hinein führenden Pfeil markiert, d.h.:



- Jeder akzeptierende Zustand $q \in F$ wird durch eine doppelte Umrandung markiert, d.h.:



- Seien $p, q \in Q$ Zustände und $a \in \Sigma$ ein Symbol des Alphabets mit $\delta(q, a) = p$. Dann füge einen mit dem Symbol a beschrifteten Pfeil von Knoten  zu Knoten  ein, d.h.:



Wie arbeitet ein DFA?
Die erweiterte Übergangsfunktion

Wenn das Wort w gelesen wird ...

Sei $A := (\Sigma, Q, \delta, q_0, F)$ ein DFA. Die Funktion

$$\delta \ : \ Q \times \Sigma^* \ \rightarrow \ Q$$

soll den Zustand $\delta(q, w)$ angeben, den man erhält wenn A im Zustand $q \in Q$ startet und dann das Wort $w \in \Sigma^*$ liest. δ ist rekursiv wie folgt definiert:

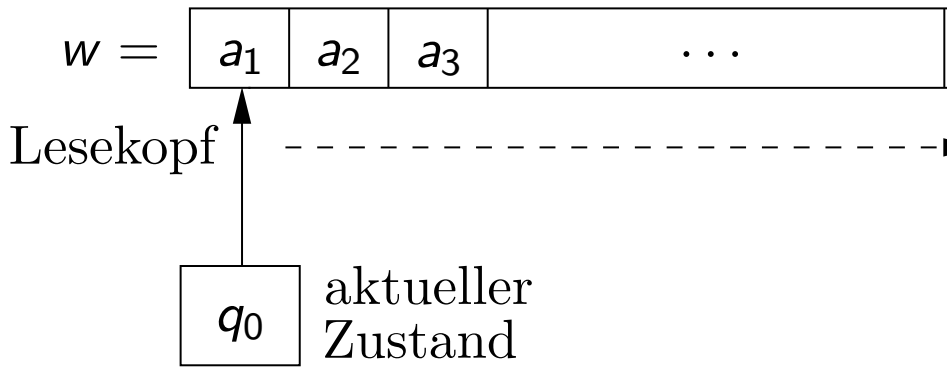
- Der **Rekursionsanfang**: Für alle $q \in Q$ ist $\delta(q, \varepsilon) := q$.
- Der **Rekursionsschritt**:
  Für alle $q \in Q$, $w \in \Sigma^*$ und $a \in \Sigma$ gilt für $q' := \delta(q, w)$:

$$\delta(q, wa) := \delta(q', a).$$

Insgesamt gilt: $\delta(q_0, w)$ ist der durch Verarbeiten des Worts w erreichte Zustand.

DFAs: Die Maschinensichtweise

Verarbeitung eines Eingabeworts durch einen DFA A:



Wir können uns einen DFA als eine Maschine vorstellen, die

* ihre Eingabe $w = a_1 a_2 a_3 \cdots a_n$ von links-nach-rechts mit Hilfe eines Lesekopfes durchläuft
* und dabei Zustandsübergänge durchführt.

Können wir heutige Rechner durch DFAs modellieren?

Die Sprache eines DFA

Wann akzeptiert $A = (\Sigma, Q, \delta, q_0, F)$ ein Wort w?

Das Eingabewort w wird vom DFA A **akzeptiert**, falls

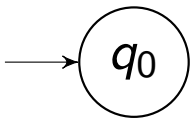
$$\delta(q_0, w) \in F,$$

d.h., der nach Verarbeiten von w erreichte Zustand gehört zur Menge F der akzeptierenden Zustände.

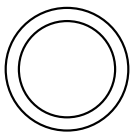
Wie „übersetzt“ sich

$$\text{Akzeptanz von } w = a_1 \cdots a_n$$

in der grafischen Darstellung von A? Es gibt einen in



startenden Weg der Länge n,

- dessen Kanten mit den Symbolen $a_1, \ldots, a_n$ beschriftet sind,
- und der in einem akzeptierenden Zustand  endet.

Reguläre Sprachen

Die Sprache $L(A)$ und reguläre Sprachen

Die von einem DFA $A = (\Sigma, Q, \delta, q_0, F)$ akzeptierte Sprache **L(A)** ist

$$L(A) := \{ w \in \Sigma^* \quad : \quad \delta(q_0, w) \in F \}.$$

Ein Wort $w \in \Sigma^*$ gehört also genau dann zur Sprache $L(A)$, wenn w vom DFA A akzeptiert wird.

Eine Sprache L heißt **regulär**, wenn es einen DFA A gibt mit

$$L(A) = L.$$

Wir möchten die Klasse aller regulären Sprachen verstehen.

Das Pattern Matching Problem

Sei Σ ein Alphabet und $w \in \Sigma^*$ ein Wort über Σ.

$$L_w = \{u\ w\ v\ :\ u, v \in \Sigma^*\}$$

ist das „Pattern Matching Problem“:

Ist das **Pattern** w ein Teilwort des **Textes** T?

? Ist die Sprache L_w regulär?

? Und wenn ja, was ist die kleinstmögliche Zustandszahl eines DFA A_w mit $L(A_w) = L_w$?

Modulo-Rechnen

m und a seien natürliche Zahlen. Für das Alphabet $\Sigma = \{0, 1, \ldots, a-1\}$ ist

$$L_{a,m} = \left\{ w \in \Sigma^* : \sum_{i=1}^{|w|} w_i \, a^{|w|-i} \equiv 0 \pmod{m} \right\}.$$

eine „Modulo-Sprache“.

? Ist die Modulosprache $L_{a,m}$ regulär?

? Und wenn ja, was ist die kleinstmögliche Zustandszahl eines DFA $A_{a,m}$ mit $L(A_{a,m}) = L_{a,m}$?

Zählen

Definiere die „Sprache L des Zählens“ durch

$$L \quad = \quad \left\{ a^n b^n \ : \ n \in \mathbb{N} \right\}.$$

? Ist L regulär?

? Und wenn ja, was ist die kleinstmögliche Zustandszahl eines DFA A mit $L(A) = L$?

Minimierung von DFAs

Minimiere die Zustandszahl

$A = (\Sigma, Q^A, \delta^A, q_0^A, F^A)$ und $B = (\Sigma, Q^B, \delta^B, q_0^B, F^B)$ seien DFAs.

(a) Wir nennen A und B **äquivalent**, wenn gilt

$$\mathbf{L(A) = L(B)}.$$

(b) A heißt

minimal,

wenn kein mit A äquivalenter DFA eine kleinere Zustandszahl besitzt.

DAS ZIEL:

- Gegeben ist ein DFA A.
- Bestimme einen minimalen, mit A äquivalenten DFA.

Zustandsminimierung: Die Idee

Der DFA $A = (\Sigma, Q, \delta, q_0, F)$ sei gegeben.

Die Idee: Wir sollten doch zwei Zustände $p, q \in Q$ zu einem einzigen Zustand verschmelzen dürfen, wenn p und q „äquivalent“ sind,

also dasselbe Ausgabeverhalten besitzen.

Was bedeutet das?

Die **Verschmelzungsrelation** $\equiv_A$ ist eine 2-stellige Relation über der Zustandsmenge Q. Wir sagen, dass Zustände $p, q \in Q$ äquivalent bzgl. A sind, (kurz $p \equiv_A q$), wenn für alle Worte $w \in \Sigma^*$:

$$\delta(p, w) \in F \;\Leftrightarrow\; \delta(q, w) \in F.$$

Wir nennen $\equiv_A$ **Verschmelzungsrelation**, da wir bzgl. $\equiv_A$ äquivalente Zustände in einen Zustand verschmelzen möchten.

Die Verschmelzungsrelation ist eine Äquivalenzrelation

Der DFA $A = (\Sigma, Q, \delta, q_0, F)$ sei gegeben. Zur Erinnerung:

$$p \equiv_A q \text{ wenn f.a. Worte } w \in \Sigma^* \text{ gilt: } \Big(\delta(p,w) \in F \Leftrightarrow \delta(q,w) \in F\Big).$$

(a) Die Verschmelzungsrelation ist

- **reflexiv**, f.a. $p \in Q$: $p \equiv_A p$,
- **symmetrisch**, f.a. $p, q \in Q$: wenn $p \equiv_A q$, dann $q \equiv_A p$ und
- **transitiv**, f.a. $p, q, r \in Q$: wenn $p \equiv_A q$ und $q \equiv_A r$, dann $p \equiv_A r$.

(b) Die Verschmelzungsrelation ist eine Äquivalenzrelation!

Können wir die Zustände einer Äquivalenzklasse von $\equiv_A$ zu einem einzigen Zustand verschmelzen?.

Was ist zu tun?

Sei der DFA $A = (\Sigma, Q, \delta, q_0, F)$ gegeben.

Wir führen die folgenden Schritte durch:

1. Wir bestimmen die Äquivalenzklassen der Verschmelzungsrelation $\equiv_A$.
2. Für jede Äquivalenzklasse von $\equiv_A$ verschmelzen wir alle Zustände der Klasse zu einem einzigen Zustand und fügen „entsprechende“ Übergänge ein. Den neuen Automaten nennen wir $\mathbf{A'}$ und bezeichnen ihn als

   den **Äquivalenzklassenautomaten** von A.

Die wichtigen Fragen

(a) Wie sollen wir die Zustandsübergänge von A' definieren, so dass A und A' dieselbe Sprache berechnen?

(b) Können wir
- die Verschmelzungsrelation $\equiv_A$ wie auch den
- Äquivalenzklassenautomaten A' effizient berechnen?

(c) Die Anzahl der Zustände von A' stimmt mit dem Index von $\equiv_A$ überein.
- Stimmt der Index mit der minimalen Zustandszahl überein?
- Wenn ja, dann ist A' **minimal**!

Wir bestimmen die Verschmelzungsrelation $\equiv_A$

Zeugen (1/2)

Sei $(\Sigma, Q, \delta, q_0, F)$ ein DFA.

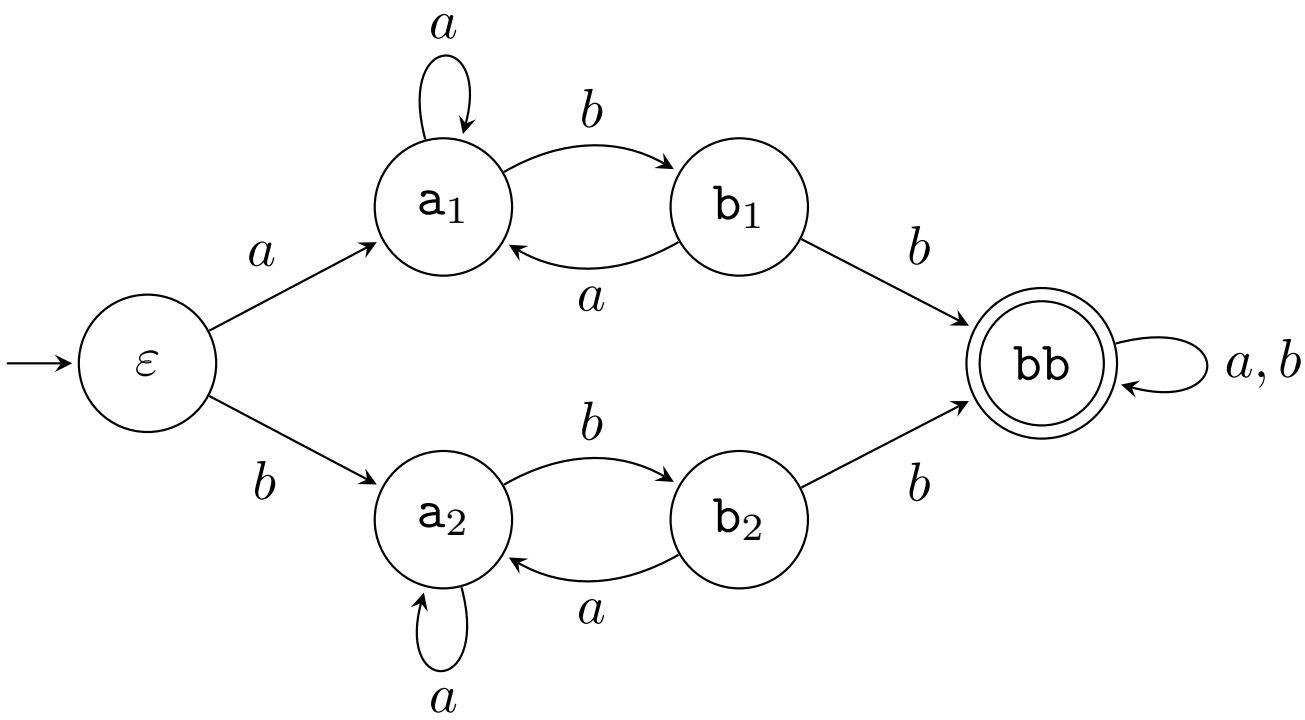
(a) Das Wort $w \in \Sigma^*$ ist **Zeuge** für die Inäquivalenz von p und q, wenn

$$\Big(\delta(p,w) \in F \wedge \delta(q,w) \notin F\Big) \vee \Big(\delta(p,w) \notin F \wedge \delta(q,w) \in F\Big).$$

Wir sagen auch, dass w die Zustände p und q **trennt**.

(b) Es ist $p \not\equiv_A q$ genau dann, wenn es einen Zeugen für die Inäquivalenz von p und q gibt, bzw wenn es ein Wort gibt, das p und q trennt.

Zeugen (2/2)



- Finde einen Zeugen für die Inäquivalenz von $\mathtt{b}_1$ und bb.
- Welcher Zeuge trennt ε und $\mathtt{a}_1$?
- Gibt es Zeugen, die die Zustände $\mathtt{a}_1$ und $\mathtt{a}_2$ trennen?

Wir bestimmen alle Paare **nicht-äquivalenter** Zustände

1. **Markiere** alle Paarmengen $\{p, q\}$ mit $p \in F$ und $q \notin F$ (als nicht-äquivalent).
   - Es ist $\delta(p, \varepsilon) \in F$ und $\delta(q, \varepsilon) \notin F$.
   - $w = \varepsilon$ ist **Zeuge** für die Nicht-Äquivalenz von p und q.
2. Wenn die Paarmenge $\{p, q\}$ bereits markiert wurde und

   $$\text{wenn } \delta(r, a) = p \text{ sowie } \delta(s, a) = q$$

   für ein $a \in \Sigma$, dann **markiere** $\{r, s\}$.
   - Da $p \not\equiv_A q$, gibt es einen **Zeugen** w mit

     $$\big(\delta(p, w) \in F \text{ und } \delta(q, w) \notin F\big) \text{ oder } \big(\delta(p, w) \notin F \text{ und } \delta(q, w) \in F\big).$$

   - Das Wort aw **trennt** r und s.
3. Halte, wenn keine neuen Paarmengen $\{r, s\}$ markiert werden können.
   - Unser Verfahren behauptet, dass $p \not\equiv_A q$ genau dann gilt, wenn die Paarmenge $\{p, q\}$ markiert wurde.

Stimmt die Behauptung: Finden wir alle Paare nicht-äquivalenter Zustände?

Unser Verfahren funktioniert!

Sei P die Menge aller Paare $\{p, q\}$ nicht-äquivalenter Zustände, die aber von unserem Verfahren **nicht** gefunden werden. **Zeige**, dass P leer ist!

Angenommen, P ist nicht-leer.

0. $\{p, q\} \in P$ habe unter allen Paaren in P einen **kürzesten** Zeugen w.
1. Wenn $w = \varepsilon$, dann ist
   - $\big(\delta(p, \varepsilon) \in F$ und $\delta(q, \varepsilon) \notin F\big)$ oder $\big(\delta(p, \varepsilon) \notin F$ und $\delta(q, \varepsilon) \in F\big)$,
   - bzw. $\big(p \in F$ und $q \notin F\big)$ oder $\big(p \notin F$ und $q \in F\big)$.

   Aber dann haben wir $\{p, q\}$ im Schritt 1. markiert. ↯
2. Wenn $w = au$ für den Buchstaben $a \in \Sigma$, dann ist
   - $\big(\delta(p, au) \in F$ und $\delta(q, au) \notin F\big)$ oder $\big(\delta(p, au) \notin F$ und $\delta(q, au) \in F\big)$,
   - bzw. $\big(\delta(\delta(p, a), u) \in F$ und $\delta(\delta(q, a), u) \notin F\big)$ oder $\big(\delta(\delta(p, a), u) \notin F$ und $\delta(\delta(q, a), u) \in F\big)$.

   Aber dann ist $\delta(p, a) \not\equiv_A \delta(q, a)$ mit dem **kürzeren** Zeugen u:
   - Nach Annahme haben wir $\{\delta(p, a), \delta(q, a)\}$ z.B. in einer Menge M_i markiert
   - und werden darauf folgend $\{p, q\}$ in M_{i+1} markieren. ↯

Der Automat nach allen Verschmelzungen

Wie sieht der Automat nach dem Verschmelzen aller äquivalenten Zustände aus?

- Für Zustand $p \in Q$ bezeichnet

$$[p]_A := \{ q \in Q \mid p \equiv_A q \}$$

die Äquivalenzklasse von p.

- Der **Äquivalenzklassenautomat** A' für A besitzt
  - die Zustandsmenge

$$Q' := \{ [p]_A \mid p \in Q \},$$

  - den Anfangszustand $q_0' := [q_0]_A$,
  - die Menge $F' := \{ [p]_A \mid p \in F \}$ der akzeptierenden Zustände und
  - das Programm δ' mit

$$\delta'([p]_A, a) := [\ \delta(p, a)\ ]_A$$

für alle $q \in Q$, $a \in \Sigma$.

Der Minimierungsalgorithmus

Berechnung von $\equiv_A$ und A'

Eingabe: Ein DFA $A = (Q, \Sigma, \delta, q_0, F)$.

Schritt 1: Entferne aus A alle **überflüssigen** Zustände, d.h. alle Zustände, die nicht von q_0 aus erreichbar sind.

Schritt 2: Bestimme alle Paarmengen $\{p, q\}$ mit $p, q \in Q$ und $p \not\equiv_A q$:

1. Markiere alle Paarmengen in $M_0 := \Big\{ \{p, q\} \ : \ p \in F, \ q \in Q \setminus F \Big\}$; Setze $i := 0$
2. Wiederhole
3. Für alle Paarmengen $\{p, q\}$ mit $p \neq q$ und für alle $x \in \Sigma$ tue folgendes:
4. Markiere $\{p, q\}$, falls $\{\delta(p, x), \delta(q, x)\} \in M_i$.
5. Sei M_{i+1} die Menge aller hierbei **neu** markierten Paarmengen.
6. $i := i + 1$
7. bis $M_i = \emptyset$
8. **Ausgabe:** $M := M_0 \cup \cdots \cup M_{i-1}$.

Schritt 3: Konstruiere $A' := (Q', \Sigma, \delta', q_0', F')$:

$Q' := \Big\{ [q]_A \ : \ q \in Q \Big\}$, wobei $[q]_A = \Big\{ p \in Q \ : \ \{p, q\} \notin M \Big\}$

$q_0' := [q_0]_A, \quad F' := \Big\{ [q]_A \ : \ q \in F \Big\}$

$\delta' : Q' \times \Sigma \to Q' \quad \text{mit} \quad \delta'\big([q]_A, x\big) := [\delta(q, x)]_A \quad \text{für alle } q \in Q \text{ und } x \in \Sigma.$

Ist der Algorithmus korrekt und effizient?

In jeder Iteration in Schritt 2 wird mindestens eine neue Paarmenge markiert:
Es gibt also höchstens $|Q|^2$ Iterationen und der Algorithmus ist effizient (?!).

Die wichtigen Fragen:

1. Sind A und der Äquivalenzklassenautomat A' äquivalent, d.h. berechnet A' dieselbe Sprache wie A? Zeige, dass die Definition

$$\delta'([p], a) := [\delta(p, a)]_A$$

„wohldefiniert“ ist, also nicht vom Repräsentanten p abhängt:

   - Wenn $p \equiv_A q$, dann ist $\delta(p, a) \equiv_A \delta(q, a)$. ✓
   - Zeige durch Induktion über die Länge von w, dass

$$\delta'([p], w) = [\delta(p, w)]_A$$

   - sowie $p \equiv_A q$ und $p \in F \Rightarrow q \in F$.✓

2. **Ist A' minimal?**

Aber zuerst entwerfen wir „wirklich-effiziente“ Minimierungsalgorithmen.

Die Algorithmen von Moore und Hopcroft

Berechnung der Verschmelzungsrelation: Der Ansatz (1/2)

- Der Verschmelzungsalgorithmus beginnt mit der Menge M_0 aller Paare, die durch das leere Wort getrennt werden.
- Mit vollständiger Induktion: M_i ist die Menge aller Paare mit einem Zeugen der Länge i.

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein DFA. Für jede natürliche Zahl i definiere die

Verschmelzungsrelation $\equiv_A^i$ der Ordnung i:

Setze für alle Zustände $p, q \in Q$

$$p \equiv_A^i q \quad :\Longleftrightarrow \quad \text{für alle Worte } w \in \Sigma^* \text{ **der Länge höchstens** } i \text{ gilt } (\, \delta(p,w) \in F \Longleftrightarrow \delta(q,w) \in F \,).$$

$$\text{Also: } p \equiv_A^i q \iff \{p,q\} \notin \bigcup_{j,\, j \leqslant i} M_j,$$

denn M_j ist die Menge aller Paare mit einem Zeugen der Länge j.

Berechnung der Verschmelzungsrelation: Der Ansatz (2/2)

Der Verschmelzungsalgorithmus berechnet nacheinander die Äquivalenzrelationen

$$\equiv_A^0, \equiv_A^1, \ldots, \equiv_A^k,$$

wobei die letzte Relation, also $\equiv_A^k$, die Verschmelzungsrelation ist.

- Die Äquivalenzrelation $\equiv_A^0$ besitzt die beiden Klassen $Q \setminus F$ und F.
- Betrachte in der $i$ten Iteration alle noch nicht getrennten Zustände p, q.
  - p, q noch nicht getrennt $\Longrightarrow$ Es gilt $p \equiv_A^{i-1} q$.
  - Gilt $\delta(p, a) \not\equiv_A^{i-1} \delta(q, a)$ für irgendeinen Buchstaben a, folgt $p \not\equiv_A^i q$.
    - ⋆ Werden $\delta(p, a)$ und $\delta(q, a)$ von dem Zeugen w getrennt, dann werden p und q von dem Zeugen aw getrennt.

Der Algorithmus von Moore

1. Der DFA $A = (\Sigma, Q, \delta, q_0, F)$ ist gegeben.
   - $Z_0 = (F, Q \setminus F)$ ist die Zerlegung der Verschmelzungsrelation der Ordnung 0.
   - Setze $i = 0$.
2. Wiederhole bis $Z_i = Z_{i+1}$:
   (a) Z_i besteht aus den Äquivalenzklassen von $\equiv_A^i$.
   (b) Für alle Buchstaben $a \in \Sigma$ und alle Äquivalenzklassen P von Z_i: Bestimme die Zerlegung $Z_i(a)$ der Äquivalenzrelation $\equiv_A^{i,a}$ mit

$$r \equiv_A^{i,a} s \;:\Longleftrightarrow\; r \equiv_A^i s \text{ und } \delta(r,a) \equiv_A^i \delta(s,a).$$

   (c) Bestimme die Zerlegung Z_{i+1} von $\equiv_A^{i+1}$ als Verfeinerung der Zerlegungen $Z_i(a)$. (Beachte

$$r \equiv_A^{i+1} s \iff r \equiv_A^{i,a} s \textbf{ für alle } a \in \Sigma.)$$

   (d) Setze $i = i + 1$.

Der Algorithmus von Moore: Die Laufzeit

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein DFA mit $n = |Q|$. Setze $L := L(A)$.

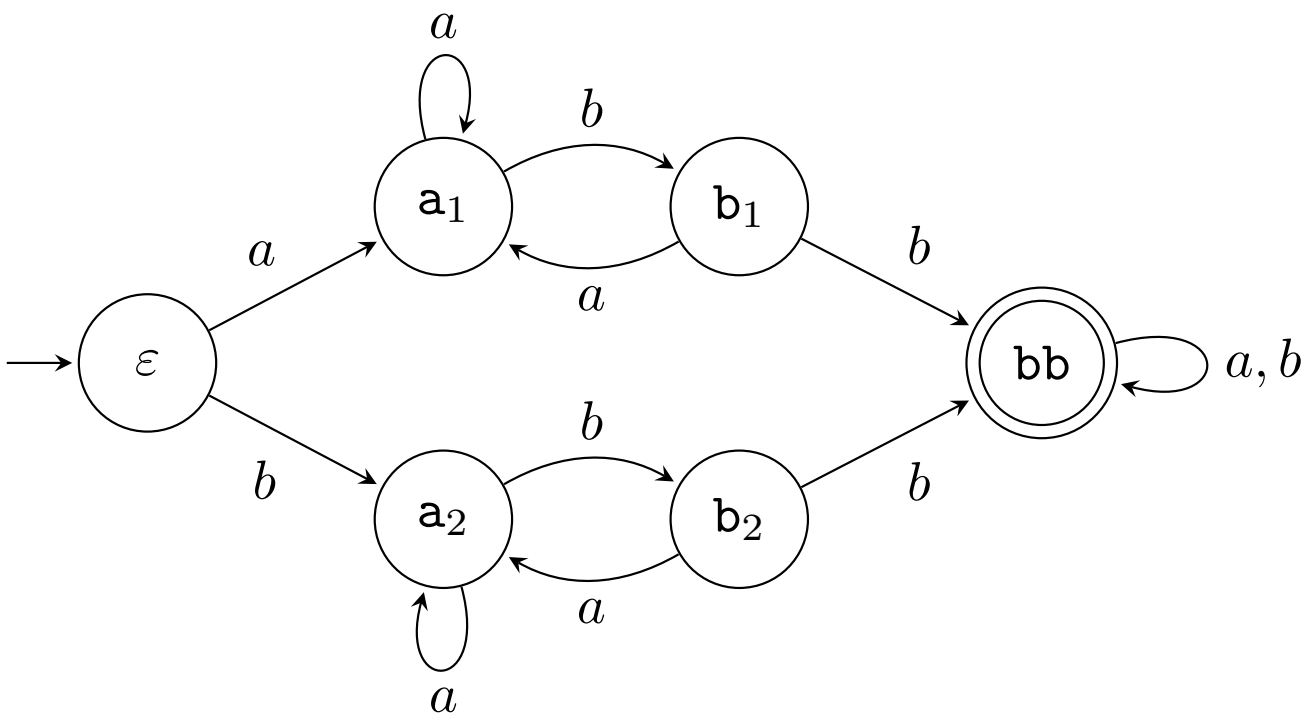
1. Definiere die **Tiefe** t von A als die kleinste Zahl i, so dass $\equiv_A^i$ und $\equiv_A^{i+1}$ übereinstimmen.
2. Übungsaufgabe:
   - Die Tiefe stimmt für alle Automaten B mit $L(B) = L(A)$ überein.
   - Für jede natürliche Zahl n gibt es einen DFA A mit n Zuständen und Tiefe $n-2$. Es gibt keinen DFA mit n Zuständen und Tiefe größer als $n-2$.
   - Die Zerlegung $Z_i(a)$ kann in Zeit $O(n)$ bestimmt werden, wenn Z_i bekannt ist.
   - Die Zerlegung Z_{i+1} kann in Zeit $O(|\Sigma| \cdot n)$ bestimmt werden, wenn die Zerlegungen $Z_i(a)$ für alle Buchstaben $a \in \Sigma$ bekannt sind.

Moores Algorithmus bestimmt die Verschmelzungsrelation $\equiv_A$ in Zeit

$$O(t \cdot |\Sigma| \cdot n),$$

wobei t die Tiefe von $L = L(A)$ ist.

Wie geht Moores Algorithmus vor?



Hopcrofts Algorithmus: Die Idee

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein DFA und seien $X, Y \subseteq Q$ Teilmengen von Q, wobei keine zwei Zustände $y_1 \in Y$ und $y_2 \notin Y$ äquivalent seien.

(a) Wir sagen, dass Y die Menge X (für den Buchstaben c) **zerlegt**, falls es $x_1, x_2 \in X$ mit $\delta(x_1, c) \in Y$ und $\delta(x_2, c) \notin Y$ gibt.

- $\delta(x_1, c)$ und $\delta(x_2, c)$ sind nach Annahme nicht äquivalent.
- $\Longrightarrow$ x_1 und x_2 können nicht äquivalent sein.

(b) Setze

$$\delta_c^{-1}(Y) := \{\, q \in Q \,:\, \delta(q, c) \in Y \,\}.$$

(c) Zerlege X in die beiden Klassen

$$X \cap \delta_c^{-1}(Y) \text{ und } X \setminus \delta_c^{-1}(Y).$$

Hopcrofts Algorithmus

1. Der DFA $A = (\Sigma, Q, \delta, q_0, F)$ ist gegeben.
   (a) $Z = (F, Q \setminus F)$ ist die Anfangszerlegung.
   (b) Setze Check $:= \{Y\}$, wobei Y die kleinere der Mengen $F, Q \setminus F$ ist.
   /* Die in Check enthaltenen Klassen von Z genügen, um alle aktuell zerlegbaren Klassen zu zerlegen. */
2. Solange Check nichtleer ist, wiederhole
   (a) Wähle irgendeine Klasse Y aus Check und entferne Y aus Check.
   /* Welche Klassen X werden von Y zerlegt? */
   (b) Für jeden Buchstaben $c \in \Sigma$: bestimme $\delta_c^{-1}(Y) = \{ q \in Q : \delta(q, c) \in Y \}$
   und bestimme alle Klassen X in Z, die von Y zerlegt werden:
   - Ersetze jede zerlegte „Mutterklasse“ X in Z durch die „Kinderklassen“ $X \cap \delta_c^{-1}(Y)$ und $X \setminus \delta_c^{-1}(Y)$.
   - Wenn $X \in$ Check, dann ersetze X in Check durch $X \cap \delta_c^{-1}(Y)$ und $X \setminus \delta_c^{-1}(Y)$.
   - Ansonsten, wenn $X \notin$ Check, dann **füge die kleinere der beiden Mengen** $X \cap \delta_c^{-1}(Y)$ und $X \setminus \delta_c^{-1}(Y)$ zu Check hinzu.

Invarianten

Während der Berechnung von Hopcrofts Algorithmus gilt:

(a) Die Äquivalenzklassen der Verschmelzungsrelation $\equiv_A$ **verfeinern** Z, d.h. jede Klasse von Z ist die Vereinigung bestimmter Äquivalenzklassen von $\equiv_A$.

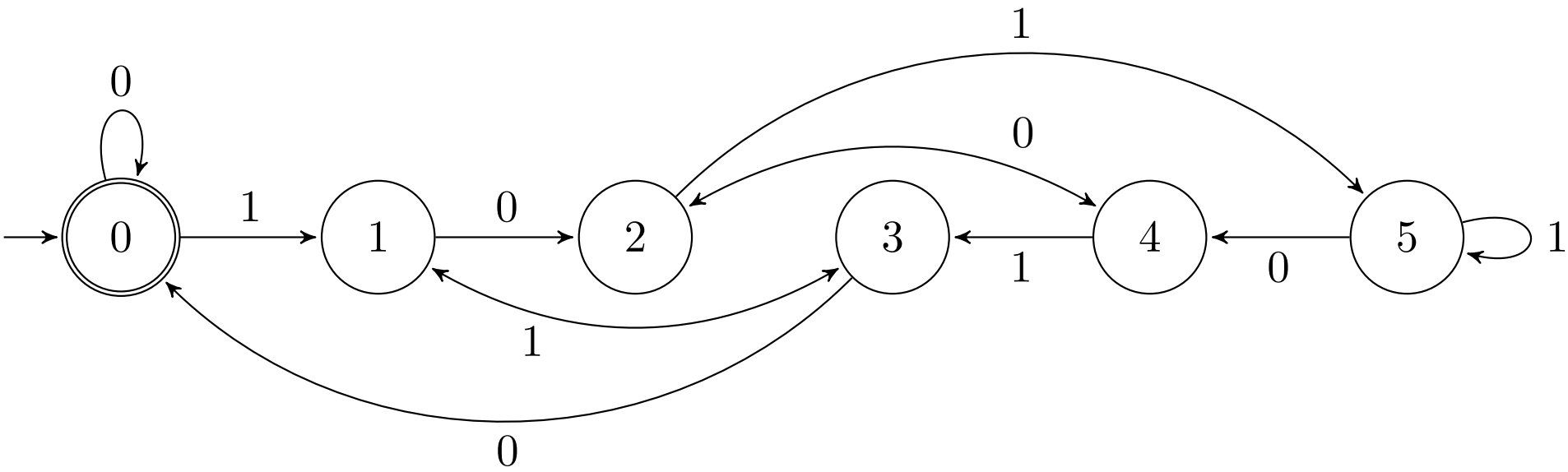
(b) Die Klassen X in Z stimmen genau dann mit den Klassen der Verschmelzungsrelation überein, wenn

$$X \cap \delta_c^{-1}(Y) = X \quad \text{oder} \quad X \cap \delta_c^{-1}(Y) = \emptyset$$

für alle Klassen X in Z und alle Klassen Y in Check gilt.

Wie geht Hopcrofts Algorithmus vor?

Der DFA A mit Zustandsdiagramm



akzeptiert die Binärdarstellungen aller durch 6 teilbaren Zahlen. D.h.:

$$L(A) \quad = \quad \left\{ w \in \{0,1\}^* \ : \ \sum_{i=1}^{|w|} w_i \, 2^{|w|-i} \equiv 0 \ (\text{mod } 6) \right\}.$$

Aufgabe: Ist der Automat minimal? Bestimme den Äquivalenzklassenautomaten A'.

Hopcrofts Algorithmus: Korrektheit

- Ist die Aktualisierung von Z korrekt?
  - Wird eine Mutterklasse X der Zerlegung Z zerlegt, dann können Zustände in der einen Kinderklasse $X \cap \delta_c^{-1}(Y)$ nicht mit Zuständen in der anderen Kinderklasse $X \setminus \delta_c^{-1}(Y)$ äquivalent sein. ✓
- Ist die Aktualisierung der Menge Check korrekt?
  - Wird eine Menge Y aus Check entfernt, dann wird Y, nach seiner Verarbeitung in Schritt 2b), keine Klassen in Z mehr zerlegen. ✓
  - Und wenn eine Klasse X durch Y zerlegt wird?
    - ⋆ Gehört X zu Check, dann wird die Mutterklasse X in Check durch beide Kinder $X \cap \delta_c^{-1}(Y), X \setminus \delta_c^{-1}(Y)$ ersetzt.
    - ⋆ Und wenn X nicht zu Check gehört? Warum darf X in Check durch die kleinere der beiden Kinderklassen ersetzt werden?

Hopcrofts Algorithmus: Die Laufzeit (1/2)

Ein Zustandsübergang $\delta(q, c) = r$ **gehört** genau dann zu einer Klasse Y (der aktuellen Zerlegung Z), wenn $r \in Y$.

1. Sei
$$t_c(Y) := |\delta_c^{-1}(Y)|$$
die Anzahl der Übergänge, die zu Y gehören.
2. Übungsaufgabe: Y kann in Zeit $O(\sum_{c \in \Sigma} |t_c(Y)|)$ verarbeitet werden.
3. Wenn die Menge Y mit $r \in Y$ aus Check entfernt wird und eine andere Menge Y' mit $r \in Y'$ irgendwann später zu Check hinzugefügt wird, dann ist
$$|Y'| \leqslant |Y|/2.$$
Y' wurde als kleinere von zwei Kinderklassen zu Check hinzugefügt.

Hopcrofts Algorithmus: Die Laufzeit (2/2)

(4) Sei C die Menge aller Klassen Y, die irgendwann zur Menge Check gehört haben. Dann gehört jeder Zustandsübergang zu höchstens

$$\log_2 n$$

verschiedenen Mengen Y in Check, und wir erhalten für die Gesamtlaufzeit

$$\begin{aligned}
\sum_{c\in\Sigma}\sum_{Y\in C} t_c(Y) &= \sum_{c\in\Sigma}\sum_{Y\in C}\left(\sum_{q\in Q\,:\,\delta(q,c)\in Y} 1\right)\\
&= \sum_{c\in\Sigma}\sum_{q\in Q}\left(\sum_{Y\in C\,:\,\delta(q,c)\in Y} 1\right)\\
&= \sum_{(c,q)\in\Sigma\times Q} \log_2 n\\
&= |\Sigma|\cdot n\cdot \log_2 n.
\end{aligned}$$

Moore versus Hopcroft

Warum ist Hopcrofts Algorithmus schneller als Moores Algorithmus?

- Moore bestimmt nacheinander die **vollständigen Zerlegungen** $\equiv_A^i$.
- Hopcrofts Algorithmus wird von den **einzelnen** Klassen $Y \in$ Check getrieben und alle von Y zerlegten Klassen X werden in ihre Kinderklassen aufgeteilt.
- **Wesentlich**:
  - Ein Zustandsübergang gehört zu logarithmisch vielen Klassen: Eine zerlegte Mutterklasse, die nicht zu Check gehört, wird nur durch die kleinere Kinderklasse ersetzt!
  - Eine Checkmenge Y kann in Zeit $O(\sum_{c\in\Sigma} |t_c(Y)|)$ verarbeitet werden.

Die Myhill-Nerode Relation

Was ist Sache?

1. Der ursprüngliche Automat $A = (Q, \Sigma, \delta, q_0, F)$ und sein Äquivalenzklassenautomat A' sind äquivalent.
2. Wir können A' effizient berechnen.

? Aber hat A' unter allen mit A äquivalenten DFAs die kleinste Zustandszahl?

Die Myhill-Nerode Relation für die Sprache $L = L(A)$ hat die Antwort.

Die Myhill-Nerode Relation

Sei $A = (\Sigma, Q, \delta, q_0, F)$ ein DFA. Wenn $\delta(q_0, u) = \delta(q_0, v)$, dann gilt

$$\text{für alle } w \in \Sigma^*: \quad uw \in L(A) \Leftrightarrow vw \in L(A).$$

L sei eine Sprache über der endlichen Menge Σ, d.h. es gilt $L \subseteq \Sigma^*$.

(a) Die **Myhill-Nerode Relation** $\equiv_L$ für L ist eine 2-stellige Relation über Σ^*. Für alle Worte $u, v \in \Sigma^*$ definiere

$$u \equiv_L v \;:\Leftrightarrow\; \text{für alle } w \in \Sigma^* \text{ gilt: } \big(uw \in L \Leftrightarrow vw \in L\big).$$

(b) Wir sagen, dass das Wort $w \in \Sigma^*$ die Worte $u, v \in \Sigma^*$ **trennt**, bzw. dass w ein **Zeuge** für die Inäquivalenz von u und v ist, wenn

$$\Big(uw \in L \wedge vw \notin L\Big) \vee \Big(uw \notin L \wedge vw \in L\Big).$$

(c) **Index(L)** ist die Anzahl der Äquivalenzklassen von $\equiv_L$.

Die Myhill-Nerode Relation ist eine Äquivalenzrelation

Warum? Die Myhill-Nerode Relation $\equiv_L$ ist

1. reflexiv, denn $u \equiv_L u$ gilt f.a. $u \in \Sigma^*$,
2. symmetrisch, denn aus $u \equiv_L v$ folgt $v \equiv_L u$ f.a. $u, v \in \Sigma^*$,
3. transitiv, denn aus $u \equiv_L v$, $v \equiv_L w$ folgt $u \equiv_L w$ f.a. $u, v, w \in \Sigma^*$.

Der absolute Hammer:

Für eine reguläre Sprache L stimmt **Index(L)**,

die Anzahl der Äquivalenzklassen von $\equiv_L$,

mit der **minimalen Zustandszahl** für DFAs A mit $L(A) = L$ überein!

Die minimale Zustandszahl $\geqslant$ Index(L)

Der DFA $A = (Q, \Sigma, \delta, q_0, F)$ akzeptiere die Sprache L, es gelte also $L(A) = L$.

Angenommen, es ist

$$\delta(q_0, u) = \delta(q_0, v).$$

1. Dann ist $\delta(q_0, uw) = \delta(q_0, vw)$ für alle Worte $w \in \Sigma^*$ und A akzeptiert das Wort uw genau dann, wenn A das Wort vw akzeptiert.
2. Aber $L(A) = L$ und $\Big(uw \in L \iff vw \in L\Big)$ folgt für alle Worte $w \in \Sigma^*$. Also

$$u \equiv_L v.$$

(a) Alle Worte, die auf denselben Zustand von A führen, sind äquivalent bzgl. $\equiv_L$.
(b) Jeder DFA A mit $L = L(A)$ hat mindestens **Index(L)** Zustände.

Ist der Äquivalenzklassenautomat minimal?

Ja, weil:

Der DFA $A = (Q, \Sigma, \delta, q_0, F)$ akzeptiere die Sprache L, es gelte also $L(A) = L$.
$\mathbf{A'} = (Q', \Sigma, \delta', q_0', F')$ sei sein Äquivalenzklassenautomat.

Angenommen, Worte $u, v \in \Sigma^*$ führen in $\mathbf{A'}$ zu verschiedenen Zuständen.

1. Für A sei $p = \delta(q_0, u)$ und $q = \delta(q_0, v)$.
2. Es folgt $p \not\equiv_A q$ und es gibt einen Zeugen $w \in \Sigma^*$ für die Nicht-Äquivalenz.
   - Also: $\big(\delta(p, w) \in F \wedge \delta(q, w) \notin F\big) \vee \big(\delta(p, w) \notin F \wedge \delta(q, w) \in F\big)$, bzw.
   - $\big(\delta(q_0, uw) \in F \wedge \delta(q_0, vw) \notin F\big) \vee \big(\delta(q_0, uw) \notin F \wedge \delta(q_0, vw) \in F\big)$, bzw.
   - $\Big(uw \in L \wedge vw \notin L\Big) \vee \Big(uw \notin L \wedge vw \in L\Big)$.
3. Wenn u und v in $\mathbf{A'}$ zu verschiedenen Zuständen führen, dann folgt $u \not\equiv_L v$.

Die **Zustandszahl von $\mathbf{A'}$** stimmt überein mit **Index(L(A))**:

Der Äquivalenzklassenautomat $\mathbf{A'}$ ist ein minimaler DFA für L(A).

Einschub: Der Myhill-Nerode Automat

Der Myhill-Nerode Automat N_L für L

Können wir einen (sogar minimalen) DFA direkt aus der Myhill-Nerode Relation bauen?

Der **Myhill-Nerode Automat** $N_L = (Q_L, \Sigma, \delta_L, q_0, F_L)$ hat die folgenden Komponenten:

- die Zustandsmenge

$$Q_L := \left\{ [u]_L \ : \ u \in \Sigma^* \right\},$$

besteht aus den Äquivalenzklassen der Myhill-Nerode Relation $\equiv_L$.

Idee: Definiere den Automaten so, dass für alle $u \in \Sigma^*$ gilt: Der Zustand, in dem der Automat nach dem Lesen des Worts u ist, gibt die Äquivalenzklasse von u bzgl. der Myhill-Nerode Relation $\equiv_L$ an. D.h.:

$$\delta_L(q_0, u) \ = \ [u]_L$$

- Startzustand $q_0 := [\varepsilon]_L$
- akzeptierende Zustände $F_L := \left\{ [u]_L \ : \ u \in L \right\}$
- Programm δ_L mit $\delta_L\big([u]_L, a\big) := [ua]_L$ für alle $u \in \Sigma^*$ und $a \in \Sigma$.

Der Myhill-Nerode Automat ist minimal

(a) Zeige durch Induktion über die Länge des Wortes $u \in \Sigma^*$, dass

$$\delta_L([\varepsilon], u) = [u]_L$$

für alle Worte $u \in \Sigma^*$ gilt.

(b) Wenn $u \equiv_L v$ und $u \in L$, dann auch $v \in L$.

(c) Deshalb ist

$$\begin{aligned} u \in L(N_L) &\iff [u] \in F_L \\ &\iff u \in L. \end{aligned}$$

(d) Es ist $L(N_L) = L$ und N_L besitzt genau Index(L) viele Zustände.

N_L ist ein minimaler DFA für L.

Der Myhill-Nerode Automat: Beispiele

Bestimme jeweils Index(L) und den Myhill-Nerode Automaten für L.

1. $L = \{w \in \{0,1\}^* : w \text{ hat gerade viele Einsen}\}$,
2. L die Menge aller Binärdarstellungen für durch 6 teilbare Zahlen,
3. $L = \{a,b\}^*\{ab\}\{a,b\}^*$,
4. $L_u = \{w \in \Sigma^* : u \text{ ist ein Teilwort von } w\}$ für ein Wort $u \in \Sigma^*$.

Welche Sprachen sind nicht regulär?
Der Satz von Myhill-Nerode

Satz von Myhill-Nerode: Wann ist eine Sprache regulär?

Sei Σ ein Alphabet und $L \subseteq \Sigma^*$ eine Sprache über Σ.

(a) L ist **regulär** $\iff$ **Index(L) ist endlich.**

(b) Der minimale DFA für L hat Index(L) Zustände.

Wir haben Teil (b) schon gezeigt. Der Beweis von Teil (a):

$\Longrightarrow$ $L \subseteq \Sigma^*$ sei regulär. Dann gibt es einen DFA A mit $L = L(A)$.

- A hat mindestens Index(L) Zustände.
- Also ist Index(L) endlich.

$\Longleftarrow$ Index(L) sei endlich.

- Der Myhill-Nerode Automat N_L ist ein DFA für L, d.h.: $L = L(N_L)$.
- L ist regulär.

$L = \{a^n b^n \ : \ n \in \mathbb{N}\}$ ist nicht regulär

Bestimme unendlich viele Worte $u_k \in \{a, b\}^*$, so dass

$$u_k \not\equiv_L u_\ell$$

für alle $k \neq \ell$ gilt.

Setze $u_i := a^i$. Für $k \neq \ell$ gilt $u_k \not\equiv_L u_\ell$, denn

$$u_k b^k \in L, \quad \text{aber} \quad u_\ell b^k \notin L.$$

Index$(L) = \infty$ und L ist nicht regulär.

DFAs können nicht (unbeschränkt) zählen.

$L = \{ww \ : \ w \in \{a, b\}^*\}$ ist nicht regulär

Wir bestimmen unendlich viele Worte $u_k \in \{a, b\}^*$, so dass

$$u_k \not\equiv_L u_\ell$$

für alle $k \neq \ell$ gilt.

Setze $u_i := a^i b$. Für $k \neq \ell$ gilt $u_k \not\equiv_L u_\ell$, denn

$$u_k a^k b \in L, \quad \text{aber} \quad u_\ell a^k b \notin L.$$

Index$(L) = \infty$ und L ist nicht regulär.

DFAs können sich nur beschränkt viele Dinge merken.

Weitere nicht-reguläre Sprachen

Keine der folgenden Sprachen ist regulär.

- $L_1 = \{ a^n b^m \ : \ n, m \in \mathbb{N}, \ n \leqslant m \}$:
  - DFAs können nicht vergleichen,
- $L_2 = \{ a^n b^m c^{n+m} \ : \ n, m \in \mathbb{N} \}$:
  - DFAs können nicht addieren, wenn man sich dazu an die Summanden erinnern muss,
- $L_3 = \{ a^{n^2} \ : \ n \in \mathbb{N} \}$:
  - DFAs können nicht quadrieren,
- $L_4 = \{ w \in \{a, b\}^* \ : \ w \text{ ist ein Palindrom} \}$:
  - Endliche Automaten haben ein nur beschränkt großes Gedächtnis.

Das Pumping-Lemma

Vorarbeiten zum Pumping Lemma (1/2)

Wann ist eine Sprache nicht regulär?

- Angenommen, A ist ein Automat mit $|Q|$ Zuständen.
  - Die Zustandsübergänge von A auf einer Eingabe $x = x_1 \cdots x_n$:

$$q_0 \xrightarrow{x_1} q_1 \xrightarrow{x_2} q_2 \xrightarrow{x_3} \cdots \xrightarrow{x_{n-1}} q_{n-1} \xrightarrow{x_n} q_n.$$

  - Wenn $n \geqslant |Q|$, dann werden dabei $n+1 > |Q|$ Zustände durchlaufen.
  - Es gibt (mind.) einen Zustand q, der (mind.) zweimal durchlaufen wird.
- Es gibt Zahlen $0 \leqslant j < k \leqslant |Q|$, so dass

$$q_0 \xrightarrow{x_1 \cdots x_j} q \xrightarrow{x_{j+1} \cdots x_k} q \xrightarrow{x_{k+1} \cdots x_n} q_n.$$

Vorarbeiten zum Pumping Lemma (2/2)

Verarbeitung eines Worts $z = z_1 \cdots z_N$ durch DFA A:

Wenn $N \geqslant |Q|$, dann $\quad q_0 \xrightarrow{x_1 \cdots x_j} q \xrightarrow{x_{j+1} \cdots x_k} q \xrightarrow{x_{k+1} \cdots x_N} q_N$.

Und wenn $q_N \in F$, dann $x \in L(A)$ und

$$
\begin{aligned}
(x_1 \cdots x_j) \cdot (x_{k+1} \cdots x_N) &\in L(A) \quad \text{und} \\
(x_1 \cdots x_j) \cdot (x_{j+1} \cdots x_k) \cdot (x_{k+1} \cdots x_N) &\in L(A) \quad \text{und} \\
(x_1 \cdots x_j) \cdot (x_{j+1} \cdots x_k)^2 \cdot (x_{k+1} \cdots x_N) &\in L(A) \quad \text{und} \\
(x_1 \cdots x_j) \cdot (x_{j+1} \cdots x_k)^3 \cdot (x_{k+1} \cdots x_N) &\in L(A) \quad \ldots
\end{aligned}
$$

Damit haben wir Folgendes bewiesen:

Das Pumping Lemma

Sei L eine reguläre Sprache.

- Dann gibt es eine **Pumpingkonstante** $N \geqslant 1$, so dass
- jedes Wort $x \in L$ mit $|x| \geqslant N$
- eine Zerlegung mit den folgenden Eigenschaften besitzt:
  - $x = uvw, \quad |uv| \leqslant N, \quad |v| \geqslant 1, \quad$ und
  - $uv^i w \in L$ für jedes $i \geqslant 0$.

Fazit: Wenn Worte der Sprache lang genug sind (also $|x| \geqslant N$), dann gibt es ein nicht-leeres Teilwort v, (also $v = x_{j+1} \cdots x_k$) das

- „aufgepumpt“ ($i \geqslant 1$)
- und „abgepumpt“ ($i = 0$) werden kann.

Kennen wir die Pumpingkonstante N?
—NEIN!

Kennen wir die Zerlegung von x in uvw?
—NEIN! Wir wissen nur, dass $|uv| \leqslant N$ und $|v| \geqslant 1$ ist.

Anwendung des Pumping Lemmas: Die Spielregeln

Wie kann man das Pumping Lemma nutzen, um zu zeigen, dass eine Sprache L **nicht** regulär ist?

(1) Für jede mögliche Pumpingkonstante $N \geqslant 1$

(2) müssen wir ein Wort $x \in L$ mit $|x| \geqslant N$ konstruieren,

(3) so dass für jede mögliche Zerlegung $x = uvw$ mit $|uv| \leqslant N$ und $|v| \geqslant 1$

$$uv^i w \notin L$$

für mindestens ein $i \geqslant 0$ gilt.

Der "Gegner" kontrolliert die **Pumpingkonstante** N vollständig und die **Zerlegung** $x = uvw$ teilweise, denn er muss $|uv| \leqslant N$ und $|v| \geqslant 1$ garantieren.

Das Pumping Lemma: Ein Anwendungsbeispiel

Die Sprache

$$L = \left\{ w \in \{0,1\}^* \mid w = 0^n 1^m \text{ für } n \leqslant m \right\}$$

ist nicht regulär.

Beweis: Angenommen, L ist doch regulär.

1. Pumping Lemma: Es gibt eine Pumpingkonstante $N \geqslant 1$, so dass jedes Wort $x \in L$ mit $|x| \geqslant N$ eine Zerlegung in $x = uvw$ besitzt, so dass gilt:

   $$(*): \qquad |uv| \leqslant N \quad \text{und} \quad |v| \geqslant 1 \quad \text{und} \quad uv^i w \in L \text{ für alle } i \geqslant 0.$$

2. Betrachte insbesondere das Wort $x := 0^N 1^N$ mit der Zerlegung $x = uvw$.
3. Es gilt: $x \in L$ und $|x| \geqslant N$.
4. Wegen $uvw = x = 0^N 1^N$ und $|uv| \leqslant N$ sowie $|v| \geqslant 1$ gibt es eine Zahl $k \geqslant 1$, so dass $v = 0^k$.
   Aber dann ist $uv^2 w = 0^{z+k} 1^z$ kein Wort in L, obwohl gemäß Pumping Lemma dieses Wort in L liegen müsste. ↯

□

Grenzen des Pumping Lemmas

Es gibt Sprachen, die nicht regulär sind, deren Nicht-Regularität mit dem Pumping Lemma nicht nachgewiesen werden kann.

Die Sprache

$$L = \{a, b\}^* \cup \{c^k\, a^m\, b^m : k, m \geqslant 0\}$$

ist nicht regulär, erfüllt aber die Aussage des Pumping Lemmas. D.h:

Es gibt eine Pumpingkonstante $N \geqslant 1$ (nämlich $N = 1$), so dass

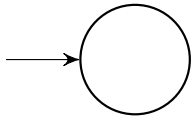
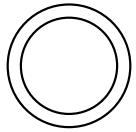
- jedes Wort $x \in L$ mit $|x| \geqslant N$
- eine Zerlegung mit den folgenden Eigenschaften besitzt:
  - $x = uvw, \quad |uv| \leqslant N, \quad |v| \geqslant 1, \quad$ und
  - $uv^i w \in L$ für jedes $i \geqslant 0$.

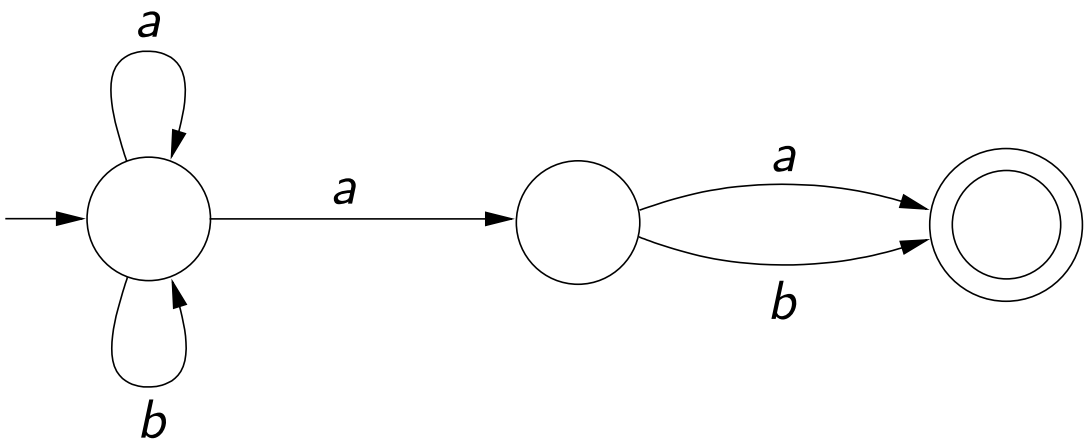
Beweis: Siehe Tafel!

NFAs

NFAs: DFAs, die raten dürfen

Ein „NFA“ akzeptiert ein Eingabewort $w \in \{a, b\}^*$ **genau dann**, wenn es im Automatengraphen **mindestens einen Weg gibt**,

- der im Startzustand  beginnt,
- dessen Kanten mit w beschriftet sind,
- und der in einem akzeptierenden Zustand  endet.



Der „NFA“ akzeptiert

$$L = \{w \in \{a, b\}^* \ : \text{ der vorletzte Buchstabe von } w \text{ ist ein } a \ \}.$$

NFAs: Die formale Definition

Ein nichtdeterministischer endlicher Automat (kurz: **NFA**)

$$A = (\Sigma, Q, \delta, q_0, F)$$

besteht aus:

- einer endlichen Menge Σ, dem Eingabealphabet,
- einer endlichen Menge Q, der Zustandsmenge,
- einer Funktion $\delta\colon Q \times \Sigma \to \mathcal{P}(Q)$, der Übergangsfunktion,
  - *die jedem Zustand $q \in Q$ und jedem Symbol $a \in \Sigma$ eine **Menge** $\delta(q, a)$ von möglichen Nachfolgezuständen zuordnet.*
  - *Möglicherweise ist $\delta(q, a) = \emptyset$: Dann „stürzt“ der Automat ab, wenn er im Zustand q ist und das Symbol a liest.*
- dem Startzustand $q_0 \in Q$ und
- einer Menge $F \subseteq Q$ von Endzuständen bzw. akzeptierenden Zuständen.

Die von einem NFA akzeptierte Sprache und die erweiterte Übergangsfunktion

$\delta(q, w) :=$ die MENGE aller möglichen Zustände nach Lesen von w im „Startzustand“ q

Sei $A := (\Sigma, Q, \delta, q_0, F)$ ein NFA. Die Funktion

$$\delta \ : \ Q \times \Sigma^* \ \to \ \mathcal{P}(Q)$$

ist rekursiv wie folgt definiert:

- **Rekursionsanfang**: F.a. $q \in Q$ ist $\delta(q, \varepsilon) \ := \{q\}$.
- **Rekursionsschritt**: F.a. $q \in Q$, $w \in \Sigma^*$ und $a \in \Sigma$ ist

$$\delta(q, wa) \ := \bigcup_{q' \in \delta(q,w)} \delta(q', a).$$

Ein möglicher Zustand q'' wird nach Lesen von wa genau dann erreicht, wenn nach Lesen von w (im Zustand q) ein Zustand q' erreicht wird und $q'' \in \delta(q', a)$ gilt:

$$q \stackrel{w}{\to} q' \stackrel{a}{\to} q''.$$

Wann akzeptiert ein NFA?

Der NFA $A = (\Sigma, Q, \delta, q_0, F)$ akzeptiert ein Wort w genau dann, wenn

$$\delta(q_0, w) \cap F \neq \emptyset.$$

Somit ist

$$L(A) = \{ w \in \Sigma^* : \delta(q_0, w) \cap F \neq \emptyset \}.$$

Äquivalenz von NFAs und DFAs

Akzeptieren NFAs nicht-reguläre Sprachen?

NEIN!

Für jeden NFA $A = (\Sigma, Q, \delta, q_0, F)$ gibt es einen DFA $A' = (\Sigma, Q', \delta', q_0', F')$ mit

$$L(A') = L(A).$$

D.h.: NFAs und DFAs akzeptieren genau dieselben Sprachen.

Aber im Allgemeinen sind die DFAs doch bestimmt sehr viel größer?!?

Die Potenzmengenkonstruktion (1/2)

Sei $A = (\Sigma, Q, \delta, q_0, F)$ der gegebene NFA.

Idee: Wir konstruieren einen DFA $A' = (\Sigma, Q', \delta', q_0', F')$, der in seinem aktuellen Zustand $q' \in Q'$

> *die **Menge** aller Zustände abspeichert, in denen der Automat A in der aktuellen Situation sein **könnte**.*

Wir definieren die Komponenten von A' daher wie folgt:

- Eingabealphabet Σ,
- Zustandsmenge $Q' := \mathcal{P}(Q)$,
- Startzustand $q_0' := \{q_0\}$,
- Endzustandsmenge $F' := \{X \in Q' \ : \ X \cap F \neq \emptyset\}$,
- Übergangsfunktion $\delta' : Q' \times \Sigma \to Q'$, wobei für alle $X \in Q' = \mathcal{P}(Q)$ und alle $a \in \Sigma$ gilt:

$$\delta'(X, a) := \bigcup_{q \in X} \delta(q, a).$$

Die Potenzmengenkonstruktion (2/2)

Und wie zeigt man, dass A und A' dieselbe Sprache akzeptieren?

Zeige, dass

$$\delta'(\{q_0\}, w) \;=\; \delta(q_0, w)$$

für jedes Wort $w \in \Sigma^*$ gilt.

Und wie, bitte schön, sollen wir das zeigen?

- Wir haben die erweiterten Übergangsfunktionen δ und δ' rekursiv definiert.
- Dann sollten wir wohl eine vollständige Induktion nach $n = |w|$ ausführen!

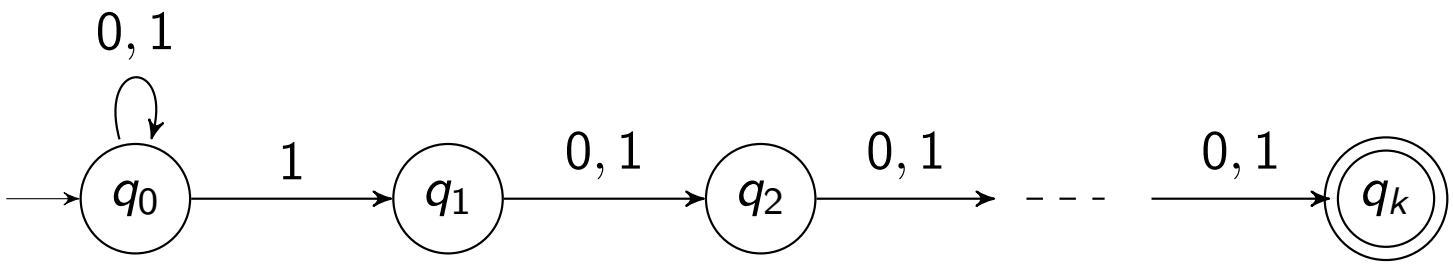
NFAs vs. DFAs: Die Zustandszahl

Nichtdeterminismus kann Zustände einsparen (1/2)

Für $k \in \mathbb{N}$ sei

$$L_k \;:=\; \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^{k-1}$$

Ein NFA, der L_k akzeptiert, kann die Position des k-letzten Buchstabens raten und dann verifizieren, dass er richtig geraten hat:



Somit gilt:

Es gibt einen NFA mit $k+1$ Zuständen, der die Sprache L_k akzeptiert.

Frage: Wie viele Zustände benötigt ein DFA, der L_k akzeptiert?

Antwort: Index(L_k) — Wie groß ist Index(L_k)?

Nichtdeterminismus kann Zustände einsparen (2/2)

Berechnung von Index(L_k) für $L_k = \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^{k-1}$:

- $u, v \in \{0,1\}^k$ seien beliebige, verschiedene Worte der Länge k.
- Dann gibt es eine Position i mit $u_i \neq v_i$.
  Ohne Beschränkung der Allgemeinheit gelte

$$u_i = 0 \quad \text{und} \quad v_i = 1.$$

$$\begin{aligned} \text{Es ist} \quad v\,0^{i-1} &= v_1 \cdots v_{i-1}\,1\,v_{i+1} \cdots v_k\,0^{i-1} \in L_k\,, \\ \text{aber} \quad u\,0^{i-1} &= u_1 \cdots u_{i-1}\,0\,u_{i+1} \cdots u_k\,0^{i-1} \notin L_k. \end{aligned}$$

- Die Worte u und v sind also nicht Myhill-Nerode äquivalent.
- Somit gibt es mindestens 2^k paarweise nicht Myhill-Nerode äquivalente Worte und Index(L_k) $\geqslant 2^k$ folgt.

Folgerung: Jeder DFA für L_k hat **mindestens** 2^k Zustände, aber es gibt NFAs für L_k mit **nur** $k+1$ Zuständen.

Können auch NFAs effizient minimiert werden?

- Wir werden später sehen, dass die Frage

$$L(N) \stackrel{?}{=} \Sigma^*$$

  extrem schwer zu beantworten ist.

- Die Frage wäre einfach zu beantworten, wenn wir NFAs effizient minimieren könnten.

Die Minimierung von NFAs ist knüppelhart.

Untere Schranken für die Größe von NFAs

Index $\Longrightarrow$ untere Schranken für NFAs

Frage:

Wie kann man für eine gegebene Sprache L zeigen, dass jeder **NFA**, der L erkennt, mindestens k Zustände hat?

Antwort:

Wenn man Glück hat, gilt $\text{Index}(L) \geqslant 2^k$, denn dann hat jeder DFA für L mind. 2^k Zustände:

- Aus einem NFA für L mit $k' < k$ Zuständen könnte man mit der Potenzmengenkonstruktion einen DFA mit $2^{k'} < 2^k$ Zuständen bauen.

Wir brauchen bessere Methoden!

Die Fooling Set Methode

Sei Σ ein endliches Alphabet und $L \subseteq \Sigma^*$ eine Sprache über Σ.
Für Worte $u_1, \ldots, u_k, v_1, \ldots, v_k \in \Sigma^*$ heißt die Menge

$$\{ (u_i, v_i) \ : \ 1 \leqslant i \leqslant k \}$$

ein **Fooling Set** für L, falls

(1) für alle $i \in \{1, \ldots, k\}$ gilt: $u_i v_i \in L$, und

(2) für alle $i, j \in \{1, \ldots, k\}$ mit $i \neq j$ gilt: $u_i v_j \notin L$ oder $u_j v_i \notin L$.

Fooling(L) ist die maximale Größe eines Fooling-Sets für L.

Übungsaufgabe:
Jeder NFA für L muss mindestens Fooling(L) Zustände besitzen.

Was ist der Zusammenhang zwischen Index(L) und Fooling(L)?

(a) Wie groß sind NFAs für $L = \{ uv \ : \ u, v \in \{0,1\}^k, u \neq v \}$?

(b) Und für $L = \{ uv \ : \ u, v \in \{0,1\}^k, u = v \}$?

NFAs mit Epsilon-Übergängen

NFAs mit ϵ-Übergängen

Ein **NFA mit ϵ-Übergängen** (kurz: ϵ-NFA) ist ein "verallgemeinerter NFA" $(Q, \Sigma, \delta, q_0, F)$, dessen Programm eine Funktion der folgenden Form ist:

$$\delta \colon Q \times \Big(\Sigma \cup \{\epsilon\}\Big) \to \mathcal{P}(Q)$$

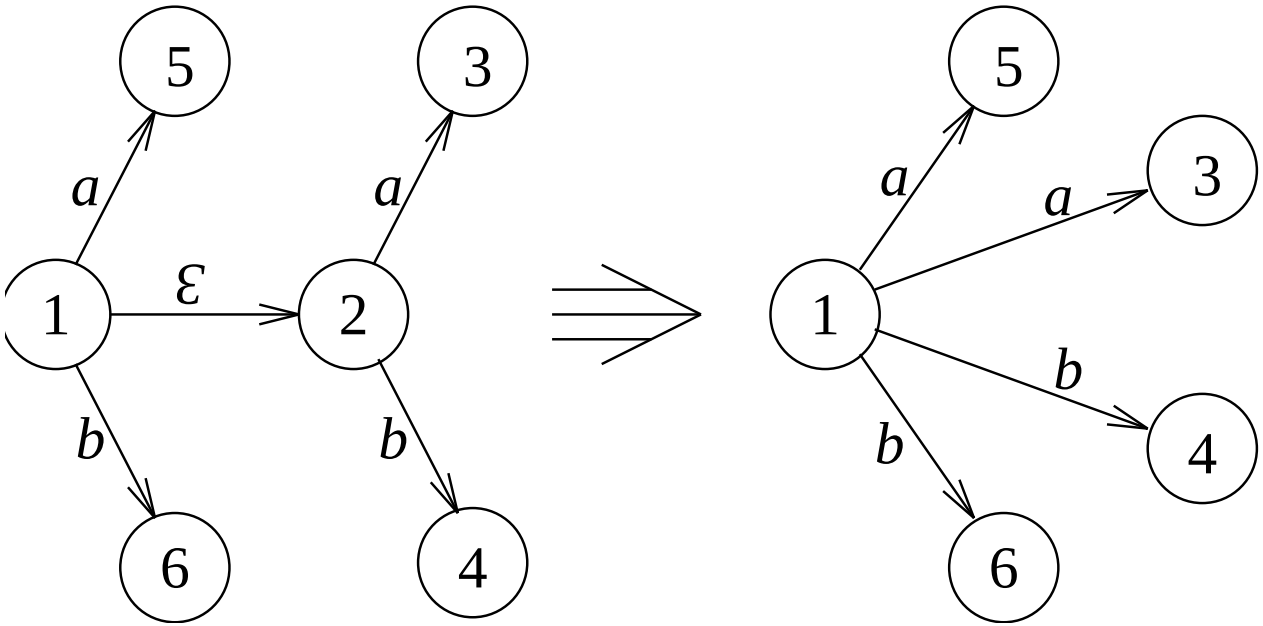
Der Automat darf also Zustandsübergänge ausführen, ohne Buchstaben zu lesen.

Frage: Können ϵ-NFAs mehr Sprachen akzeptieren als NFAs und DFAs?

Antwort: Nein!

Das Entfernen von ϵ-Übergängen: Idee

Entferne ϵ-Übergänge nach dem folgenden Schema



Die Zustandsmenge ist unverändert. Aber die Anzahl der neuen Übergänge kann quadratisch anwachsen.

Abschlusseigenschaften regulärer Sprachen

Abschlusseigenschaften regulärer Sprachen (1/6)

$L, L_1, L_2 \subseteq \Sigma^*$ seien reguläre Sprachen. Dann sind auch die folgenden Sprachen regulär:

(a) $\overline{L} := \Sigma^* \setminus L$.

(b) $L_1 \cup L_2$ und $L_1 \cap L_2$.

(c) $L_1 \cdot L_2$.

(d) L^*.

Die Klasse aller regulären Sprachen ist also abgeschlossen unter Booleschen Operationen sowie unter Konkatenation und Stern-Bildung.

Beweis:

(a) Abschluss unter Komplementbildung:

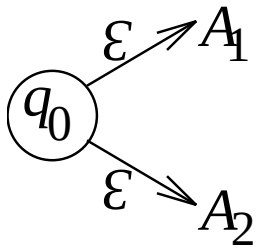
Vertausche akzeptierende und verwerfende Zustände.

Wenn der DFA $A = (Q, \Sigma, \delta, q_0, F)$ die Sprache L akzeptiert, dann akzeptiert der Automat $B = (Q, \Sigma, \delta, q_0, Q \setminus F)$ die Komplementsprache $\overline{L}$.

(b) Abschluss unter Vereinigung:

Verwende ϵ-Übergänge.

Wenn die DFAs $A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ und $A_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ die Sprachen L_1 und L_2 akzeptieren, dann akzeptiert der ϵ-NFA



$A = (\{q_0\} \dot\cup Q_1 \dot\cup Q_2, \Sigma, \delta, q_0, F \cup F')$ die Sprache $L_1 \cup L_2$.

Abschluss unter Durchschnitt: folgt, da $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$.

Alternative: Verwende den Produktautomaten $A = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q := Q_1 \times Q_2$
- $q_0 := (q_1, q_2)$
- $\delta\big((p_1, p_2), a\big) := \big(\delta_1(p_1, a), \delta_2(p_2, a)\big)$, für alle $(p_1, p_2) \in Q$, $a \in \Sigma$
- für Durchschnitt: $F := F_1 \times F_2$
  für Vereinigung: $F := (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Abschlusseigenschaften regulärer Sprachen (3/6)

Beachte: Reguläre Sprachen sind natürlich nur unter **endlicher** Vereinigung und **endlicher** Durchschnittsbildung abgeschlossen.

Beispiel:

- Für jedes $n \in \mathbb{N}$ besteht die Sprache $L_n := \{a^n b^n\}$ aus nur einem Wort und ist damit regulär. Aber

$$\bigcup_{n \in \mathbb{N}} L_n$$

ist die Sprache $\{a^n b^n : n \in \mathbb{N}\}$, von der wir wissen, dass sie nicht regulär ist.

- Für jedes $n \in \mathbb{N}$ ist die Sprache $L'_n := \{a, b\}^* \setminus L_n$ regulär (da L_n regulär ist und da die regulären Sprachen unter Komplementbildung abgeschlossen sind).
Aber

$$\bigcap_{n \in \mathbb{N}} L'_n$$

ist gerade die Sprache $\{a, b\}^* \setminus \{a^n b^n : n \in \mathbb{N}\}$, also das Komplement der Sprache $\{a^n b^n : n \in \mathbb{N}\}$ und daher nicht regulär.

Abschlusseigenschaften regulärer Sprachen (4/6)

(c) Abschluss unter Konkatenation:

- Verbinde jeden akzeptierenden Zustand von A_1 durch einen ϵ-Übergang mit dem Startzustand von A_2.
- Die akzeptierenden Zustände von A_2 sind die akzeptierenden Zustände des neuen Automaten.

(d) Abschluss unter Stern-Bildung:

- Führe einen neuen, akzeptierenden Startzustand q_0' ein.
- Verbinde alle akzeptierenden Zustände von A durch ϵ-Übergänge mit q_0' und
- verbinde q_0' durch einem ϵ-Übergang mit dem alten Startzustand q_0.
- q_0' ist der einzige akzeptierende Zustand des neuen Automaten.

- Der **Quotient**: Seien $R, S \subseteq \Sigma^*$ Sprachen. Wenn R regulär ist, dann auch

$$R/S := \{ u \in \Sigma^* \ : \ \text{es gibt } v \in S, \text{ so dass } uv \in R \}.$$

  Achtung: Es wird nicht gefordert, dass die Sprache S regulär ist!

- Ein **Homomorphismus** $h : \Sigma^* \to \Delta^*$ weist jedem Buchstaben $a \in \Sigma$ ein Wort $h(a) \in \Delta^*$ zu. Für ein Wort $x = a_1 \cdots a_n \in \Sigma^n$ ist $h(x) := h(a_1) \cdots h(a_n)$.
  Wenn L eine reguläre Sprache ist, dann sind auch

$$h(L) := \{ h(x) \ : \ x \in L \} \text{ und } h^{-1}(L) := \{ x \in \Sigma^* \ : \ h(x) \in L \}$$

  regulär.

- Eine **Substitution** ist eine Funktion $s : \Sigma^* \to \mathcal{P}(\Delta^*)$, die jedem Buchstaben $a \in \Sigma$ eine Sprache $s(a) \subseteq \Delta^*$ zuordnet. Es muss gelten

$$s(\epsilon) = \{\epsilon\} \text{ und } s(x \cdot y) = s(x) \cdot s(y) \text{ für alle } x, y \in \Sigma^*.$$

  Wenn L regulär ist, dann ist auch $s(L)$ regulär.

Abschlusseigenschaften regulärer Sprachen (6/6)

Wenn L regulär ist, dann auch

- **Präfix(L)** $= \{ u \ : \ \text{es gibt } v \text{ mit } uv \in L \ \}$,
- $\frac{1}{2}$**L** $= \{ u \ : \ \text{es gibt } v \text{ mit } uv \in L \text{ und } |u| = |v| \}$,
- **Suffix(L)**$= \{ v \ : \ \text{es gibt } u \text{ mit } uv \in L \ \}$,
- **Teilwort(L)** $= \{ w \ : \ \text{es gibt } u, v \text{ mit } uwv \in L \ \}$,
- **Reverse(L)** $= \{ a_n \cdots a_2 \cdot a_1 \ : \ a_1, \ldots, a_n \in \Sigma, \ a_1 \cdot a_2 \cdots a_n \in L \}$.

Frage: Und wie zeigt man das?

Antwort: Arbeite mit NFAs.

Entscheidungsprobleme

Welche Fragen sind effizient beantwortbar? (1/2)

A sei ein DFA oder NFA.

Ist $L(A) \neq \emptyset$?

- Sei q_0 der Anfangszustand von A und sei F die Menge der akzeptierenden Zustände.
- $L(A)$ ist genau dann nicht-leer, wenn es einen Weg von q_0 zu einem Zustand in F gibt.
  - Wende Tiefensuche auf q_0 an.
  - $L(A)$ ist nicht-leer, wenn Tiefensuche einen akzeptierenden Zustand findet.

Die Laufzeit ist proportional zur Anzahl der Kanten im Zustandsdiagramm, also proportional zu $O(|\Sigma| \cdot |Q|)$ für DFAs, bzw. zu $O(|\Sigma| \cdot |Q|^2)$ für NFAs.

Welche Fragen sind effizient beantwortbar? (2/2)

D, D_1, D_2 seien deterministische Automaten

Die folgenden Fragen lassen sich effizient beantworten (in Zeit $O(|\Sigma| \cdot |Q|)$ bzw. $O(|\Sigma| \cdot |Q_1| \cdot |Q_2|)$):

(a) Ist $L(D) = \Sigma^*$? (Universalität)

(b) Ist $L(D_1) = L(D_2)$? (Äquivalenz)

(c) Ist $L(D_1) \subseteq L(D_2)$? ("Containment")

(d) Ist $L(D)$ endlich? (Endlichkeit)

Beweis: Übungsaufgabe! Idee: Nutze jeweils aus, dass die Frage

$$\text{Ist } L(D) \neq \emptyset?$$

effizient beantwortet werden kann.

Die Fragen (a)–(c) sind für NFAs extrem schwierig: Später weisen wir

PSPACE-Vollständigkeit

nach: Diese Fragen sind also mindestens so schwierig wie NP-vollständige Probleme.

Das Wortproblem für NFAs

Für einen NFA A und ein Wort w entscheide, ob A die Eingabe w akzeptiert.

Wie schwierig ist das Wortproblem?

- Nicht ganz so einfach wie für DFAs, aber die Menge $\delta(q_0, w)$ der von q_0 aus erreichbaren Zustände lässt sich effizient berechnen: Zeit $O(|w| \cdot |Q|^2)$ reicht aus!
- Akzeptiere genau dann, wenn $\delta(q_0, w) \cap F \neq \emptyset$.

Sehr schwierige Entscheidungsfragen für DFAs und NFAs

(1) Das **Interpolationsproblem** für DFAs:
Für Teilmengen $P, N \subseteq \Sigma^*$ und einen Schwellenwert $k \geqslant 1$,

gibt es einen DFA mit höchstens k Zuständen, der alle Worte in P akzeptiert und alle Worte in N verwirft?

Dieses Problem ist NP-vollständig. (Hier ohne Beweis)

(2) Das **Minimierungsproblem** für NFAs:

- Gegeben zwei NFAs A und B, entscheide, ob B ein zu A äquivalenter NFA mit minimaler Zustandszahl ist.
  Dieses Problem ist PSPACE-vollständig. (Beweis später)
- Es ist sogar schwierig, die minimale Zustandszahl *approximativ* zu bestimmen.
- Was ist da los? Bereits die Frage „$L(N) \neq \Sigma^*$?“ ist extrem schwer (nämlich PSPACE-vollständig) und gehört wahrscheinlich nicht einmal zur Klasse NP.

Worte in $\Sigma^* \setminus L(N)$ sind unter Umständen sehr lang.

Reguläre Ausdrücke

Reguläre „Muster“

Sei Σ ein Alphabet.

1. Die Menge aller Worte über Σ haben wir mit

$$\Sigma^*$$

   beschrieben.
2. Sei u ein Wort über Σ. Dann wird die Menge aller Worte über Σ, die u als Teilwort besitzen, beschrieben durch

$$\Sigma^* \cdot u \cdot \Sigma^*.$$

3. Die Menge aller Worte über Σ, deren vorletzter Buchstabe ein a ist, wird beschrieben durch:

$$\Sigma^* \cdot a \cdot \Sigma$$

Reguläre Ausdrücke

Die Menge der regulären Ausdrücke über einem endlichen Alphabet Σ wird rekursiv wie folgt definiert:

Basisregel: Die Ausdrücke $\emptyset$, ϵ und a für $a \in \Sigma$ sind regulär.

Die Ausdrücke beschreiben die leere Sprache ($L(\emptyset) = \emptyset$),
die Sprache des leeren Wortes ($L(\varepsilon) = \{\varepsilon\}$) und
die Sprache des einbuchstabigen Wortes a ($L(a) = \{a\}$).

Rekursive Regeln: Sind R und S reguläre Ausdrücke, dann auch $(R \mid S)$, $(R \cdot S)$ und R^*.

$\mid$ beschreibt die Vereinigung und wird manchmal auch mit $+$ bezeichnet ($L((R \mid S)) = L(R) \cup L(S)$),
$\cdot$ die Konkatenation und wird manchmal auch mit $\circ$ bezeichnet ($L((R \cdot S)) = L(R) \cdot L(S)$), und
$*$ beschreibt die Kleene-Hülle ($L(R^*) = L(R)^*$).

Abkürzende Schreibweise

Zur vereinfachten Schreibweise und besseren Lesbarkeit vereinbaren wir folgende Konventionen:

- Den Punkt bei der Konkatenation darf man weglassen.
  (Schreibe (RS) statt $(R \cdot S)$)
- Bei Ketten gleichartiger Operationen darf man die Klammern weglassen.
  (Schreibe $(R_1|R_2|R_3)$ statt $((R_1|R_2)|R_3)$, und $(R_1R_2R_3)$ statt $((R_1R_2)R_3)$)
- Äußere Klammern, die einen regulären Ausdruck umschließen, dürfen weggelassen werden.
- Zur besseren Lesbarkeit dürfen zusätzliche Klammern eingefügt werden.
- Bei fehlenden Klammern gelten folgende "Präzedenzregeln":
  "$*$" bindet stärker als "$\cdot$"; "$\cdot$" bindet stärker als "$|$".

Beispiel:

$a \,|\, bbc^*$ ist eine verkürzte Schreibweise für den regulären Ausdruck $(a \,|\, ((b \cdot b) \cdot c^*))$.
Die von ihm beschriebene Sprache ist $L(a \,|\, bbc^*) = \{a\} \cup (\{bb\} \cdot \{c\}^*)$.

Reguläre Ausdrücke definieren reguläre Sprachen

Sei Σ ein endliches Alphabet und R ein **regulärer Ausdruck** über Σ. Dann ist die Sprache $L(R)$ **regulär**.

Beweis: Per Induktion über den Aufbau der regulären Ausdrücke.

Induktionsanfang: Betrachte die gemäß Basisregeln gebildeten regulären Ausdrücke.

$\emptyset$, ϵ, und a, für $a \in \Sigma$ sind reguläre Ausdrücke, die die regulären Sprachen $L(\emptyset) = \emptyset$, $L(\epsilon) = \{\varepsilon\}$ und $L(a) = \{a\}$ beschreiben.

Induktionsschritt: Betrachte die gemäß den rekursiven Regeln gebildeten regulären Ausdrücke.

Seien R und S reguläre Ausdrücke, die gemäß Induktionsannahme reguläre Sprachen $L(R)$ und $L(S)$ beschreiben.
Wir wissen bereits, dass die Klasse der regulären Sprachen abgeschlossen ist unter Vereinigung, Konkatenation und Kleene-Stern.

$\Longrightarrow$ Auch die folgenden Sprachen sind regulär: $L(R) \cup L(S) = L((R \mid S))$, $L(R) \cdot L(S) = L((R \cdot S))$, $L(R)^* = L(R^*)$.

$\Longrightarrow$ $(R \mid S)$, $(R \cdot S)$ und R^* beschreiben reguläre Sprachen. $\square$

Zwischenstand

- Jeder reguläre Ausdruck definiert also eine reguläre Sprache.
- Und umgekehrt, gibt es zu jeder regulären Sprache auch einen regulären Ausdruck?

Reguläre Sprachen haben reguläre Ausdrücke (1/3)

Sei Σ ein endliches Alphabet. Zu jeder regulären Sprache $L \subseteq \Sigma^*$ gibt es einen regulären Ausdruck R über Σ mit $L = L(R)$.

Beweis:

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DFA, der L akzeptiert. Es gelte $Q = \{1, \ldots, n\}$.

Idee: Benutze dynamische Programmierung, um reguläre Ausdrücke $R^k_{p,q}$ zu konstruieren, die die Mengen $L^k_{p,q}$ beschreiben, wobei

$$L^k_{p,q} := \Big\{ w \in \Sigma^* \;:\; \delta(p,w) = q, \text{ und alle Zwischenzustände liegen in der Menge } \{1,\ldots,k\} \Big\},$$

für alle $p, q \in Q$ und alle $k \in \{0, 1, \ldots, n\}$ gilt.

Es ist: $L = \bigcup_{q \in F} L^n_{q_0,q}$. Für $F = \{q_1, \ldots, q_s\}$ wird L daher durch den regulären Ausdruck $(R^n_{q_0,q_1} \mid \cdots \mid R^n_{q_0,q_s})$ beschrieben.

Reguläre Sprachen haben reguläre Ausdrücke (2/3)

Rekursiv (für $k \in \{0, 1, 2, \ldots, n\}$) konstruiere reguläre Ausdrücke $L^k_{p,q}$ für alle $p, q \in Q$.

Reguläre Ausdrücke für $L^0_{p,q}$, für alle $p, q \in Q$:

- $L^0_{p,q}$ ist die Menge aller Worte, die Zustand q von Zustand p aus **ohne** Zwischenzustände erreichen. Also:
  - Falls $p \neq q$, so ist $L^0_{p,q} = \{a \in \Sigma : \delta(p, a) = q\}$
  - Falls $p = q$, so ist $L^0_{p,q} = \{\varepsilon\} \cup \{a \in \Sigma : \delta(p, a) = q\}$
- Also ist $L^0_{p,q}$ eine endliche Menge von Buchstaben bzw. dem leeren Wort. $L^0_{p,q}$ wird von einem regulären Ausdruck $R^0_{p,q}$ beschrieben (durch das Symbol $\emptyset$ oder durch Verknüpfen der Buchstaben bzw. des leeren Wortes mit dem "|"-Operator).

Für alle $p, q, r, s \in Q$ und für jedes $k \geqslant 1$ konstruiere reguläre Ausdrücke für die Mengen $L^k_{p,q}$, wobei wir annehmen, dass reguläre Ausdrücke für $L^{k-1}_{r,s}$ schon bekannt sind.

Reguläre Sprachen haben reguläre Ausdrücke (3/3)

Reguläre Ausdrücke für $L_{p,q}^k$:

- $L_{p,q}^k$ ist die Menge aller Worte w, die Zustand q von Zustand p aus mit Zwischenzuständen aus der Menge $\{1, \ldots, k\}$ erreichen.
  - Entweder benötigt ein Wort w nur Zwischenzustände in der Menge $\{1, \ldots, k-1\}$, d.h. es liegt in $L_{p,q}^{k-1}$,
  - oder Zustand k ist Zwischenzustand.
    - ⋆ Dann führt die Berechnung von p nach k,
    - ⋆ „pendelt“ zwischen k und k
    - ⋆ und erreicht dann q von Zustand k aus.
  - Wie oft kann Zustand k angenommen werden? Keine Ahnung! Macht aber nichts, denn

$$L_{p,q}^k \quad = \quad L_{p,q}^{k-1} \quad \cup \quad L_{p,k}^{k-1} \left(L_{k,k}^{k-1}\right)^* \quad L_{k,q}^{k-1}.$$

- Daher wird $L_{p,q}^k$ durch den folgenden regulären Ausdruck beschrieben:

$$R_{p,q}^k \quad := \quad R_{p,q}^{k-1} \quad | \quad R_{p,k}^{k-1} \cdot \left(R_{k,k}^{k-1}\right)^* \cdot R_{k,q}^{k-1}.$$

Der Satz von Kleene

Wir haben somit den Satz von Kleene bewiesen:

Sei Σ ein endliches Alphabet. Dann gilt

Eine Sprache $L \subseteq \Sigma^*$ ist regulär $\Longleftrightarrow$
Es gibt einen regulären Ausdruck R mit $L = L(R)$.

Fragen:

(a) Wie groß ist der zu einem regulären Ausdruck äquivalente NFA im schlimmsten Fall?

Übung!

(b) Wie groß ist der zu einem regulären Ausdruck äquivalente DFA im schlimmsten Fall?

Übung!

(c) Wie groß ist der zu einem DFA A äquivalente reguläre Ausdruck R im schlimmsten Fall, wenn wir das obige Verfahren anwenden?

Übung!

Reguläre Ausdrücke für Komplement und Durchschnitt (1/2)

- Wir wissen: Die Klasse aller regulären Sprachen ist abgeschlossen unter Komplement- und Durchschnittbildung.
- Obwohl die regulären Ausdrücke keine expliziten Operatoren für Komplement- und Durchschnittbildung enthalten, gibt es daher für alle regulären Ausdrücke R und S einen regulären Ausdruck
  - $\tilde{R}$, der die Sprache $\overline{L(R)}$ beschreibt, und
  - T, der die Sprache $L(R) \cap L(S)$ beschreibt.
- **Frage:** Wie können wir bei gegebenen R und S solche regulären Ausdrücke $\tilde{R}$ und T konstruieren?
- **Antwort:**

$$
\begin{array}{lll}
R,\ S & \rightsquigarrow & \text{NFAs} \quad A_{L(R)},\ A_{L(S)} \\
& \rightsquigarrow & \text{DFAs} \quad B_{L(R)},\ B_{L(S)} \\
& \rightsquigarrow & \text{DFAs} \quad B_{\overline{L(R)}},\ B_{L(R)\cap L(S)} \\
& \rightsquigarrow & \text{reguläre Ausdrücke} \quad \tilde{R},\ T.
\end{array}
$$

- **Laufzeit unserer Verfahren:** Für $\tilde{R}$: $2^{2^{O(|R|)}}$. Für T: $2^{2^{O(|R|+|S|)}}$.
- **Frage:** Geht das effizienter?

Reguläre Ausdrücke für Komplement und Durchschnitt (2/2)

Wir kennen Verfahren, die bei gegebenen regulären Ausdrücken R und S in Zeit

- $2^{2^{O(|R|)}}$ einen regulären Ausdruck $\tilde{R}$ für die Sprache $\overline{L(R)}$
- $2^{2^{O(|R|+|S|)}}$ einen regulären Ausdruck T für die Sprache $L(R) \cap L(S)$

liefern.

Frage: Geht das effizienter?

Antwort von Gelade und Neven (2008): (hier ohne Beweis)

- Für $\tilde{R}$ ist unser Verfahren im Wesentlichen optimal:

  Es gibt reguläre Ausdrücke R_n der Länge $O(n)$, so dass die kürzesten regulären Ausdrücke, die die Sprache $\overline{L(R_n)}$ beschreiben, mindestens die Länge 2^{2^n} haben.

- Für T gibt es ein (relativ naheliegendes) Verfahren, das mit Laufzeit $2^{O(|R|\cdot|S|)}$ auskommt. Dieses Verfahren ist im Wesentlichen optimal:

  Es gibt reguläre Ausdrücke R_n, S_n der Länge $O(n^2)$, so dass die kürzesten regulären Ausdrücke für die Sprache $L(R_n) \cap L(S_n)$ mindestens die Länge 2^n haben.

Reguläre Grammatiken

Grammatiken

Programmiersprachen lassen sich am besten als die Sprache aller syntaktisch korrekten Programme auffassen.

Definiere die Syntax durch eine Grammatik.

Wie erzeugt man arithmetische Ausdrücke mit ganzzahligen Konstanten?

Wir arbeiten mit den Variablen

- A: erzeuge einen arithmetischen Ausdruck,
- I: erzeuge eine natürliche Zahl und
- Z: erzeuge eine Ziffer.

$$\begin{aligned} A &\rightarrow A + A \mid A - A \mid A * A \mid (A) \mid I \mid +I \mid -I \\ I &\rightarrow Z\,I \mid Z \\ Z &\rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

Wir haben Produktionen (oder Ersetzungsregeln) benutzt.

Die Komponenten einer Grammatik

Eine Grammatik $G = (\Sigma, V, S, P)$ besteht aus:

- einem endlichen Alphabet Σ,
- einer endlichen Menge V von **Variablen** (oder Nichtterminalen) mit $\Sigma \cap V = \emptyset$,
- dem **Startsymbol** $S \in V$ und
- einer endlichen Menge P von **Produktionen** der Form $u \to v$ mit

$$u \in (\Sigma \cup V)^* \, V \, (\Sigma \cup V)^* \quad \text{und} \quad v \in (\Sigma \cup V)^*$$

Beginne mit dem Startsymbol S und wende dann Produktionen an:

Eine Produktion $\mathbf{u} \to \mathbf{v}$ ersetzt ein Vorkommen von $\mathbf{u}$ durch ein Vorkommen von $\mathbf{v}$.

Die von einer Grammatik erzeugte Sprache

- Für eine Produktion $u \to v$ und ein Wort $\mathbf{w_1 = xuy}$ wird das Wort $\mathbf{w_2 = xvy}$ abgeleitet. Wir schreiben

$$xuy \Rightarrow xvy.$$

- Für Worte $r, s \in (\Sigma \cup V)^*$ schreiben wir

$$r \overset{*}{\Rightarrow} s \quad :\Longleftrightarrow \quad \text{Es gibt Worte } w_1 = r, w_2, \ldots, w_k = s, \text{ für } k \geqslant 1, \text{ so dass } w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_k.$$

s kann in null, einem oder mehreren Schritten aus r abgeleitet werden.

- $L(G) = \left\{ w \in \Sigma^* \ : \ S \overset{*}{\Rightarrow} w \right\}$ ist die von der Grammatik G erzeugte Sprache.

Reguläre Grammatiken

Frage: Welcher Typ von Grammatik erzeugt reguläre Sprachen?

Eine Grammatik $G = (\Sigma, V, S, P)$, bei der alle Produktionen von der Form

$$\begin{array}{lcll} X & \to & \epsilon & \text{für } X \in V \quad \text{oder} \\ X & \to & aY & \text{für } X, Y \in V \text{ und } a \in \Sigma \end{array}$$

sind, heißt (rechts-)regulär.

Die zentralen Fragen:

- Wird jede reguläre Sprache von einer regulären Grammatik erzeugt?
- Ist die von einer regulären Grammatik erzeugte Sprache regulär?

Reguläre Grammatiken: Ein Beispiel

Eine Grammatik für die Sprache aller Worte über $\Sigma = \{a, b\}$, die das Wort *aba* als Teilwort besitzen:

- Die Variablen:
  - S: ist das Startsymbol; es soll ein *beliebiges* Präfix erzeugen,
  - V_a: bedeutet, dass wir gerade ein *a* erzeugt haben und dass als nächstes ein *b* erzeugt werden soll,
  - V_{ab}: bedeutet, dass wir gerade *ab* erzeugt haben und dass als nächstes ein *a* erzeugt werden soll,
  - T: soll ein *beliebiges* Suffix erzeugen.

- Die Produktionen:

$$\begin{aligned} S &\to aS \mid bS \mid aV_a \\ V_a &\to bV_{ab} \\ V_{ab} &\to aT \\ T &\to aT \mid bT \mid \epsilon \end{aligned}$$

Reguläre Sprachen besitzen reguläre Grammatiken

Sei $A = (Q, \Sigma, \delta, q_0, F)$ ein NFA, der die Sprache $L := L(A)$ akzeptiert.

Auf Eingabe $w = a_1 \cdots a_n$ führt A Zustandsübergänge

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n$$

durch.

Sei $G = (\Sigma, V, S, P)$ die reguläre Grammatik mit

- Variablenmenge $V := Q$
- Startsymbol $S := q_0$
- Produktionen

$$p \to aq \quad \text{für alle } p, q, a \text{ mit } \delta(p, a) \ni q$$
$$p \to \epsilon \quad \text{für alle } p \in F.$$

Behauptung: Diese Grammatik G erzeugt die Sprache $L = L(A)$.

Reguläre Grammatiken erzeugen reguläre Sprachen

Sei $G = (\Sigma, V, S, P)$ eine reguläre Grammatik, die die Sprache $L := L(G)$ erzeugt.

Eine Ableitung von G hat die Form

$$S \Rightarrow a_1 V_1 \Rightarrow a_1 a_2 V_2 \Rightarrow \cdots \Rightarrow a_1 \cdots a_n V_n \Rightarrow a_1 \cdots a_n.$$

Sei $A = (Q, \Sigma, \delta, q_0, F)$ der NFA mit

- Zustandsmenge $Q := V$,
- Startzustand $q_0 := S$,
- Überführungsfunktion $\delta : Q \times A \to \mathcal{P}(Q)$, wobei für alle $X \in Q$ und $a \in \Sigma$ gilt:

$$\delta(X, a) := \{Y \ : \ (X \to aY) \ \in P\}.$$

- Endzustandsmenge $F := \{X \in V \ : \ (X \to \epsilon) \ \in \ P\}$

Behauptung: Dieser NFA A akzeptiert genau die Sprache $L = L(G)$.

Probabilistische Automaten

Würfelnde Automaten

Ein **probabilistische endlicher Automat (PFA)**

$$W = (\Sigma, Q, \delta, q_0, F)$$

darf würfeln: Das Programm

$$\delta : Q \times \Sigma \times Q \to [0, 1]$$

weist allen Zuständen $p, r \in Q$ und jedem Buchstaben $a \in \Sigma$ die

Wahrscheinlichkeit $\delta(p, a, r)$

zu, dass r als Nachfolgezustand von p gewählt wird, wenn a gelesen wird.

Die Übergangsmatrix

Für jeden Buchstaben $a \in \Sigma$ ist P_a die Übergangsmatrix, wobei

$$P_a[p, r] := \delta(p, a, r)$$

die Wahrscheinlichkeit ist, dass W von p in r wechselt, wenn a gelesen wird.

P_a ist eine **stochastische Matrix**:

Für alle Zustände $p, r \in Q$ ist $P_a[p, r] \geqslant 0$ und es ist $\sum_{r \in Q} P_a[p, r] = 1$.

Die erweiterte Übergangsfunktion

Sei $W = (\Sigma, Q, \delta, q_0, F)$ ein probabilistischer endlicher Automat (PFA).

(a) Für ein Wort $u \in \Sigma^*$, einen Anfangszustand p und einen Zustande $r \in Q$ ist

$$\delta(p, u, r)$$

die Wahrscheinlichkeit, dass sich W nach dem Lesen des Wortes u im Zustand $r \in Q$ befindet. Wir geben die folgende rekursive Definition an:

$$\delta(p, ua, r) := \sum_{s \in Q} \delta(p, u, s) \cdot \delta(s, a, r).$$

(b) Die Akzeptanzwahrscheinlichkeit $p_W(u)$ des Wortes w wird definiert durch

$$p_W(u) := \mathrm{pr}[\, W \text{ akzeptiert } u \,] := \sum_{r \in F} \delta(q_0, u, r).$$

(c) Die reelle Zahl $\lambda \in [0, 1]$ sei gegeben. Die Sprache $L_\lambda(W)$ besteht aus allen Worten $u \in \Sigma^*$, die mit Wahrscheinlichkeit größer als λ akzeptiert werden:

$$L_\lambda(W) := \{\, u \in \Sigma^* \;:\; p_W(u) > \lambda \}.$$

Erweiterte Übergangsfunktion und Matrizenprodukte

Wenn $u = a_1 \cdots a_n$, dann folgt mit vollständiger Induktion:

$$\delta(p, u, r) = \left(P_{a_1} \cdots P_{a_n}\right)[p, r].$$

Frage: Ist die Sprache $L_\lambda(W)$ für jeden probabilistischen endlichen Automaten und für jedes λ eine reguläre Sprache?

Akzeptieren PFAs nur reguläre Sprachen?

Der PFA $W = (\Sigma, Q, \delta, q_0, F)$ mit

1. $\Sigma = \{0, 1\}$,
2. Zustandsmenge $Q = \{0, 1\}$,
3. Anfangszustand $q_0 = 0$
4. dem akzeptierenden Zustand 1 und
5. dem Programm δ, definiert durch die beiden Übergangsmatrizen

$$P_0 = \begin{bmatrix} 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \quad \text{und} \quad P_1 = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 1 \end{bmatrix}.$$

Für das Wort $u = a_1 \cdots a_n$ gilt

$$p_W(u) = \left(P_{a_1} \cdots P_{a_n}\right)[0, 1] = 0.a_n \cdots a_1.$$

Es gibt überabzählbar viele Sprachen $L_\lambda(W)$, aber nur abzählbar viele DFAs!

Schade?!

Es gibt einen PFA W und einen Schwellenwert λ, so dass $L_\lambda(W)$ nicht regulär ist.

Ist Mutter Natur zu tadeln?

- Wie sollen wir
$$u \overset{?}{\in} L_\lambda(W)$$
experimentell entscheiden, wenn wir nur wenige Berechnungen von W auf u beobachten dürfen?
- Wir sollten für eine reelle Zahl $\delta > 0$ fordern, dass alle Akzeptanzwahrscheinlichkeiten **außerhalb** eines Intervalls
$$(\lambda - \delta, \lambda + \delta)$$
liegen.

Akzeptanz mit Lücke

δ sei eine positive reelle Zahl. Ein PFA W akzeptiert die Sprache $L = L_\lambda(W)$ mit (**Schwellenwert** λ und) **Lücke** δ, falls

$$p_W(u) \notin (\lambda - \delta, \lambda + \delta)$$

für alle Worte $u \in \Sigma^*$ gilt.

- Jeder DFA A ist ein PFA, der die Sprache $L = L_1(A)$ mit Schwellenwert $1/2$ und der größtmöglichen Lücke $1/2$ akzeptiert.
- Können PFAs mit großer Lücke Zustände gegenüber DFAs „einsparen“?
  - Der Index von $EQ_k = \{ww : w \in \{0,1\}^k\}$ beträgt mindestens 2^k.
  - Es gibt einen PFA mit poly(k) Zuständen, der EQ_k mit Schwellenwert $\frac{1+1/k}{2}$ und Lücke $\frac{1-1/k}{2}$ akzeptiert.

Zeige den **Satz von Rabin:** Sei W ein PFA, der die Sprache $L = L_\lambda(W)$ mit Lücke $\delta > 0$ akzeptiert. Dann ist die Sprache L regulär.

Satz von Rabin (1/2)

$W = (\Sigma, Q, \delta, q_0, F)$ sei ein PFA.

Angenommen, $u_1, \ldots, u_N$ sind Vertreter verschiedener Myhill-Nerode Klassen: Für je zwei Worte $u_i \neq u_j$ gibt es ein Wort x mit z. B. $u_i x \in L$, aber $u_j x \notin L$. Also:

$$\begin{aligned} p_W(u_i x) &= \sum_{p \in Q, q \in F} \delta(q_0, u_i, p) \cdot \delta(p, x, q) \geqslant \lambda + \delta, \\ p_W(u_j x) &= \sum_{p \in Q, q \in F} \delta(q_0, u_j, p) \cdot \delta(p, x, q) \leqslant \lambda - \delta \end{aligned}$$

und insbesondere

$$\begin{aligned} 2 \cdot \delta = \lambda + \delta - (\lambda - \delta) &\leqslant p_W(u_i x) - p_W(u_j x) \\ &= \sum_{p \in Q, q \in F} \Big(\delta(q_0, u_i, p) - \delta(q_0, u_j, p) \Big) \cdot \delta(p, x, q) \\ &= \sum_{p \in Q} \Big(\big(\delta(q_0, u_i, p) - \delta(q_0, u_j, p) \big) \cdot \sum_{q \in F} \delta(p, x, q) \Big) \end{aligned}$$

Satz von Rabin (2/2)

Wir wissen: $2 \cdot \delta \leqslant \sum_{p \in Q} \Big((\delta(q_0, u_i, p) - \delta(q_0, u_j, p)) \cdot \sum_{q \in F} \delta(p, x, q) \Big)$.

(a) $0 \leqslant \sum_{q \in F} \delta(p, x, q) < |Q|$, denn $0 \leqslant \delta(p, x, q) \leqslant 1$ für alle $p, q \in Q$.

(b) Als Konsequenz folgt

$$2 \cdot \delta / |Q| \leqslant \sum_{p \in Q} \Big| \delta(q_0, u_i, p) - \delta(q_0, u_j, p) \Big|.$$

(c) Die Vektoren

$$\xi_i := (\delta(q_0, u_i, p) \ : \ p \in Q)$$

haben in der 1-Norm also einen Abstand von mindestens $2\delta/|Q|$ voneinander.

(d) Für jedes $\mu > 0$ gibt es aber nur endlich viele Vektoren im $|Q|$-dimensionalen Würfel $[0,1]^{|Q|}$ mit paarweisem Abstand von mindestens $\mu \Longrightarrow$

- Der Myhill-Nerode Index von L ist endlich und
- L ist regulär.

□

Zweiwege Automaten

Sprachen, die ohne Speicher akzeptiert werden können

Deterministische Zweiwege Automaten (**2DFAs**)

- können in jedem Schritt entscheiden, ob sie den vorangegangenen linken oder den folgenden rechten Buchstaben der Eingabe $w \in \Sigma^*$ besuchen möchten.
- Das linke Ende der Eingabe ist mit dem Buchstaben α und das rechte Ende mit dem Buchstaben β beschriftet, wobei $\alpha, \beta \notin \Sigma$ gelte.

Die formale Definition: Ein 2DFA $(\Sigma, Q, \delta, q_0, F)$ hat die folgenden Komponenten:

- Das Eingabealphabet Σ mit $\{\alpha, \beta\} \cap \Sigma = \emptyset$,
- eine endliche Menge Q von Zuständen,
- das Programm δ, das wir als Funktion

$$\delta \colon Q \times (\Sigma \cup \{\alpha, \beta\}) \to Q \times \{ \text{ links, rechts } \}$$

  auffassen: Die Eingabe w wird durch das Wort $\alpha w \beta$ repräsentiert.
- Den Anfangszustand $q_0 \in Q$
- sowie eine Menge $F \subseteq Q$ akzeptierender Zustände.

Die Sprache eines Zweiwege Automaten

Sei A ein 2DFA.

1. A beginnt im Zustand q_0 und liest den ersten Buchstabens α der Eingabe $\alpha w \beta$.
2. δ schreibt Zustandsänderungen, aber auch die Leserichtung vor.
   - A darf $\alpha w \beta$ nicht nach links verlassen.
   - A akzeptiert das Wort w genau dann, wenn A die transformierte Eingabe $\alpha w \beta$ nach rechts in einem Zustand aus F verlässt.

Die Sprache

$$L(A)$$

ist die Menge aller von A akzeptierten Worte $w \in \Sigma^*$.

2DFAs versus DFAs

(a) Können 2DFAs nicht-reguläre Sprachen akzeptieren?

- Nein, das würde Mutter Natur nicht zulassen.

(b) Gibt es reguläre Sprachen, die 2DFAs mit sehr viel weniger Zuständen akzeptieren als DFAs?
Ja, hier sind einige Beispiele:

- $L_k = \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^{k-1}$,
- $M_k = \{w\$w \,:\, w \in \{0,1\}^k\}$.

(c) 2NFAs sind nichtdeterministisch rechnende Zweiwege Automaten.

- Können 2NFAs nicht-reguläre Sprachen akzeptieren?
  Auch das lässt Mutter Natur nicht zu. Aber warum?
- Gibt es reguläre Sprachen, die 2NFAs mit sehr viel weniger Zuständen akzeptieren als 2DFAs? Diese Frage ist seit über 50 Jahren **offen**.

2NFAs akzeptieren nur reguläre Sprachen (1/5)

Wir simulieren den 2NFA $N = (\Sigma, Q_N, \delta_N, q_0, F_N)$ durch einen NFA N^*.

1. $\perp$ ist das Symbol für „undefiniert“ und $\mathcal{F}$ ist die Menge aller Funktionen

$$f : Q_N \to (Q_N \cup \{\perp\}).$$

2. Die Zustandsmenge von N^* ist

$$Q_{N^*} = Q_N \times \mathcal{F}.$$

Wenn N^* den Zustand (p, f) annimmt, dann spekuliert N^*, dass N eine Berechnung besitzt, die

(a) den bisher gelesenen Präfix der Eingabe zum **ersten Mal** im Zustand p nach rechts hin verlässt und

(b) den Präfix im Zustand $f(r)$ **nach rechts** verlässt, wenn es den Präfix im Zustand r **von rechts aus** betritt. ($f(r) = \perp$, wenn N den Präfix nicht mehr verlässt.)

Fazit: Der Zustand (p, f) wettet darauf, dass N den bisher gelesenen Präfix

(a) zum ersten Mal nach rechts im Zustand p verlässt wird und

(b) dass f die Reaktion auf einen „Besuch von rechts“ festhält.

3. Zustand p' ist für den Buchstaben a **konsistent** mit (p, f),

$$p' \in K((p, f), a),$$

wenn es eine „mit f konsistente Berechnung“ gibt, die

- im Zustand p „beginnt“ und
- zum ersten Mal im Zustand p' den „Buchstaben a nach rechts hin verlässt“.

Formal: Es gibt eine Zustandsfolge

$$(p_1, q_1, \ldots, p_{k-1}, q_{k-1}, p_k) \text{ mit}$$

- $p_1 = p$,
- $(q_i, \text{ links}) \in \delta_N(p_i, a)$ und $p_{i+1} = f(q_i)$ für $i = 1, \ldots, k-1$ sowie $(p', \text{ rechts}) \in \delta_N(p_k, a)$.

2NFAs akzeptieren nur reguläre Sprachen (3/5)

Die Funktion $f : Q_N \to (Q_N \cup \{\bot\})$ sei beliebig.

4. Definition von N^*:
   - N^* darf von seinen Startzustand q_0^* in den Zustand (p, f) wechseln, falls
     $$(p,\ \text{rechts}) \in \delta_N(q_0, \alpha) \quad \text{und} \quad (f(r),\ \text{rechts}) \in \delta_N(r, \alpha) \text{ für alle } r.$$
     Wenn N^* den Zustand (p, f) annimmt: N besitzt eine Berechnung, die α im Zustand p nach rechts verlässt und f ist die Reaktion dieser Berechnung auf einen Besuch von α, von rechts kommend. ($f(r) = \bot$, wenn $\delta_N(r, \alpha) = \emptyset$.)
   - N^* besitzt einen Zustandsübergang
     $$(p', f') \in \delta_{N^*}\Big((p, f), a\Big),$$
     (a) wenn $p' \in K((p, f), a)$ und
     (b) für alle Zustände $r \in Q_N$: Wenn es eine Berechnung gibt, die a von rechts im Zustand r betritt, dann ist $f'(r) = \bot$ oder a wird im Zustand $f'(r)$ nach rechts wieder verlassen.

Wie formalisiert man Teil (b)? Es ist $(f'(r),\ \text{rechts}) \in \delta_N(r, a)$ *oder*
es gibt einen Zustand $s \in Q_N$, so dass $(s,\ \text{links}) \in \delta_N(r, a)$ und $f'(r) \in K((s, f), a)$.

2NFAs akzeptieren nur reguläre Sprachen (4/5)

Und wenn N eine Eingabeposition **mehrmals** in demselben Zustand r besucht? Dann reagiert die Berechnung doch möglicherweise auf „Rechts-Besuche“ im Zustand r mit **verschiedenen** Nachfolgezuständen?

O.B.d.A.: Wir können uns auf Berechnungen von N beschränken, die jede Eingabeposition **höchstens einmal** im selben Zustand besuchen.

2NFAs akzeptieren nur reguläre Sprachen (5/5)

5. Erfinde einen neuen Zustand „ja“ für N^* und setze $F_{N^*} := \{ \text{ ja } \}$.
6. Wenn N^* das Endesymbol β im Zustand (p, f) liest:
   Wann sollte N^* akzeptieren?
   - Wenn $p' \in K((p, f), \beta)$ für einen Zustand $p' \in F_N$ gilt.
     Definiere in einem solchen Fall

     $$\delta_{N^*}((p, f), \beta) := \{(\text{ ja }, \text{rechts })\}.$$

   - In allen anderen Fällen ist $\delta_{N^*}((p, f), \beta) := \emptyset$.
7. Zeige durch vollständige Induktion über die Länge der Eingabe αu:

   N^* nimmt genau dann den Zustand (p, f) nach Lesen von αu an, wenn N eine Berechnung hat, die
   - ★ αu zum ersten Mal im Zustand p verlässt
   - ★ und αu im Zustand $f(r)$ verlässt, falls αu im Zustand r von rechts betreten wird.

N akzeptiert $w \Leftrightarrow N^*$ akzeptiert $\alpha w \beta$.

2DFAs vs 2NFAs: Die Zustandszahl (1/2)

Die Sprache

$$L_k := \{0,1\}^* \cdot \{1\} \cdot \{0,1\}^{k-1}$$

„trennt“ DFAs und NFAs:

- Es gibt einen NFA für L_k mit $k+1$ Zuständen,
- jeder DFA benötigt mindestens 2^k Zustände.

Gibt es eine reguläre Sprache mit **kleinen** 2NFAs, aber „**super-polynomiell**“ großen 2DFAs?

(a) Warum ist die Frage interessant?

Dies ist die „einfachste“ Version der Frage, ob „Speicherplatz“ signifikant eingespart werden kann, wenn nichtdeterministisch gerechnet werden darf.

(b) Warum ist die Frage schwierig?

2DFAs können Teile der Eingabe mehrfach besuchen: Der Myhill-Nerode Index ist nicht anwendbar.

2DFAs vs 2NFAs: Die Zustandszahl (2/2)

Die Sprache $\mathbf{B_k}$, die möglicherweise 2DFAs und 2NFAs trennt:

(a) Σ_k sei die Menge aller gerichteten bipartiten Graphen mit den Knotenmengen $L := \{\texttt{links}\} \cdot \{1, \ldots, \texttt{k}\}$ und $R := \{\texttt{rechts}\} \cdot \{1, \ldots, \texttt{k}\}$ sowie der Kantenmenge $E \subseteq L \times R$.

(b) Für ein Wort $w = w_1 \cdots w_n \in \Sigma_k^n$ identifiziere die Knotenmenge R von w_i mit der Knotenmenge L von w_{i+1}. Wir erhalten einen n-partiten Graphen $G(w)$.

- Die Knoten von w_1, die zu L gehören, sind die **Quellen** von $G(w)$,
- die Knoten von w_n, die zu R gehören, sind die **Senken** von $G(w)$.

(c) $B_k = \{w \in \Sigma_k^* :$
Es gibt einen Weg in $G(w)$ von einer Quelle zu einer Senke$\}$.

B_k wird von einem NFA mit k Zuständen akzeptiert,

2DFAs „scheinen“ $2^{\Omega(k)}$ Zustände zu benötigen.

Zusammenfassung: Reguläre Sprachen (1/2)

(1) Wir haben deterministische Automaten effizient **minimiert**.

- Dazu haben wir den **Äquivalenzklassenautomat** mit dem Algorithmus von Moore, bzw. dem Algorithmus von Hopcroft berechnet.
- Um Minimalität nachzuweisen, haben wir die **Myhill-Nerode Relation** definiert $\Longrightarrow$ der Äquivalenzklassenautomat ist minimal.
- Bis auf eine Umbenennung der Zustände stimmen alle minimalen Automaten mit dem **Myhill-Nerode Automaten** überein:
  - ⋆ Der **Index** ist eine untere Schranke für die Anzahl benötigter Zustände.
  - ⋆ Der Satz von Myhill-Nerode: Eine Sprache L ist genau dann regulär, wenn der Index von L endlich ist.

(2) Mit dem **Index** oder dem **Pumping-Lemma** kann man Nicht-Regularität nachweisen.

Zusammenfassung: Reguläre Sprachen (2/2)

(3) Nichtdeterministische Automaten erlauben eine kompaktere Beschreibung.
- Die **Potenzmengenkonstruktion** haben wir für die Bestimmung äquivalenter deterministischer Automaten benutzt.
- Mit der **Fooling Set Methode** haben wir ein Werkzeug zum Nachweis unterer Schranken für die Größe von NFAs kennengelernt.

(4) DFAs und NFAs akzeptieren genau die Klasse der regulären Sprachen.
- **Reguläre Grammatiken** und **reguläre Ausdrücke** defineren ebenfalls genau die Klasse der regulären Sprachen.
- Vernünftige probabilistische endliche Automaten (**PFAs mit positiver Lücke**), **alternierende Automaten** und **Zweiwege Automaten**, also 2NFAs akzeptieren genau die Klasse der reguläre Sprachen.

(5) Reguläre Sprachen sind **abgeschlossen** unter:
- Komplement, Vereinigung, Durchschnitt, Konkatenation, Stern-Bildung, (inversen) Homorphismen, Substitution und Quotientenbildung.

(6) Viele **Entscheidungsfragen** können effizient für DFAs beantworted werden. Für NFAs ist schon die Frage „$L(N) \neq \Sigma^*$?“ außerst schwierig.
- Aber das Wortproblem „$w \in L(N)$?“ ist effizient lösbar.