Automaten und formale Sprachen
Vorlesung
Sommersemester 2023

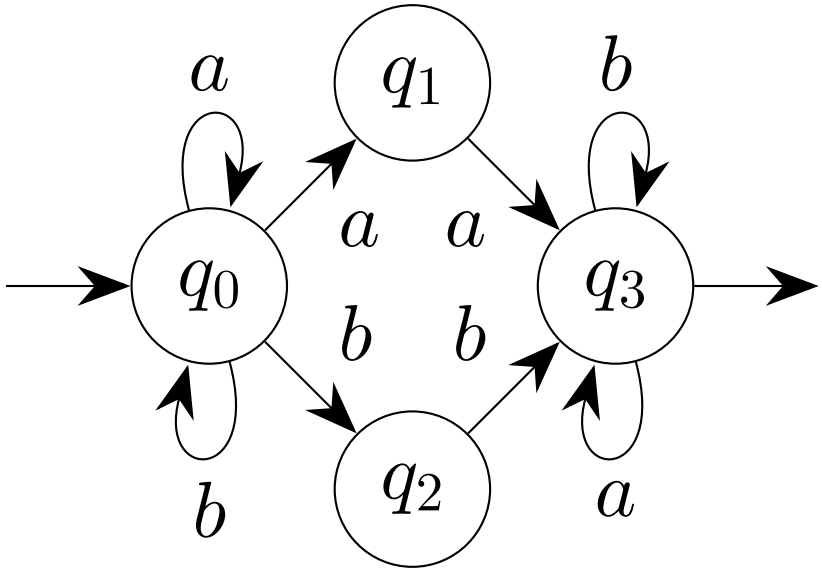
Johannes Waldmann, HTWK Leipzig

7. Juli 2023

Einleitung

Beispiel 1

- nichtdeterministischer endlicher Automat A:
  markierter gerichteter Graph
  Alphabet (Markierungen) $\{a, b\}$,
  Zustandsmenge $\{q_0, q_1, q_2, q_3\}$,
  Zustand q_0 initial, q_3 final



- Bsp: Rechnung von A ist $q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_3 \xrightarrow{b} q_3$

  Folge der Markierungen $abaab$ ist Element von Lang(A)

- was ist das Komplement von Lang(A)?

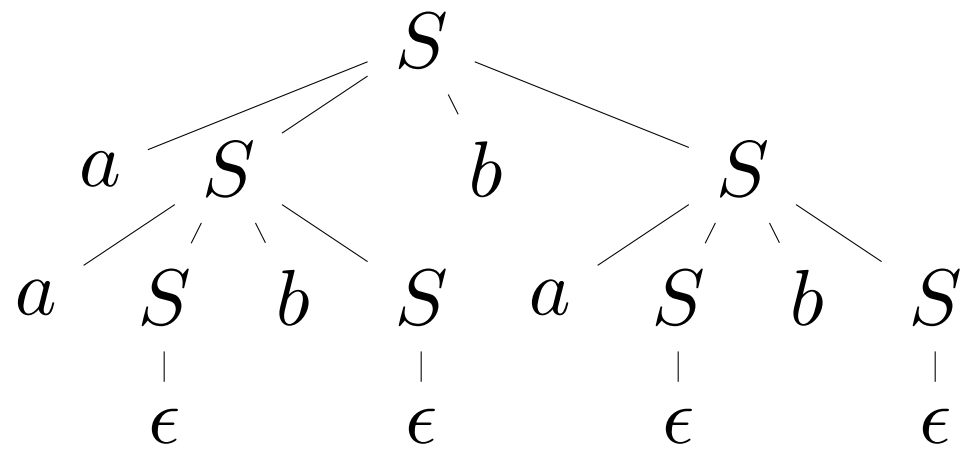
Inhalt von Beispiel 1

- Def: nichtdeterministischer endlicher Automat A
- Def: Sprache Lang(A) des Automaten A
- Def: die Klasse der regulären Sprachen (durch endliche Automaten darstellbaren)
- Def: Sprach-Operation Komplement
- Satz: die Menge der regulären Sprachen ist unter Komplement abgeschlossen,

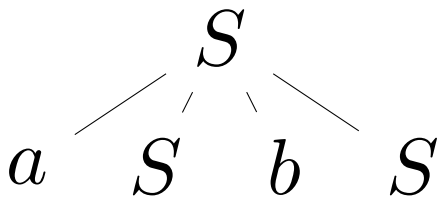
  Beweis durch Algorithmus zur Konstruktion eines vollständigen deterministischen Automaten B mit Lang$(A) =$ Lang(B), Komplement von B ist dann einfach

Beispiel 2

- `(2*(3+4)-5)*(6+7)` arithmetischer Ausdruck
  `(  (  )  )  (  )` Klammern, umbenannt: $aabbab$
- die Dyck-Sprache $D = \{\epsilon, ab, aabb, abab, \dots\} \subseteq \{a, b\}^*$ ist die Menge aller solcher Klammer-Wörter.
- D = Menge der Randwörter der Ableitungsbäume der kontextfreien Grammatik mit Regeln $S \to \epsilon$,



$S \to$



- D ist nicht durch endlichen Automaten darstellbar.

Inhalt von Beispiel 2

- Def: kontextfreie Grammatik (CFG) G
- Def: Sprache Lang(G) von G (Menge der mit G ableitbaren Wörter $=$ Menge der Randwörter der G-Ableitungsbäume)
- Satz: wenn L_1 kontextfrei und L_2 regulär, dann $L_1 \cap L_2$ kontexfrei

  Beweis (konstruktiv): Algorithmus mit Eingabe: CFG G für L_1, Automat A für L_2, Ausgabe: CFG G für $L_1 \cap L_2$.
- Satz: wenn L regulär, dann ... (Schleifen-Eigenschaft)
- Anwendung: $D \cap a^*b^* = \{a^k b^k \mid k \geq 0\}$ hat diese Schleifen-Eigenschaft nicht, also ist D nicht kontextfrei

Beispiel 3

- Zeigen Sie, daß Lang$(G_1) =$ Lang(G_2) für
  - $G_1 = (\{a, b\}, \{S\}, S, \{S \to \epsilon, S \to aSbS\})$
  - $G_2 = (\{a, b\}, \{S\}, S, \{S \to \epsilon, S \to SaSb\})$

  zu jeder G_1-Ableitung $T \to_1^* w$ ist eine G_2-Ableitung $S \to_2^* w$ anzugeben und umgekehrt
- das Korrektur-Programm der autotool-Aufgabe soll entscheiden, ob Lang$(G_1) =$ Lang(G_2)
  (G_1 Aufgabenstellung, G_2 Einsendung)
- aber die Implementierung testet nur (für einige Wörter w, ob $w \in$ Lang$(G_1) \iff w \in$ Lang(G_2).
- bei zu wenigen Tests werden auch falsche G_2 akzeptiert. Geht das besser? Nein, und das kann man beweisen!

Inhalt Beispiel 3

- Def: das Äquivalenz-Problem E:
  die Menge aller Paare von CFG G_1, G_2 mit
  Lang$(G_1) =$ Lang(G_2)
- Satz: E ist nicht entscheidbar
  (Def: es gibt keinen Algorithmus, der für jedes Paar (G_1, G_2) von CFG nach endlich vielen Schritten korrekt angibt, ob $(G_1, G_2) \in E$.
- Beweismethode: ein bekanntes unentscheidbares Problem (das Halteproblem) wird auf das Äquivalenz-Problem *reduziert*: $H \leq E$
- Satz: $\forall A, B$: wenn B entscheidbar und $A \leq B$, dann A entscheidbar

Überblick über Anwendungen AFS

- Automaten: sind Zustands-Übergangs-Systeme (wenn endlich, dann einfach)

  Bsp: Status-Übergänge eines technischen Gerätes, eines (nebenläufigen) Systems, einer interaktiven Benutzerschnittstelle

- reguläre Sprachen zur Beschreibung unendlicher Tokenklassen von Programmiersprachen, Bsp: Bezeichner, Ganzzahl-Literale

- kontextfreie Grammatiken zur Beschreibung der Syntax von Programmiersprachen, zur Definition von (konkreten und abstrakten) Syntaxbäumen als Grundlage für Definition der Semantik

Historische Einordnung und Warnung

- eine Grundlage der Informatik (Bsp.: Noam Chomsky: *Three models for the description of language* 1956, John Conway: *Regular algebra and finite machines*, 1971)
- mit ebenso frühen praktischen Anwendungen
  - Def. von Progr.-sprachen, Backus-Naur-Form (1959)
  - UNIX-Standard-Werkzeuge: grep, lex, yacc, u.a. für Implementierung von Interpretern und Compilern (gcc)
- die Gefahren in der Praxis sind (bis heute)
  - bekannte Methoden ignorieren
  - oder falsch (ineffizient) neu implementieren
  - Methoden an falscher Stelle anwenden (z.B. reguläre Ausdrücke für nicht reguläre Sprachen)

Systematische Einordnung

- Modellierung (von Zustandsübergangssystemen durch endlichen Automaten, von Sprachen durch Grammatiken,. . .)
- Analyse: Algorithmen zum Entscheiden von Eigenschaften der Modelle
- Synthese: Algorithmen zum Herstellen (Kombinieren) von Modellen
- Grenzen: nicht jede Aufgabe läßt sich so modellieren (Bsp. gewisse Sprachen sind nicht regulär)
- Grenzen: nicht jede Eigenschaft läßt sich entscheiden (Bsp. Sprach-Äquivalenz von CFG)

Literatur

- Standard-Lehrbücher, z.B.
  John E. Hopcroft, Jeffrey D. Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley 1979 `https://archive.org/details/HopcroftUllman_cinderellabook/`
- dieses Skript `https://www.imn.htwk-leipzig.de/~waldmann/lehre.html`,
  Skript von Frau Prof. Schwarz
- Original-Quellen, z.B. frühe Jahrgänge der *Zeitschrift Theoretical Computer Science*
- Quelltexte zu AFS-Aufgaben in autotool

Organisation

- wöchentlich Vorlesung,

  dort werden Hausaufgaben gestellt:

- Aufgaben autotool: 50 Prozent der Pflichtaufgaben

  Präsenations-Aufgaben (für Übung): jeder 3 mal.

  (jede Woche 3 Aufgaben mit 3 Teilaufgaben, also 9 Personen (in 3 Dreier-Gruppen). nach 4 Wochen war jeder einmal dran, das Semester hat 14 Wochen)

- Planung der Hausaufgaben:

- schließlich Klausur 120 min ohne Hilfsmittel

Wiederholung: Wörter

- Def: ein Alphabet ist eine nichtleere Menge von Zeichen (meist endlich, meist bezeichnet mit Σ)
- Def: ein Wort w über einem Alphabet Σ
  ist eine endliche Folge $[w_1, w_2, \ldots, w_n]$ mit $\forall i : w_i \in \Sigma$
  Def: in dieser Darstellung heißt n die Länge von w,
  Schreibweise: $|w| = n$
- Def: die Menge aller Wörter über Σ heißt Σ^*.
  Vorsicht: jedes Wort ist endlich (hat endliche Länge).
  Die Menge Σ^* ist unendlich.
  Def: die Menge aller Wörter über Σ der Länge n heißt Σ^n.
  Def: . . . Länge $\leq n$ heißt $\Sigma^{\leq n}$.

W.: Operationen und Relationen auf Wörtern

- Operation: Verkettung $u \cdot v$
  bezeichnet das Wort w der Länge $|u| + |v|$
  mit $w_i =$ wenn $(i \leq |u|)$ dann u_i sonst $v_{i-|u|}$.
- Satz: die Verkettung ist assoziativ
  (aber nicht kommutativ)
- Satz: das Wort . . . ist dafür links und rechts neutral
- Satz: Die Struktur $(\Sigma^*, \ldots, \cdot)$ ist ein Monoid
- Relationen (Satz: jede davon ist Halbordnung auf Σ^*)
  - die Präfix-Relation $u \leq_{\mathrm{P}} v \iff \exists q : u \cdot q = v$
  - die Infix-Relation $u \leq_{\mathrm{I}} v \iff \exists p, q : p \cdot u \cdot q = v$
  - die Suffix-Relation $u \leq_{\mathrm{S}} v \iff \exists p : p \cdot u = v$

W.: Sprachen, Sprach-Operationen

- Def: eine Sprache L über Σ ist eine Menge $L \subseteq \Sigma^*$.
- Operationen: Vereinigung, Durchschnitt, Differenz
- Op.: Verkettung: $A \cdot B := \{u \cdot v \mid u \in A \wedge v \in B\}$.
- Satz: Die Verkettung von Sprachen ist assoziativ.
- Satz: das dafür links und rechts neutrale Element ist ...
- Satz: Die Menge aller formalen Sprachen über Σ bildet einen *Halbring*,
  mit: Addition $=$ Vereinigung, Multiplikation $=$ Verkettung,
  Null-Element $= \emptyset$, Eins-Element $= \ldots$
  Distributiv-Gesetze $(A + B) \cdot C = (A \cdot C) + (B \cdot C)$,
  $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$,

Aufgaben

Für KW 15: Aufgaben 2, 3, 4

1. Hasse-Diagramme für: $\leq_{\mathrm{P}}, \leq_{\mathrm{I}}, \leq_{\mathrm{S}}$ auf $\{a, b\}^{\leq 3}$.

2. für die Sprachen: $A = \{a\}^*, B = \{b\}^*, C = \{a, b\}^*$:

   und die Sprachen: $A \cdot B, B \cdot A, A \cdot C, C \cdot A,$

   $A \cup B, A \cap B, C \setminus A, A \setminus C$, Zusatz: $(C \setminus A) \cdot (C \setminus B)$

   das Hasse-Diagramm für die Teilmengen-Relation.

   Bsp: $\forall u \in A : u \in C$, also $A \subseteq C$.

   Bsp: $ab \in C, ab \notin A$, also $C \not\subseteq A$.

3. R bezeichnet die Relation $(\leq_{\mathrm{P}}) \cup (\leq_{\mathrm{S}})$ auf $\{a, b\}^*$

   Bsp: $(ba, aba) \in R$, denn $ba \leq_{\mathrm{S}} aba$. Bsp: $(ab, aba) \in R$.

   Ist R reflexiv? symmetrisch? antisymmetrisch? transitiv?

4. Hasse-Diagramm für die Teilmengen-Relation auf den Relationen

   $\leq_{\mathrm{P}}, \leq_{\mathrm{I}}, \leq_{\mathrm{S}}, \leq_{\mathrm{P}} \circ \leq_{\mathrm{I}}, \leq_{\mathrm{I}} \circ \leq_{\mathrm{P}}, \leq_{\mathrm{P}} \circ \leq_{\mathrm{S}}, \leq_{\mathrm{S}} \circ \leq_{\mathrm{P}}, =$

   Bsp: $\forall u, v : u \leq_{\mathrm{P}} v \Rightarrow u \leq_{\mathrm{I}} v$, also $(\leq_{\mathrm{P}}) \subseteq (\leq_{\mathrm{I}})$.

   Bsp: $b \leq_{\mathrm{I}} aba, \neg(b \leq_{\mathrm{P}} aba)$, also $(\leq_{\mathrm{I}}) \not\subseteq (\leq_{\mathrm{P}})$.

Reguläre Ausdrücke

W: Operationen auf Sprachen

- Vereinigung: $A \cup B = \{w \mid w \in A \vee w \in B\}$
- Verkettung: $A \cdot B = \{u \cdot v \mid u \in A \wedge v \in B\}$
- iterierte Verkettung (eine fixierte Anzahl der Iterationen):

  $A^0 = \{\epsilon\}, \quad$ für $k \in \mathbb{N} : A^{k+1} = A \cdot A^k$
- iterierte Verkettung (alle endlichen Iterationen):

  $A^* = \bigcup_{k \in \mathbb{N}} A^k$

  Bsp: $w = 0011010111 \in \{0, 101, 11\}^*$,

  denn $w = 0 \cdot 0 \cdot 11 \cdot 0 \cdot 101 \cdot 11 \in \{0, 101, 11\}^6$,

Monotonie-Eigenschaften der Verkettung

- Satz: $A \subseteq B \Rightarrow (A \cdot C) \subseteq (B \cdot C)$
- Beweis: gegeben ist $\forall w : (w \in A \Rightarrow w \in B)$.
  zu zeigen ist: $\forall w : (w \in (A \cdot C) \Rightarrow w \in (B \cdot C))$.
  äq. $\forall w : (\exists u, v : u \in A \wedge v \in C \wedge w = uv) \Rightarrow w \in (B \cdot C))$.
  wegen $(p \Rightarrow q) \equiv (\neg p \vee q)$, $(\neg \exists x : R(x)) \equiv (\forall x : \neg R(x))$
  äq. $\forall w : (\forall u, v : u \notin A \vee v \notin C \vee w \neq uv) \vee w \in (B \cdot C))$.
  äq. $\forall w, u, v : u \notin A \vee v \notin C \vee w \neq uv \vee w \in (B \cdot C)$.
  äq. $\forall w, u, v : (u \in A \wedge v \in C \wedge w = uv) \Rightarrow w \in (B \cdot C)$.
  äq. $\forall w, u, v : \cdots \Rightarrow \exists u', v' : u' \in B \wedge v' \in C \wedge w = u'v'$
  ist wahr, weil wir wählen können $u' = u, v' = v$
- Satz: $B \subseteq C \Rightarrow A \cdot B \subseteq A \cdot C$, Beweis ähnlich.

Eigenschaften der Stern-Operation

- Satz: $A \subseteq B \Rightarrow A^* \subseteq B^*$
- Hilfsssatz: $A \subseteq B \Rightarrow \forall k \in \mathbb{N} : A^k \subseteq B^k$.

  Beweis: Induktion nach k. Anfang: $A^0 = \{\epsilon\} = B^0$.

  Schritt: $A^{k+1} = A \cdot A^k \subseteq B \cdot A^k \subseteq B \cdot B^k = B^{k+1}$.
- Beweis des Satzes: $A^* = \bigcup_{k \in \mathbb{N}} A^k \subseteq \bigcup_{k \in \mathbb{N}} B^k = B^*$.
- Satz: $A^* = \{\epsilon\} \cup A \cdot A^*$.
- Beweis: $A^* = \bigcup_{k \geq 0} A^k = A^0 \cup \bigcup_{k \geq 1} A^k =$
  $A^0 \cup \bigcup_{k \geq 0} (A \cdot A^k) = A^0 \cup A \cdot \bigcup_{k \geq 0} A^k = \{\epsilon\} \cup A \cdot A^*$

Abschluß-Eigensch. der Stern-Operation

- Satz: $A^* \cdot A^* = A^*$.

  Hilfssatz: $\forall k \in \mathbb{N} : A^k A^* \subseteq A^*$

  Induktion: Anfang: $A^0 A^* = \{\epsilon\} A^* = A^* \subseteq A^*$

  Schritt: $A^{k+1} A^* = (AA^k) A^* = A(A^k A^*) \subseteq AA^* \subseteq A^*$

  Beweis d. Satzes: $A^* = \{\epsilon\} A^* = A^0 A^* \subseteq A^* A^*$

  $A^* A^* = (\bigcup_{k \in \mathbb{N}} A^k) A^* = \bigcup_{k \in \mathbb{N}} (A^k A^*) \subseteq \bigcup_{k \in \mathbb{N}} A^* = A^*$

- Satz: $\forall k \geq 1 : (A^*)^k = A^*$. (Induktion.)

- Satz: $(A^*)^* = A^*$.

  Beweis: $(A^*)^* = \bigcup_{k \in \mathbb{N}} (A^*)^k = (A^*)^0 \cup \bigcup_{k \geq 1} (A^*)^k = (A^*)^0 \cup \bigcup_{k \geq 1} A^* = \{\epsilon\} \cup A^* = A^*$.

Reguläre Ausdrücke (Syntax)

- Def (Syntax) die Menge RX(Σ) der regulären Ausdrücke über Alphabet Σ ist die kleinste Menge M mit:
  - $\emptyset \in M$
  - $\epsilon \in M$
  - $\forall c \in \Sigma : c \in \Sigma$
  - $\forall X, Y \in M : (X + Y) \in M$
  - $\forall X, Y \in M : (X \cdot Y) \in M$
  - $\forall X \in M : (X^*) \in M$

```
data RX s
  = Empty
  | Epsilon
  | Letter s
  | Union (RX s) (RX s)
  | Dot   (RX s) (RX s)
  | Star  (RX s)
```

- Beispiel

```
Union Epsilon (Star (Dot (Letter False)(Letter True
:: RX Bool
```

- Übung: abstrakte Syntax für $\{0, 101, 11\}^*$

Reguläre Ausdrücke (Semantik)

- Def (Semantik): die Sprache Lang(E) von $E \in$ RX(Σ) ist:
  - Lang$(\emptyset) = \emptyset$ (die leere Sprache)
  - Lang$(\epsilon) = \{[]\}$
    (die Einermenge aus dem leeren Wort)
  - für $c \in \Sigma$: Lang$(c) = \{[c]\}$
    (die Einermenge mit dem Wort der Länge 1 mit dem Buchstaben c)
  - Lang$(X + Y) =$ Lang$(X) \cup$ Lang(Y)
  - Lang$(X \cdot Y) =$ Lang$(X) \cdot$ Lang(Y)
  - Lang$(X^*) =$ Lang$(X)^*$
- Bsp. $w = 0011010111 \in$ Lang$((0 + 1 \cdot 0 \cdot 1 + 1 \cdot 1)^*)$.
- Def: Die Menge REG(Σ) der *regulären Sprachen* über Σ ist $\{L \mid \exists E \in$ RX$(\Sigma) :$ Lang$(E) = L\}$.

Semantik reg. Ausdrücke (Erläuterung)

- für $E \in \mathsf{RX}(\Sigma), w \in \Sigma^*$: um $w \in \mathsf{Lang}(E)$ zu zeigen, muß für zusammengesetzten Ausdruck E angegeben werden:
  - falls $E = X \cup Y$: ob $w \in \mathsf{Lang}(X)$ oder $w \in \mathsf{Lang}(Y)$
  - falls $E = X \cdot Y$: eine Zerlegung $w = u \cdot v$ mit $u \in \mathsf{Lang}(X)$ und $v \in \mathsf{Lang}(Y)$
  - falls $E = X^*$: eine Zahl $k \in \mathbb{N}$ und Beweis für $w \in X^k$
- Beispiel: $w = 0100111 \in (01^*)^*$:
  - für den äußeren Stern wähle 3, zeige $0100111 \in (01^*)^4 = (01^*) \cdot ((01^*) \cdot (01^*))$
  - wähle Zerlegungen $0100111 = 01 \cdot (0 \cdot 0111)$
  - zeige $01 \in 01^*, 0 \in 01^*, 0111 \in 01^*$
  - zerlege $0 \cdot 1, 0 \cdot \epsilon, 0 \cdot 111$,
  - zeige $1 \in 1^*, \epsilon \in 1^*, 111 \in 1^*$, für die Sterne wähle $1, 0, 3$.

Semantik reg. Ausdrücke: Beweisbäume

- die vorige Begründung für $w \in \text{Lang}(E)$

  kann als Baum notiert werden (Ü: ergänzen!)

$$(*)\frac{(\cdot)\dfrac{0 \in 0, (*)\dfrac{1 \in 1, \epsilon \in 1^*}{1 \in 1^*}}{01 \in 01^*}, (*)\dfrac{\cdots}{00111 \in (01^*)^*}}{0100111 \in (01^*)^*}$$

- als Blätter (äußere Knoten): $\epsilon \in X^*$, $\forall c \in \Sigma : c \in c$,

  als Verzweigungsknoten (innere Knoten):

$$(\cup)\frac{w \in X}{w \in X + Y}, (\cup)\frac{w \in Y}{w \in X + Y},$$

$$(\cdot)\frac{u \in X, v \in Y}{uv \in XY}, (*)\frac{u \in X, v \in X^*}{u \cdot v \in X^*}$$

Eindeutige reguläre Ausdrücke

- ein regulärer Ausdruck $E \in \mathrm{RX}(\Sigma)$ heißt *eindeutig*, falls für jedes $w \in \mathrm{Lang}(E)$ genau ein Beweisbaum mit Wurzel $w \in E$ existiert. (sonst: mehrdeutig)
- Bsp: $(a+b)^*a(a+b)^*$ ist mehrdeutig, denn $aa \in (a+b)^0a(a+b)^1$ und $aa \in (a+b)^1a(a+b)^0$. Übung: ein dazu äquivalenter eindeutiger Ausdruck
- Satz: es ist entscheidbar ($=$ es gibt einen Algorithmus, der feststellt), ob X eindeutig ist.
  Satz: Zu jedem X gibt es einen eindeutigen Y mit $\mathrm{Lang}(X) = \mathrm{Lang}(Y)$
  Beweise (Konstruktionen) später (mit endl. Automaten)
- Bsp: $(a+b)^*(aa+bb)(a+b)^*$ ist mehrdeutig, äq. eind.?

Repräsentation von Mengen von Wörtern

- Realisierung der Semantik durch Programm?

  ```
  lang :: RX sigma -> Menge (Wort sigma)
  ```

- `Wort sigma`: einfach verkettete Liste

  ```
  type Wort sigma = [sigma]
  ```

- `Menge w`: kann unendlich sein! also keine Liste
  Ansatz: repräsentiere Menge von `w`
  durch ihre charakteristische Funktion

  ```
  type Menge w = (w -> Bool)
  ```

- `lang :: RX sigma -> ([sigma] -> Bool)`

- (später alles viel besser, mittels endlicher Automaten)

Naive Realisierung der Semantik reg. Ausdr.

- `lang :: RX sigma -> ([sigma] -> Bool)`

  Implementierung durch Fallunterscheidung nach dem Konstruktor des ersten Arguments

```
lang e = case e of
  Empty -> empty
  Epsilon -> epsilon
  Letter c -> singleton [c]
  Union x y -> union (lang x) (lang y)
  Dot x y   -> dot (lang x) (lang y)
  Star x    -> star (lang x)
empty = \ w -> False -- charakt. Fkt. d. leeren Men
singleton u = \ w -> (u == w) -- .. der Einermenge
epsilon = singleton []
```

Semantik der Verkettung

- $A \cdot B = \{u \cdot v \mid u \in A \wedge v \in B\}$
- für $|w| = n$: $w \in A \cdot B \iff$

  $\exists i \in \{0, 1, \ldots, n\} : w[1 \ldots i] \in A \wedge w[i+1 \ldots n] \in B$

  enthält Spezialfälle: $i = 0 \Rightarrow u = \epsilon$, $i = n \Rightarrow v = \epsilon$.

- ```
  dot a b = \ w ->
    any (\ i -> let (u,v) = splitAt i w
                in  ...
        )
    [0 .. length w]
  ```

  endlich viele Möglichkeiten ($|w| + 1$), alle probieren
  ```

Semantik der iterierten Verkettung (Ansatz)

- es gilt $A^* = \{\epsilon\} \cup A \cdot A^*$,
- entsprechendes Programm

```
star a = union epsilon (dot a (star a))
```

- terminiert nicht für Testfall

```
star eps [0]
  => (union eps (dot eps (star eps))) [0]
  => eps [0] || dot eps (star eps) [0]
  => False || dot eps (star eps) [0]
  => dot eps (star eps) [0]
  => (eps [] && star eps [0]) || ..
  => (True && star eps [0]) || ..
  => star eps [0] || ..
  => (union eps ...)  [0] || ..
```

Semantik der iterierten Verkettung (Lösung)

- wahr, aber ungeeignet: $A^* = \{\epsilon\} \cup A \cdot A^*$,
- wahr (Ü) und geeignet ist: $A^* = \{\epsilon\} \cup (A \setminus \{\epsilon\}) \cdot A^*$
- „geeignet“, denn:
  alle $u \in (A \setminus \{\epsilon\})$ haben Länge ≥ 1, deswegen wird char. Fkt. von A^* auf Wort v der Länge $< |w|$ angewendet
- Implementierung terminiert für Testfall (Übung)

```
star a =
  union epsilon (dot (minus a epsilon) (star a))
star eps [0]
  => (union eps (dot (minus eps eps) (star eps))) [
  => eps [0] || dot (minus eps eps) (star eps)) [0]
  => ... -- Hausaufgabe
```

Zusatz: Sternhöhe

- Def: die Sternhöhe des reg. Ausdrucks $E \in \mathsf{RX}(\Sigma)$ ist:
  - $\mathsf{sh}(\emptyset) = 0$, $\mathsf{sh}(\epsilon) = 0$, $\forall c \in \Sigma : \mathsf{sh}(c) = 0$
  - $\mathsf{sh}(X + Y) = \max(\mathsf{sh}(X), \mathsf{sh}(Y))$
  - $\mathsf{sh}(X \cdot Y) = \max(\mathsf{sh}(X), \mathsf{sh}(Y))$
  - $\mathsf{sh}(X^*) = 1 + \mathsf{sh}(X)$.
- Bsp: $\mathsf{sh}((01^*)^*) = 2$
- Def: $\mathsf{shl}(L)$ Sternhöhe der regulären Sprache L: $\min\{\mathsf{sh}(X) \mid \mathsf{Lang}(X) = L\}$.
- Bsp: $\mathsf{shl}((01^*)^*) = 1$, denn $(01^*)^* = \epsilon + 0(0 + 1)^*$
- Satz: $\forall k \geq 0 : \exists L : \mathsf{shl}(L) = k$. (Eggan 1963)
- Frage (1963): wie bestimmt man $\mathsf{shl}(L)$ für $L \in \mathsf{REG}$? Antwort (Algorithmus): Hashiguchi 1988, Kirsten 2005

Z: Erweiterte reg. Ausdrücke u. Sternhöhe

- Def: erweiterter regulärer Ausdruck ERX(Σ): ... und
  - (Syntax) $\forall X \in M : \overline{X} \in M$
    (Semantik) Lang($\overline{X}$) $= \Sigma^* \setminus$ Lang(X)
- Def: erweiterte Sternhöhe esh : ERX(Σ) $\to \mathbb{N}$
  wie bisher und: esh($\overline{X}$) = esh(X)
- Def: erweiterte Sternhöhe für Sprachen:
  eshl(L) $= \min\{$esh(X) $\mid X \in$ ERX(Σ), Lang(X) $= L\}$
  Bsp: (für $\Sigma = \{a, b\}$):
  - eshl(Σ^*) $= 0$, denn $\Sigma^* =$ Lang($\overline{\emptyset}$).
  - eshl(a^*) $= 0$, denn $a^* =$ Lang($\overline{\overline{\emptyset} \cdot b \cdot \overline{\emptyset}}$)
- Vermutung (offen seit 1980) $\forall L \in$ REG : eshl(L) ≤ 1.

Beispiel-Lösung einer Aufgabe

- für alle Sprachen A, B gilt: $(A \cup B)^* = A^*(BA^*)^*$
- Beweis (Teil 1): zeige $(A \cup B)^* \supseteq A^*(BA^*)^*$

  $A^*(BA^*)^* \subseteq (A \cup B)^*((A \cup B) \cdot (A \cup B)^*)^* \subseteq$
  $(A \cup B)^*((A \cup B)^*)^* \subseteq ((A \cup B)^*)^* = (A \cup B)^*$

  Test: jeden Schritt begründen (Verweis auf Satz in Skript)
- Beweis (Teil 2): zeige $(A \cup B)^* \subseteq A^*(BA^*)^*$

  mit Hilfssatz $\forall k \in \mathbb{N} : (A \cup B)^k \subseteq A^*(BA^*)^*$. Ind. nach k:

  Anfang: $(A \cup B)^0 = \{\epsilon\} \in A^0(BA^*)^0 \subseteq A^*(BA^*)^*$.

  Schritt: $(A \cup B)^{k+1} = (A \cup B)(A \cup B)^k \subseteq$
  $(A \cup B)A^*(BA^*)^* = AA^*(BA^*)^* \cup BA^*(BA^*)^* \subseteq \ldots$.

  Test: Schritte begründen, vervollständigen.

Aufgaben

SS 23: Aufgaben 1, 2, 3, 4, 5

1. geben Sie einen regulären Ausdruck an für die Menge der Binärdarstellungen aller natürlichen Zahlen, die durch vier teilbar sind.

   das gleiche für: . . . durch drei teilbar sind.

   Darstellung ohne führende Nullen, Darstellung der 0 ist ϵ.

2. Implementieren und testen Sie die Funktionen

   (a) `f :: RX sigma -> Bool`
   mit $f(X) \iff \epsilon \in \mathsf{Lang}(X)$.
   Direkte rekursive Implementierung! Eine indirekte Implementierung ist `f x = lang x []`. Eine direkte

Implementierung kann durch Spezialisierung der Implementierung von `lang` erhalten werden.

(b) `g :: RX sigma -> Bool`
mit $g(X) \iff$ Lang(X) ist leer.

(c) `h :: RX sigma -> Bool`
mit $h(X) \iff$ Lang(X) ist unendlich.

3. welche Ausdrücke sind mehrdeutig?

$ab^* + a^*bba^*$, $(ab^*)^*$, $(ab^*)^*(a^*b)^*$

einen äquivalenten eindeutigen Ausdruck für $(a+b)^*aa(a+b)^*$

4. Zu Folie „Semantik der iterierten Verkettung (Lösung)“

(a) Beweise Sie $A^* = \{\epsilon\} \cup (A \setminus \{\epsilon\}) \cdot A^*$

(b) vervollständigen Sie und bestätigen Sie damit Termination der Rechnung `star eps [0] => ...`

(c) wie teuer ist die Auswertung von $ba^k \in \text{Lang}((a+b)^*b(a+b)^k)$ in der angegebenen Implementierung?
Das heißt: eine Repräsentation `x` des regulären Ausdrucks für (z.B.) $k = 4$ in `ghci` schreiben, dann die Funktion `lang x [1,0,0,0,0]` auswerten. Für Zeit- und Platz-Messung vorher `:set +s` ausführen. Dann k ändern und diskutieren.

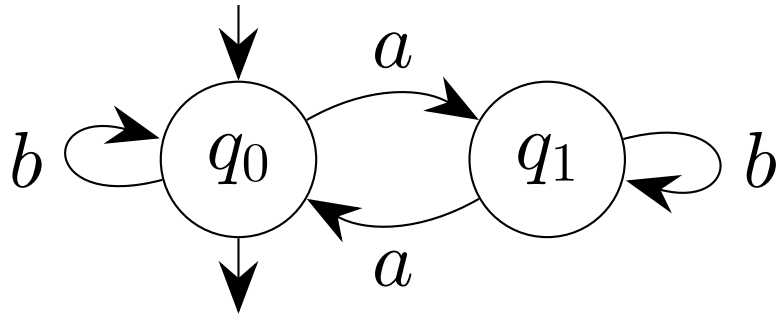
5. für alle Sprachen A, B gilt: $(AB)^* = \{\epsilon\} \cup A(BA)^*B$

6. Die Sprache $(ab)^*$ hat erweiterte Sternhöhe 0.

Deterministische endliche Automaten

Motivation, Beispiel

- Zustandsübergangssystem (Zustand $=$ Knoten, Übergang $=$ markierte Kante)



- Systemverhalten $=$ Menge der Pfadmarkierungswörter $=$ eine formale Sprache

  Bsp: Lang$(A) = \{w \mid w \in \{a,b\}^*, |w|_a \equiv 0 \pmod 2\}$

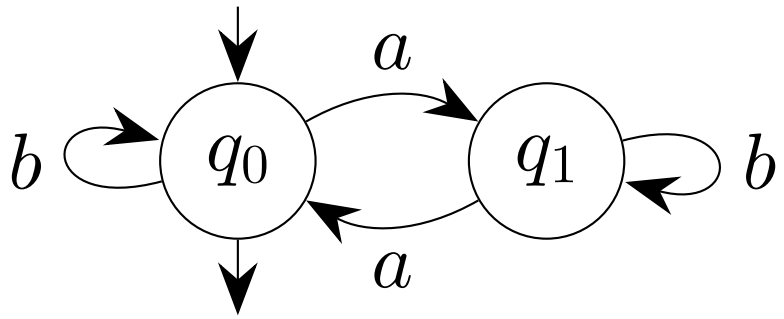
- Ziel: äquivalente, kompakte, effiziente Repräsentation von regulären Sprachen, Bsp: $b^*(ab^*ab^*)^*$

Definition (Syntax)

- Def: deterministischer Automat $A = (\Sigma, Q, i, F, \delta)$ mit
  - endliches Alphabet Σ, endliche Zustandsmenge Q,
  - Initialzustand $i \in Q$, Finalzustandsmenge $F \subseteq Q$
  - (Zeichen-)Übergangsfunktion $\delta : Q \times \Sigma \to Q$

  Abkürzung: DFA (deterministic finite automaton)

- Beispiel

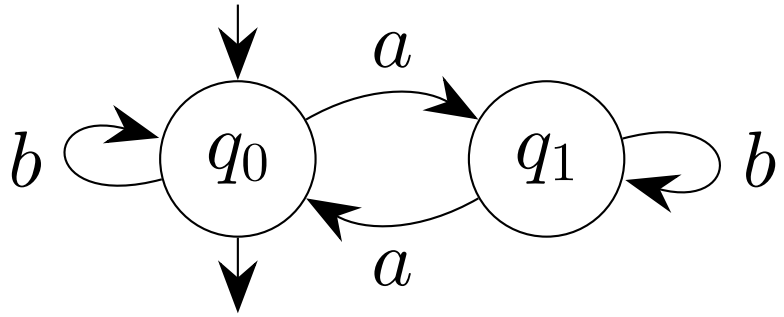


$$(\{a, b\}, \{q_0, q_1\}, q_0, \{q_0\},$$
$$\{((q_0, a), q_1), ((q_0, b), q_0), ((q_1, a), q_0), ((q_1, b), q_1)\}$$

Definition (Semantik)

- Def: Wort-Übergangsfunktion $\delta^* : Q \times \Sigma^* \to Q$
  durch Induktion nach Länge des zweiten Arguments:
  $\forall q \in Q : \delta^*(q, \epsilon) = q$,
  $\forall q \in Q, c \in \Sigma, w \in \Sigma^* : \delta^*(q, c \cdot w) = \delta^*(\delta(q, c), w)$.

- Beispiel



  $\delta^*(q_0, aba) = \delta^*(\delta(q_0, a), ba) = \delta^*(q_1, ba) = \delta^*(\delta(q_1, b), a)$
  $= \delta^*(q_1, a) = \delta^*(\delta(q_1, a), \epsilon) = \delta^*(q_0, \epsilon) = q_0$.

- Def: Sprache d. Automaten Lang$(A) = \{w \mid \delta^*(i, w) \in F\}$.
- „$w \in$ Lang(A)?“ kann in Zeit $|w|$ beantwortet werden.

Funktionale Modellierung

- Datentyp, Beispiel

```
data DFA s q = DFA
  { initial :: q
  , final :: q -> Bool
  , delta
     :: q -> s -> q
  }
```

```
data Sigma = A | B
a :: DFA Sigma Bool
a = DFA { initial = False
  , final = \ q -> not q
  , delta = \ q c ->
      case c of
        A -> not q ; B -> q }
```

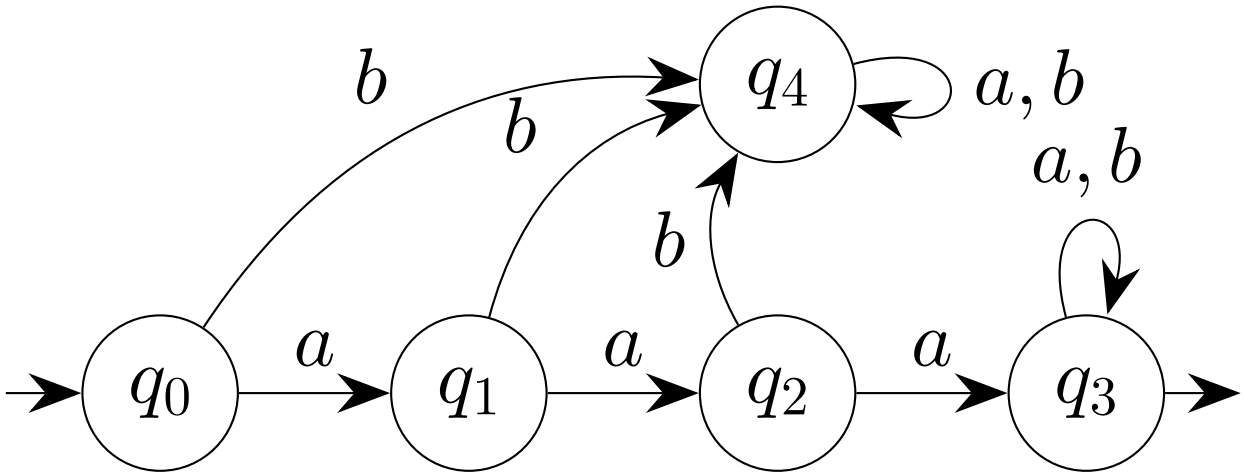
- Semantik

```
lang :: DFA s q -> Lang s
lang a = \ w ->
  final a (foldl (delta a) (initial a) w)
```

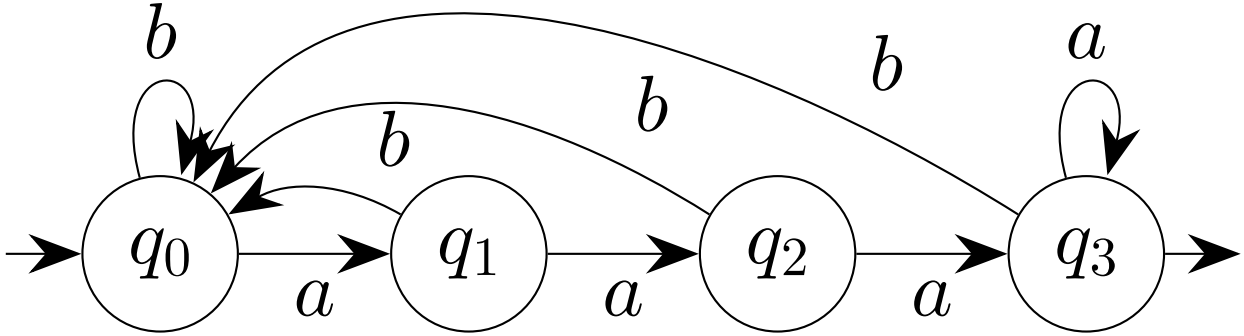
- Test `lang a [A,B,A,B]`

Beispiele (1)

- Sprache $a^3\{a,b\}^*$

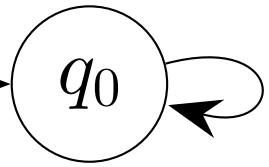


- Sprache $\{a,b\}^*a^3$

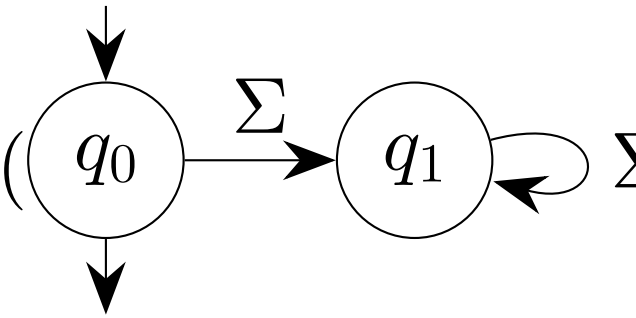


- ein DFA für die Sprache $\{a,b\}^3b\{a,b\}^*$
- ein DFA für die Sprache $\{a,b\}^*b\{a,b\}^3$

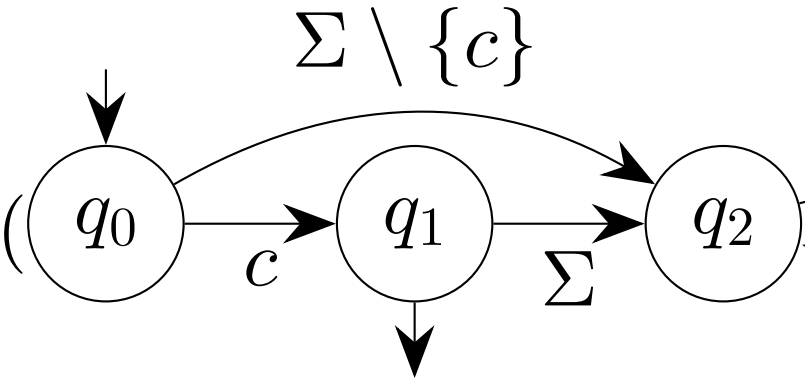
Beispiele (2)

- Lang(

$) = \emptyset$. Beachte: $i = q_0, F = \emptyset$

- Lang(

$) = \{\epsilon\}$. Hier $i = q_0, F = \{q_0\}$

- Lang(

$) = \{[c]\}$.

- ein DFA für die Sprache $\{w\}$ (für $w \in \Sigma^*$)

Ein Nicht-Beispiel

- Satz: Für $L = \{a^k b^k \mid k \geq 0\}$ existiert kein DFA A mit Lang$(A) = L$.

- Beweis (indirekt): falls doch, dann $L =$ Lang(A) für $A = (\{a, b\}, Q, i, F, \delta)$ mit $|Q| = n$.

  Es gibt $0 \leq p < q \leq n$ mit $\delta^*(i, a^p) = \delta^*(i, a^q)$. (Beweis: $\{\delta^*(i, a^0), \delta^*(i, a^1), \ldots, \delta^*(i, a^n)\}$ muß Duplikate enthalten)

  Es gilt $a^p b^p \in L =$ Lang(A), also $\delta^*(i, a^p b^p) = f \in F$.

  Dann gilt auch $\delta^*(i, a^q b^p) = f$, also $a^q b^p \in$ Lang(A),

  im Widerspruch zu $a^q b^p \notin L$.

Algor. f. Eigenschaften von DFA-Sprachen

- Gegeben DFA $A = (\Sigma, Q, i, F, \delta)$. Gefragt:
  Lang$(A) = \emptyset$?, $\epsilon \in$ Lang(A)?, Lang(A) ist unendlich?
- Satz: $\epsilon \in$ Lang$(A) \iff i \in F$. Beweis: $\delta^*(i, \epsilon) = i$.
- Vorsicht: es gilt *nicht* $\forall A$: (Lang$(A) = \emptyset \iff F_A = \emptyset$)
- Def: Zustand $q \in Q$ heißt
  - *erreichbar*, wenn $\exists w : \delta^*(i, w) = q$
  - *produktiv*, wenn $\exists w : \delta^*(q, w) \in F$.

  Menge E der erreichbaren, P der produktiven Zustände kann in $|\Sigma| \times |Q|$ Zeit bestimmt werden.
- Satz: Lang$(A) = \emptyset \iff E \cap P = \emptyset$
- Lang(A) unendl. gdw. $\exists p \in E \cap P, w \in \Sigma^+ : \delta^*(p, w) = p$.

Korrektheit des Algor. für Lang$(A) = \emptyset$

- Satz: Lang$(A) = \emptyset \iff E \cap P = \emptyset$
- Beweis für $\ldots \Leftarrow \ldots$ (indirekt, d.h., zeige $\neg \ldots \Rightarrow \neg \ldots$)
  Lang$(A) \neq \emptyset \Rightarrow \exists w \in$ Lang$(A) \Rightarrow \exists w : \delta^*(i, w) = f \in F$.
  Für dieses f gilt $f \in E$ (denn $\delta^*(i, w) = f$)
  und $f \in P$ (denn $\delta^*(f, \epsilon) = f \in F$)
  also $f \in E \cap P$, also $E \cap P \neq \emptyset$.
- Beweis für $\ldots \Rightarrow \ldots$, (indirekt, d.h., zeige $\neg \ldots \Leftarrow \neg \ldots$)
  $E \cap P \neq \emptyset \Rightarrow \exists q : q \in E \wedge q \in P$
  also $\exists u \in \Sigma^* : \delta^*(i, u) = q$ (erreichbar)
  und $\exists v \in \Sigma^* : \delta^*(q, v) \in F$ (produktiv).
  Dann $\delta^*(i, u \cdot v) = \delta^*(\delta^*(i, u), v) = \delta^*(q, v) \in F$,
  also $u \cdot v \in$ Lang(A), also Lang$(A) \neq \emptyset$.

Komplement einer DFA-Sprache

- Satz: wenn $A \in \mathrm{DFA}(\Sigma)$, dann existiert $B \in \mathrm{DFA}(\Sigma)$ mit $\mathrm{Lang}(B) = \Sigma^* \setminus \mathrm{Lang}(A)$.

- Beweis: sei $A = (\Sigma, Q, i, F, \delta)$. Dann $B = (\Sigma, Q, i, Q \setminus F, \delta)$.

  (nur die Menge der Finalzustände wird geändert.)

  Korrektheit: $w \notin \mathrm{Lang}(A) \iff \delta^*(i, w) \notin F \iff \delta^*(i, w) \in (Q \setminus F) \iff w \in \mathrm{Lang}(B)$.

- Kommentar: der Determinismus ist eine strenge Forderung, erlaubt aber diese einfache Konstruktion.

  Komplementierung ist für nicht-deterministische Modelle (Bsp: reguläre Ausdrücke) wesentlich umständlicher.

Synchrone DFAs (Kreuzprodukt-Konstruktion)

- zwei Automaten über dem gleichen Alphabet
  $A_1 = (\Sigma, Q_1, i_1, F_1, \delta_1), A_2 = (\Sigma, Q_2, i_2, F_2, \delta_2)$
  berechnen für jedes $w \in \Sigma^*$ ein Paar von Zuständen
  $(\delta_1^*(i_1, w), \delta_2^*(i_2, w)) \in Q_1 \times Q_2$.
- diese Rechnung wird realisiert durch den
  Kreuzprodukt-Automaten $A = (\Sigma, Q_1 \times Q_2, i, F, \delta)$
  mit $i = (i_1, i_2)$ und $\delta((q_1, q_2), c) = (\delta_1(q_1, c), \delta_2(q_2, c))$.
  Es gilt $\forall w : \delta^*(i, w) = (\delta_1^*(i_1, w), \delta_2^*(i_2, w))$ (Ind. nach $|w|$)
- durch passende Wahl von F erreicht man
  - $\mathsf{Lang}(\ldots, F_1 \times F_2, \ldots) = \mathsf{Lang}(A_1) \cap \mathsf{Lang}(A_2)$
  - $\mathsf{Lang}(\ldots, ?, \ldots) = \mathsf{Lang}(A_1) \cup \mathsf{Lang}(A_2)$
  - $\mathsf{Lang}(\ldots, ?, \ldots) = \mathsf{Lang}(A_1) \setminus \mathsf{Lang}(A_2)$

  $\Rightarrow$ Menge der DFA-Sprachen ist Boolesch abgeschlossen

Implementierung d. Kreuzprodukt-K.

- `cross :: DFA s q1 -> DFA s q2 -> DFA s (q1, q2)`
- Spezifikation

```
prop_cross  a1 a2 w =
  lang (cross a1 a2) w == (lang a1 w && lang a2 w)
```

- Test mit Leancheck, dazu automatisches (typgesteuertes) Aufzählen von Automaten (!)
- Implementierung

```
cross a1 a2 = DFA
  { initial = _
  , final = \ (q1, q2) -> _
  , delta = \ (q1, q2) c -> _
  }
```

Ausblick: Verkettung und Stern für DFA

- für $A_1, A_2 \in \mathrm{DFA}(\Sigma)$ gibt es *keine* direkte Konstruktion eines DFA für $\mathrm{Lang}(A_1) \cdot \mathrm{Lang}(A_2)$ oder $\mathrm{Lang}(A_1)^*$.

- wir werden eine indirekte Konstruktion kennenlernen:
  - (einfache) Konstruktion eines nichtdeterministischen Automaten für $\mathrm{Lang}(A_1) \cdot \mathrm{Lang}(A_2)$ bzw. $\mathrm{Lang}(A_1)^*$
  - (aufwendige) Konstruktion eines äquivalenten deterministischen Automaten.

- dann folgt dann, daß die Menge der regulären Sprachen in der Menge der DFA-Sprachen enthalten ist

- noch später zeigen wir, daß Gleichheit gilt (zu jedem DFA wird ein äquivalenter regulärer Ausdruck konstruiert)

Anwendung des Booleschen Abschlusses

- wir betrachten $L := \{x \mid w \in \{a,b\}^* \wedge |w|_a = |w|_b\}$.

  $L = \{\epsilon, ab, ba, aabb, abab, abba, baab, \ldots\}$

- Satz: es gibt keinen DFA A mit Lang$(A) = L$.

- Beweis (indirekt): falls Lang$(A) = L$:

  konstruiere DFA B für a^*b^* (3 Zustände).

  konstruiere Kreuzprodukt-Automaten C für Lang$(A) \cap$ Lang(B).

  dann Lang$(C) =$ Lang$(A) \cap a^*b^* = \{a^k b^k \mid k \geq 0\}$.

  das ist Widerspruch zu vorher gezeigter Aussage,

  daß $\{a^k b^k \mid k \geq 0\}$ keine DFA-Sprache ist.

Zusatz: synchronisierende Wörter

- Def: für DFA $A = (\Sigma, Q, i, F, \delta)$:

  ein Wort $w \in \Sigma^*$ heißt *synchronisierend*,

  falls $\exists t \in Q : \forall s \in Q : \delta^*(s, w) = t$.

  (egal, wo die Rechnung beginnt, sie endet in t)

- Vermutung (Cerny, 1964): Für alle A gilt: Falls es für A überhaupt ein synchronisierendes Wort gibt, dann auch eines mit Länge $\leq (|Q| - 1)^2$.

- siehe Jean-Eric Pin (2012) `https://www.irif.fr/~jep/Problemes/Cerny.html#Cerny`

Hausaufgaben

Schwerpunkte: 1–3 (Präsentation der anderen Aufgaben nur, soweit Zeit ist, evtl. auch in folgenden Wochen)

1. geben Sie einen DFA A an für die Menge der Binärdarstellungen aller natürlichen Zahlen, die durch drei teilbar sind.
   MSB (most significant bit) links, führende Nullen sind erlaubt.
   Für den Automaten $A = (\{0,1\}, Q, i, F, \delta)$ soll gelten:
   - Q repräsentiert die Restklassen nach dem Modul 3,
   - $\forall w \in \{0,1\}^* : \delta^*(i,w)$ *ist* die Restklasse von w (dem Zahlenwert der Binärzahl) nach dem Modul 3.

   Dadurch sind Q, i, F, δ eindeutig bestimmt.

2. Geben Sie einen möglichst kleinen DFA an für:
   (a) $\{001, 0101, 10101\}$
   (b) $\{w : w \in \{0,1\}^5 \wedge |w|_1 = 3\}$.
   Beweisen Sie, daß jede endliche Sprache durch einen DFA dargestellt werden kann. (Bauen Sie hierfür nicht einen kleinsten DFA, sondern einen kurzen Beweis.)

3. Wie muß die Menge F in der Kreuzprodukt-Konstruktion gewählt werden, um die symmetrische Differenz der Sprachen Lang(A_1), Lang(A_2) zu erhalten?
   Wenden Sie die Konstruktion an für die Automaten (mit jeweils zwei Zuständen) über $\Sigma = \{a, b\}$
   für Lang$(A_1) = \{w : |w|_a \equiv 0 \pmod 2\}$
   und Lang$(A_2) = \{w : |w|_b \equiv 0 \pmod 2\}$.
   Überprüfen Sie mit der funktionalen Implementierung.

4. Geben Sie (kleine) DFAs A_1, A_2 für jeweils unendliche Sprachen an, für die $\mathsf{Lang}(A_1) \cap \mathsf{Lang}(A_2) = \emptyset$ gilt. Diskutieren Sie Erreichbarkeit und Produktivität im Kreuzprodukt-Automaten.

5. Beweisen Sie: für $A = (\Sigma, Q, i, F, \delta)$ gilt:
   $\mathsf{Lang}(A)$ ist unendlich $\iff \exists w \in \mathsf{Lang}(A) : |Q| \leq |w|$.
   Geben Sie ein (nur von $|Q|$ abhängige) Zahl B an mit:
   $\mathsf{Lang}(A)$ ist unendlich
   $\iff \exists w \in \mathsf{Lang}(A) : |Q| \leq |w| \leq B$.

6. Zeigen Sie, daß die Sprache aller Dezimaldarstellungen von Quadratzahlen keine DFA-Sprache ist.
   Hinweis: indirekter Beweis. Bilden Sie den Durchschnitt mit einer geeigneten (sehr einfachen) DFA-Sprache. Es genügt, Ziffern $0, 1, 2$ zu verwenden (aber davon viele)

7. Synchronisierende Wörter, Cerny-Vermutung: Rechnen Sie Beispiele aus angegebener Quelle (J. E. Pin 2012) nach, auch H. Zantema (LATA 2017)

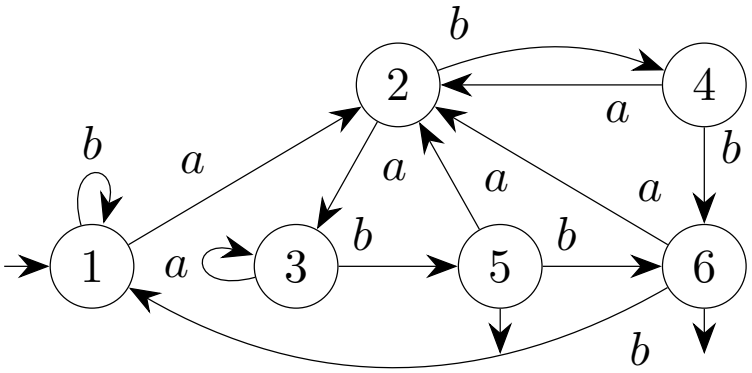
`https://research.tue.nl/en/publications/finding-dfas-with-maximal-shortest-synchronizing-wo`

Nichtdeterministische Automaten

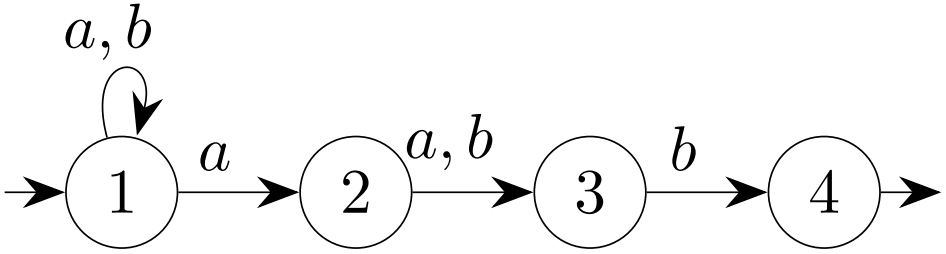
Motivation, Beispiel

- kleiner regulärer Ausdruck $X = (a + b)^* a(a + b)b$,

  großer äquivalenter DFA



- Lang(X) ist Pfadmarkierungssprache dieses markierten Graphen, dessen Übergangsrelation keine Funktion ist:



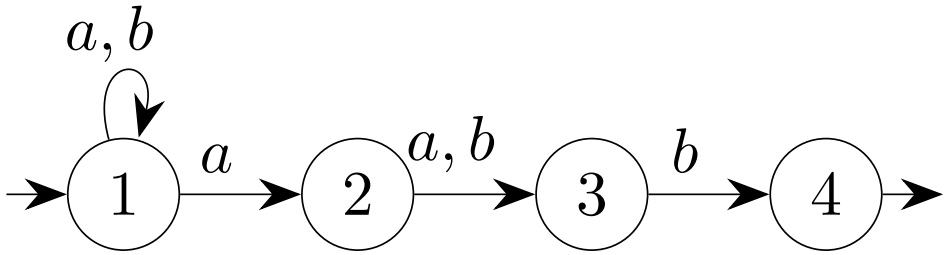
$\delta(1, a) = ?, \delta(4, b) = ?$

Definition NFA (Syntax)

- Def: nichtdeterministischer Automat $A = (\Sigma, Q, I, F, \delta)$ mit
  - endliches Alphabet Σ, endliche Zustandsmenge Q,
  - Initialzustandsmenge $I \subseteq Q$, Finalzustandsm. $F \subseteq Q$
  - (Zeichen-)Übergangsrelation (Ansatz) $\delta \subseteq Q \times \Sigma \times Q$.

  Abkürzung: NFA (non-deterministic finite automaton)

- Bsp:



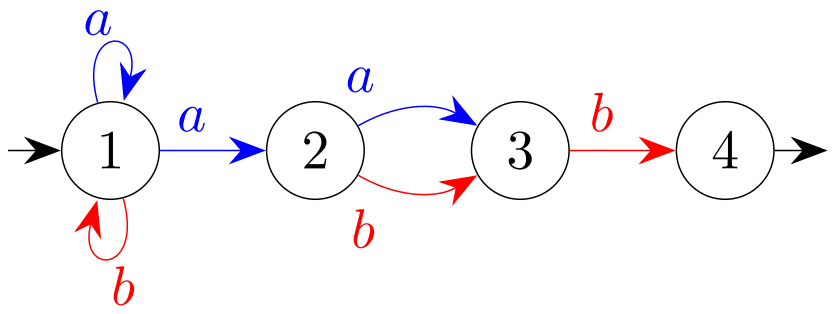
  $A = (\{a, b\}, \{1, 2, 3, 4\}, \{1\}, \{4\}, \delta)$ mit

  $\delta = \{(1, a, 1), (1, b, 1), (1, a, 2), (2, a, 3), (2, b, 3), (3, b, 4)\}$

- äquivalent, erlaubt kürzere Texte: $\delta : \Sigma \to (Q \times Q)$

  $\delta = \{(a, \{(1, 1), (1, 2), (2, 3)\}), (b, \{(1, 1), (2, 3), (3, 4)\})\}$

Diskussion NFA (Syntax)

- 

$\delta : \Sigma \to (Q \times Q)$

$\delta = \{(a, \{(1,1),(1,2),(2,3)\}), (b, \{(1,1),(2,3),(3,4)\})\}$

δ ordnet jedem Buchstaben eine Relation auf Q zu

- ist ökonomische Notation, weil man Eigenschaften von, und Operationen auf, Relationen verwenden kann:
  - $\delta(a)$ nicht nacheindeutig, $\delta(b)$ ist keine totale Funktion
  - Produkt (Verkettung, Nacheinander„ausführung“)

    $\delta(a) \circ \delta(b) = \{(1,1),(1,3),(2,4)\}$

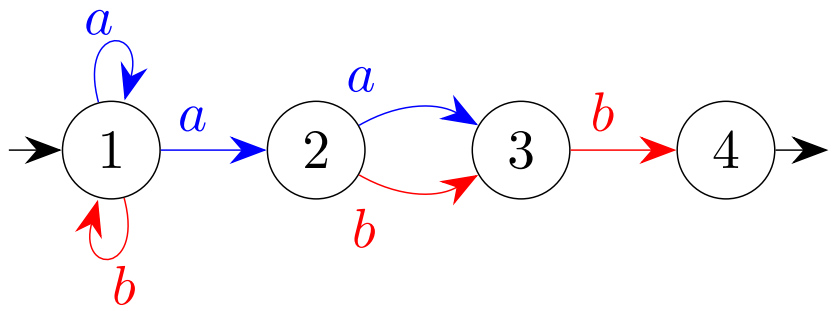
    $\delta(a) \circ \delta(b) \circ \delta(b) = (\delta(a) \circ \delta(b)) \circ \delta(b) = \{(1,1),(1,4)\}$

    $\delta(a) \circ \delta(b) \circ \delta(a) \circ \delta(b) = (\delta(a) \circ \delta(b)) \circ (\delta(a) \circ \delta(b)) = \{\ldots\}$

Definition NFA (Semantik)

- die Wort-Übergangs-Relation $\delta^* : \Sigma^* \to (Q \times Q)$
  Def. durch Induktion nach Länge des Arguments:
  $\delta^*(\epsilon) = \mathsf{Id}_Q, \quad \delta^*(c \cdot w) = \delta(c) \circ \delta^*(w)$

- Bsp:



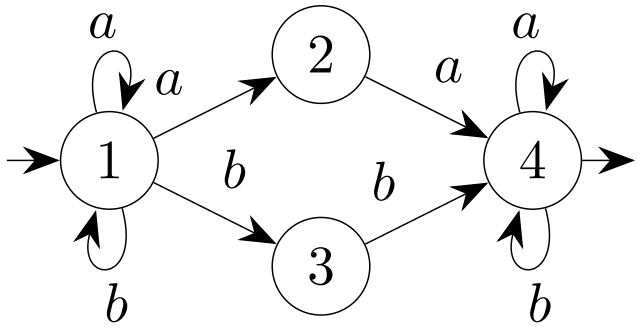
  $\delta^*(abb) = \delta(a) \circ \delta(b) \circ \delta(b) \circ \mathsf{Id}_Q = \{(1,1),(1,4)\}$

- Def: Sprache des Automaten $\mathsf{Lang}(A)$
  $= \{w \mid (\delta^*(w) \cap (I \times F)) \neq \emptyset\}$
  $= \{w \mid \exists i \in I, f \in F : (i,f) \in \delta^*(w)\}.$

- Bsp: $w = abb \in \mathsf{Lang}(A)$, denn $I \times F = \{(1,4)\}$,
  $\delta^*(w) \cap I \times F = \{(1,1),(1,4)\} \cap \{(1,4)\} = \{(1,4)\} \neq \emptyset.$

Beispiele, Ausblick

- $(a+b)^*(a^2+b^2)(a+b)^*$,



- $\forall w \in \Sigma^*$ mit $w = w_1 \dots w_n$: $\{w\} = \mathsf{Lang}(A)$ für $A = (\Sigma, \{0, \dots, n\}, \{0\}, \{n\}, \delta)$ mit $(i-1, i) \in \delta(w_i)$
- Satz: jeder DFA ist ein NFA.

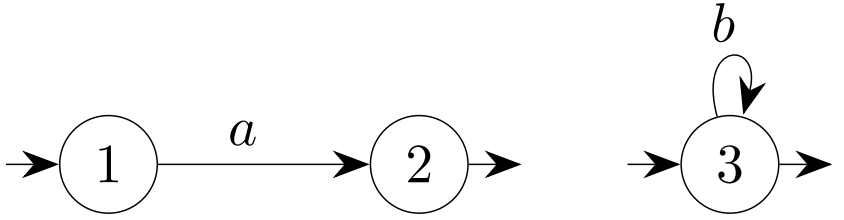
  Begründung: jede Funktion ist eine Relation.
- nächste Schritte: NFA $\rightarrow$ DFA (heute), RX $\leftrightarrow$ NFA

  in kleinen Beispielen scheint das immer möglich

Anwendung: disjunkte Vereinigung für NFA

- gegeben: NFAs $A_1 = (\Sigma, Q_1, I_1, F_1, \delta_1)$,
  $A_2 = (\Sigma, Q_2, I_2, F_2, \delta_2)$ mit $Q_1 \cap Q_2 = \emptyset$.
  Dann gilt für $A = (\Sigma, Q_1 \cup Q_2, I_1 \cup I_2, F_1 \cup F_2, \delta_1 \cup \delta_2)$:
  $\mathsf{Lang}(A) = \mathsf{Lang}(A_1) \cup \mathsf{Lang}(A_2)$.

- Bsp: $A_1 = (\{a, b\}, \{1, 2\}, \{1\}, \{2\}, \{(a, \{(1, 2\}), (b, \emptyset)\})$,
  $A_2 = (\{a, b\}, \{3\}, \{3\}, \{3\}, \{(a, \emptyset), (b, \{(3, 3\})\})$.



$A =$
$(\{a, b\}, \{1, 2, 3\}, \{1, 3\}, \{2, 3\}, \{(a, \{(1, 2)\}), (b, \{(3, 3\})\})$.

- mglw. mehrere Initial-Zustände, das geht mit DFA nicht
  (aber $\cup$ für DFA mit Kreuzprodukt-Konstruktion)

Eigensch. von NFA-Sprachen, Reduzierte NFA

- Satz: $\epsilon \in \text{Lang}(A) \iff I \cap F \neq \emptyset$.
- Def: erreichbare (E)/produktive (P) Zustände
  $E = \{q \mid \exists i \in I, w \in \Sigma^* : (i, q) \in \delta^*(w)\}$
  $P = \{q \mid \exists f \in F, w \in \Sigma^* : (q, f) \in \delta^*(w)\}$.
- Satz: $\text{Lang}(A) = \emptyset \iff E \cap P = \emptyset$.
- Def: ein NFA heißt *reduziert*, wenn $Q = E = P$
- Satz: $\forall A \in \text{NFA}(\Sigma) : \exists B \in \text{NFA}(\Sigma)$: B ist reduziert und $\text{Lang}(A) = \text{Lang}(B)$.
  Beweis (Konstruktion): Zustandsmenge einschränken auf $(E \cap P)$, auch I, F, δ entsprechend einschränken
  Testfrage: warum geht das nicht für DFA?

Das Wortproblem für NFA (Motivation)

- Def: das Wortproblem ist die Spezifikation:

  gegeben NFA A über Σ, Wort $w \in \Sigma^*$,
  gesucht: Wahrheitswert von $w \in \mathsf{Lang}(A)$.

- die naive Methode ist: alle mit w beschrifteten Pfade in A durchprobieren. Das können aber exponentiell viele sein.

- die Methode lt. Definition ist: die Matrix der Relation $\delta^*(w)$ ausrechnen, das dauert $|w| \cdot |Q|^3$. Immerhin schon polynomiell. (Matrix-Mult. im Bool-Halbring geht sogar etwas schneller, brauchen wir aber nicht, denn...)

- wir wollen nicht die volle Relation (Matrix) $\delta^*(w)$, sondern nur das Bild der Menge I unter $\delta^*(w)$

Menge $=$ Vektor, Relation $=$ Matrix

- Für alle Bilder einer Menge $M \subseteq Q$ unter einer Relation $R \subseteq Q \times Q$ $\{y \mid \exists x \in M : (x, y) \in\}$ schreiben wir $M \circ R$.

  Bsp: $Q = \{1, 2, 3, 4\}, M = \{1, 2\}, R = \{(1, 1), (2, 3), (3, 4)\}$

  $M \circ R = \{1, 3\}$.

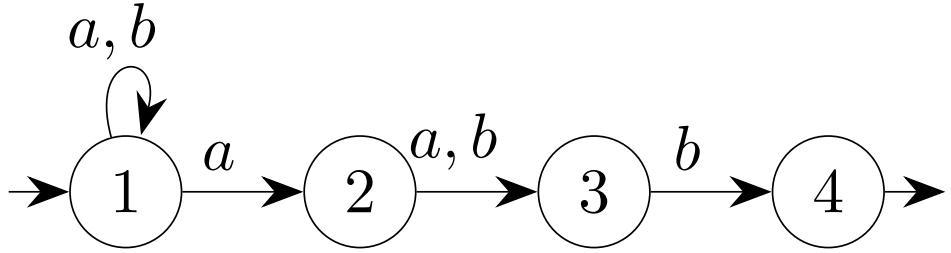
- Vgl. Vektor und Matrix im Booleschen Halbring

$$M = \begin{pmatrix} 1 & 1 & 0 & 0 \end{pmatrix}, R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}, M \circ R = \begin{pmatrix} 1 & 0 & 1 & 0 \end{pmatrix}$$

- diese Operation ist immer noch assoziativ: für Menge M, Relationen R, S: $M \circ (R \circ S) = (M \circ R) \circ S$,

  Kosten links: $|Q|^2 + |Q|^3$, rechts: $|Q|^2 + |Q|^2$.

Das Wortproblem für NFA (Algorithmus)

- Eingabe: $A = (\Sigma, Q, I, F, \delta)$, $w = [c_1, \ldots, c_n]$ mit $c_i \in \Sigma$,
  Ausgabe (Spezifikation): ist $w \in \text{Lang}(A)$?
  Implementierung:
  $M_0 = I, \forall k \geq 0 : M_{k+1} = M_k \circ \delta(c_k)$, **return** $(M_n \cap F) \neq \emptyset$.
- Korrektheit: Imp. ist äq. zu Def. Lang(A) ($\circ$ ist assoz.)
  $(I \circ \delta(c_1)) \cdots \circ \delta(c_n) = I \circ (\delta(c_1) \cdots \delta(c_n))) = I \circ \delta^*(w)$
- Laufzeit: $|w| \cdot |Q|^2$
- Bsp: $A =$ 

$, w = ab$
  $M_0 = I = \{1\}, M_1 = M_0 \circ \delta(a) = \{1\} \circ \delta(a) = \{1, 2\},$
  $M_2 = M_1 \circ \delta(b) = \{1, 2\} \circ \delta(b) = \{1, 3\}$. $M_2 \cap F = \emptyset$.

NFA zu äquivalentem DFA (Algorithmus)

- Eingabe: NFA $A = (\Sigma, Q, I, F, \delta)$, Ausgabe: DFA $A' = (\Sigma, Q', i', F', \delta')$. Spezifikation: Lang$(A) =$ Lang(A').

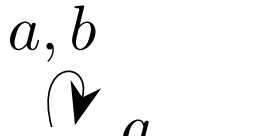
- Idee: $M \circ \delta(c)$ (wie eben) für *alle* $M \subseteq Q$ und $c \in \Sigma$.

- Implementierung: $Q' = 2^Q$ (die Potenzmenge von Q), $i = I$, $\delta'(M, c) = M \circ \delta(c)$, $F = \{M \mid M \subseteq Q \wedge M \cap F \neq \emptyset\}$

- Korrektheit: durch Induktion nach $|w|$ zeigen wir $\forall w \in \Sigma^*, M \subseteq Q : M \circ \delta^*(w) = \delta'^*(M, w)$.

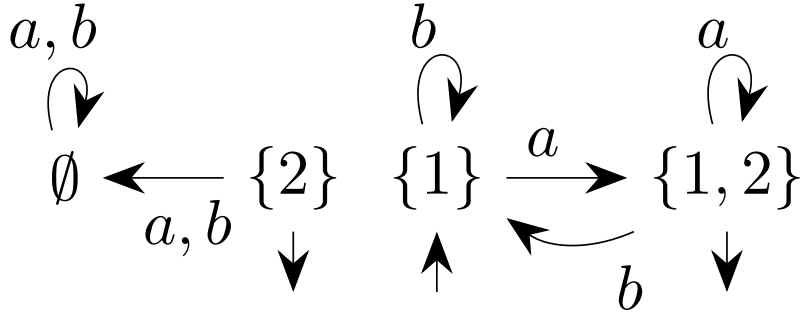
  Dann $w \in$ Lang $A \iff I \circ \delta^*(w) \cap F \neq \emptyset \iff$ $\delta'^*(I, w) \in F' \iff w \in$ Lang(A').

NFA zu äquivalentem DFA (Beispiel)

- Bsp: $A =$ 

$, Q = \{1, 2\}, I = \{1\}, F = \{2\}$
- Algorithmus: $Q' = 2^Q$, $i = I$, $\delta'(M, c) = M \circ \delta(c)$, $F = \{M \mid M \subseteq Q \wedge M \cap F \neq \emptyset\}$
- Rechnung im Beispiel:

M	$\emptyset$	$\{1\}$	$\{2\}$	$\{1,2\}$
$M \circ \delta(a)$	$\emptyset$	$\{1,2\}$	$\emptyset$	$\{1,2\}$
$M \circ \delta(b)$	$\emptyset$	$\{1\}$	$\emptyset$	$\{1\}$



- optional: nicht erreichbare Zustände $(\emptyset, \{2\})$ löschen
- besser noch: gar nicht erst erzeugen (Durchqueren des Graphen von I aus)

Vergleich von DFA und NFA

- Satz: für jede Sprache L sind diese Aussagen äquivalent:

  (1) $\exists$ DFA A mit Lang$(A) = L$,

  (2) $\exists$ NFA A mit Lang$(A) = L$,

- Beweis: $2 \Rightarrow 1$: Potenzmengenkonstruktion

  $1 \Rightarrow 2$: trivial (DFA ist NFA, denn Funktion ist Relation)

Sprach-Äquivalenz von DFA und NFA

- Def: für eine Klasse C von Automaten:
  Das Sprach-Äquivalenz-Problem für C ist:
  Eingabe: $A_1, A_2 \in C$, Ausgabe: Lang$(A_1) =$ Lang(A_2)?
- Für $C =$ DFA: konstruiere Kreuzprodukt-DFA A für symmetrische Differenz von Lang(A_1) und Lang(A_2), prüfe Lang$(A) = \emptyset$.
- für $C =$ NFA: konvertiere A_1, A_2 (einzeln) zu äquivalenten DFA, dann wie oben.
- Das ist korrekt, aber teuer (Potenzmengen), deswegen in autotool vorher:
  für die 50 (?) kleinsten Wörter $w \in$ Lang(A_1) testen, ob $w \in$ Lang(A_2), und umgekehrt.

Ausblick: Vergleich von (DFA,) NFA und REG

- Wir wollen zeigen: für jede Sprache L gilt $(1) \iff (2)$ für
  (1) $\exists X \in \mathsf{REG} : \mathsf{Lang}(X) = L$
  (2) $\exists A \in \mathsf{NFA} : \mathsf{Lang}(A) = L$

- Beweisplan für $(1) \Rightarrow (2)$: strukturelle Induktion über X
  - Anfang: $\emptyset, \{\epsilon\}, \{[c]\}$ sind NFA-Sprachen.
  - Schritt: NFA-Sprachen sind abgeschlossen unter: Vereinigung (ja), Verkettung (?), Stern (?)

- Beweisplan für $(2) \Rightarrow (1)$: Konstruktion eines zu NFA A äquivalenten regulären Ausdrucks

- Anwendung: Entscheidungsverfahren für Sprach-Äquivalenz-Problem für reguläre Ausdrücke

Zusatz: Die Shuffle-Operation

- Def: $u \sqcup\!\sqcup v$ (der Shuffle der Wörter u und v)

  ist die Menge aller Wörter w mit:

  es gibt eine rot/blau-Färbung der Positionen von w mit: das rote Teilwort ist u, das blaue Teilwort ist u.

- Bsp: $\mathit{fbaoro} \in (\mathit{foo} \sqcup\!\sqcup \mathit{bar})$,

  Bsp: $aa \sqcup\!\sqcup bb = \{aabb, abab, abba, baab, baba, bbaa\}$.

- Def: für $L_1, L_2 \subseteq \Sigma^*$: $L_1 \sqcup\!\sqcup L_2 = \bigcup_{u \in L_1, v \in L_2} (u \sqcup\!\sqcup v)$

- Bsp: $a^* \sqcup\!\sqcup b = a^* b a^*$, $a^* \sqcup\!\sqcup b^* = \{a, b\}^*$,
  $(aa)^* \sqcup\!\sqcup (bb)^* = \{w : |w|_a \equiv 0 \pmod 2 \wedge |w|_b \equiv 0 \pmod 2\}$

Aufgaben

1. Für NFA $A = (\Sigma, Q, Q, Q, \delta)$ mit $\Sigma = \{a, b\}$, $Q = \{1, 2, 3\}$, $\delta(a) = \{(1, 2), (2, 3), (3, 1)\}, \delta(b) = \{(1, 1), (1, 3), (3, 3)\}$:

   geben Sie ein Wort $w \in \Sigma^* \setminus \text{Lang}(A)$ an.

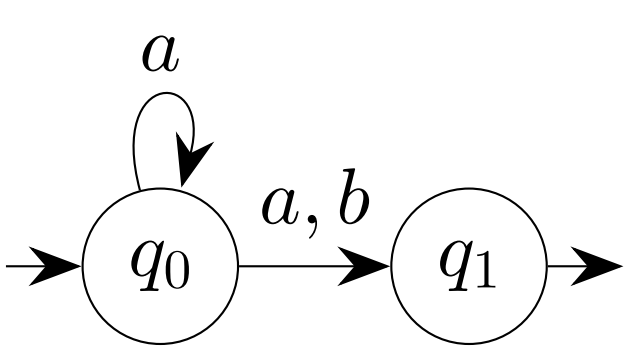
   Berechnen Sie $\delta^*(w)$ als Boolesche Matrix. Rechnen Sie effizient! Nicht von links nach rechts multiplizieren, sondern wiederholte Teilwörter ausnutzen.

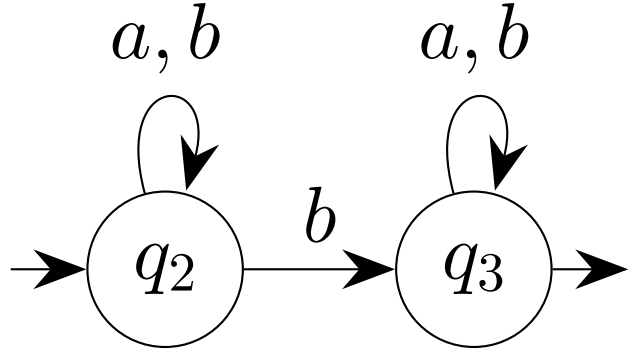
2. Für A aus der vorigen Aufgabe: führen Sie die Potenzmengenkonstruktion vor.

   Wenden Sie die Potenzmengenkonstruktion auf einen DFA an (konkretes Beispiel, allgemeine Aussage)

3. Für gegebene NFAs $A_1 = (\Sigma, Q_1, I_1, F_1, \delta_1)$, $A_2 = (\Sigma, Q_2, I_2, F_2, \delta_2)$: Konstruieren Sie den Kreuzprodukt-NFA $A = (\Sigma, Q_1 \times Q_2, I_1 \times I_2, F, \delta)$, der synchrone Rechnungen in A_1 und A_2 realisiert.

   Geben Sie δ allgemein an, illustrieren Sie durch

$A_1 =$ 

, A_2 

,

   Kann man damit durch geeignete Wahl von F ausrechnen: Vereinigung? Durchschnitt? Differenz?

   Überprüfen Sie im Beispiel. Diskutieren Sie Allgemeingültigkeit.

4. Gegeben sind NFAs $A_1 = (\Sigma, Q_1, I_1, F_1, \delta_1)$, $A_2 = (\Sigma, Q_2, I_2, F_2, \delta_2)$.

   Konstruieren Sie NFA $A = (\Sigma, Q_1 \times Q_2, ?, ?, ?)$ mit $\mathsf{Lang}(A) = \mathsf{Lang}(A_1) \sqcup\!\sqcup \mathsf{Lang}(A_2)$.

   Begründen Sie die Korrektheit mit Bezug auf die Definition (die rot/blau-Färbung der Wörter: wie sieht man die im Automaten?)

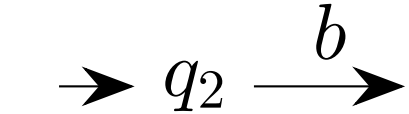
   Wenden Sie am Beispiel $(aa)^* \sqcup\!\sqcup (bb)^*$ an. Überprüfen Sie mit Autotool-Aufgabe 17-2.

   Wieviele Elemente sind in $a^i \sqcup\!\sqcup b^k$?

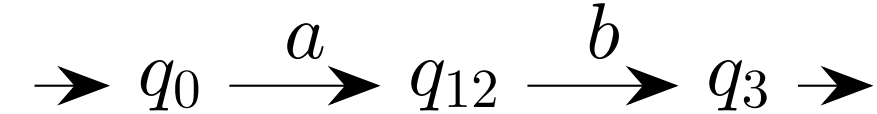
NFA und Reguläre Ausdrücke

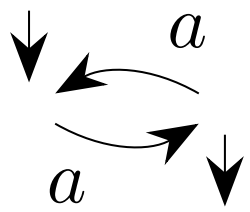
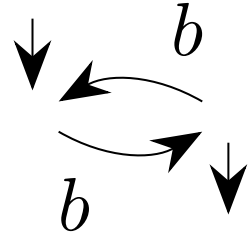
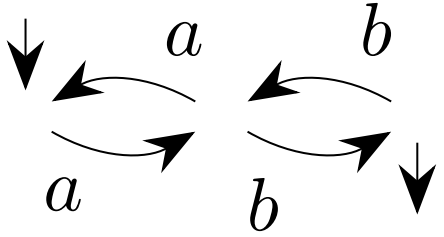
NFA mit spontanen Übergängen (Motiv.)

- Motivation: gegeben NFA A_1, A_2 über Σ. Gesucht NFA A über Σ mit $\text{Lang}(A) = \text{Lang}(A_1) \cdot \text{Lang}(A_2)$.
- im Spezialfall $|F_1| = |I_2| = 1$ (beides Einermengen)

  $A_1 =$  , $A_2 =$  ,

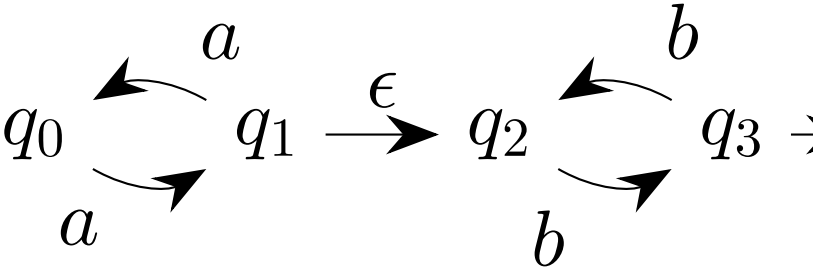
  kann man A_1 und A_2 übereinanderlegen.

  
- Ist hier korrekt, im allgemeinen aber falsch

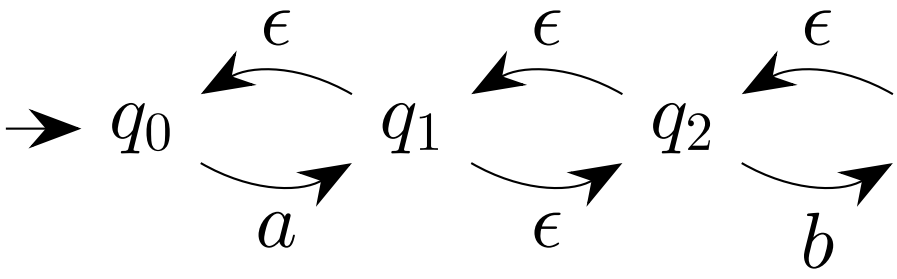
  $A_1 =$  , $A_2 =$  , $A =$ 

NFA mit spontanen Übergängen (Def. ENFA)

- Def (Syntax) ein ENFA $(\Sigma, Q, I, F, \delta, \sigma)$ (Epsilon-NFA) ist NFA $(\Sigma, Q, I, F, \delta)$ und Relation $\sigma \subseteq Q \times Q$.

- Bsp: 

- Def (Semantik) $\delta^*_\sigma(\epsilon) = \sigma^*$, $\delta^*_\sigma(c \cdot w) = \sigma^* \circ \delta(c) \circ \delta^*_\sigma(w)$.
  Bsp: $(q_0, q_2) \in \delta^*_\sigma(a)$, denn $q_0 \xrightarrow{\sigma^0} q_0 \xrightarrow{\delta(a)} q_1 \xrightarrow{\sigma^1} q_2$
  $\text{Lang}(A) = \{w \mid w \in \Sigma^* \wedge (\delta^*_\sigma(w) \cap I \times F) \neq \emptyset\}$

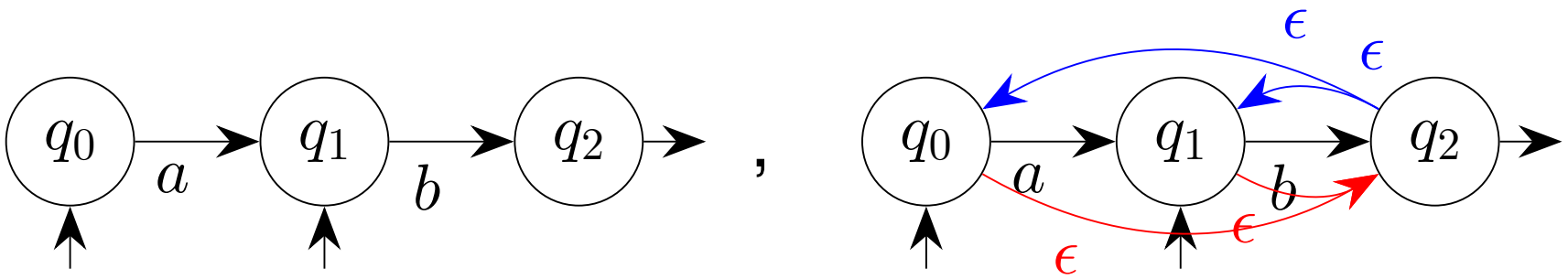
- Bsp: Lang(

Vereinigung und Verkettung für ENFA

- folgendes für zustands-disjunkte ENFA ($Q_1 \cap Q_2 = \emptyset$):
  $A_1 = (\Sigma, Q_1, I_1, F_1, \delta_1, \sigma_1)$, $A_2 = (\Sigma, Q_2, I_2, F_2, \delta_2, \sigma_2)$
- Vereinigung: $\mathsf{Lang}(A_1) \cup \mathsf{Lang}(A_2) = \mathsf{Lang}(A)$
  für $A = (\Sigma, Q_1 \cup Q_2, I_1 \cup I_2, F_1 \cup F_2, \delta_1 \cup \delta_2, \sigma_1 \cup \sigma_2)$.
  Begründung: jeder Pfad in A von $I_1 \cup I_2$ nach $F_1 \cup F_2$
  liegt ganz in A_1 oder ganz in A_2
- Verkettung: $\mathsf{Lang}(A_1) \cdot \mathsf{Lang}(A_2) = \mathsf{Lang}(A)$
  für $A = (\Sigma, Q_1 \cup Q_2, I_1, F_2, \delta_1 \cup \delta_2, \sigma_1 \cup \sigma_2 \cup (F_1 \times I_2))$.
  Begründung: jeder Pfad in A von I_1 nach F_2
  enthält genau eine der neuen ϵ-Kanten aus $F_1 \times I_2$.

Stern für ENFA? Ansatz

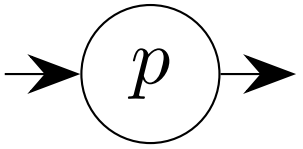
- Für $A = (\Sigma, Q, I, F, \delta, \sigma)$: mit Idee von vorher (Verkettung „mit sich selbst“): $A' = (\Sigma, Q, I, F, \delta, \sigma \cup (F \times I))$.
- $\text{Lang}(A') = \text{Lang}(A)^+$, Begründung: zerteile A'-Pfad $I \ni i \xrightarrow{\delta^*_{\sigma'}} f \in F$ bei jeder blauen Kante. Jedes Bruchstück (es gibt wenigstens eines) ist $\in \text{Lang}(A)$.
- Für $\text{Lang}(A)^* = \text{Lang}(A)^+ \cup \{\epsilon\}$ probieren wir $(\Sigma, Q, I, F, \delta, \sigma \cup (F \times I) \cup (I \times F))$.



ist falsch ($I \ni q_0 \xrightarrow{a} q_1 \xrightarrow{\epsilon} q_2 \in F$, aber $a \notin \text{Lang}(A)^*$)

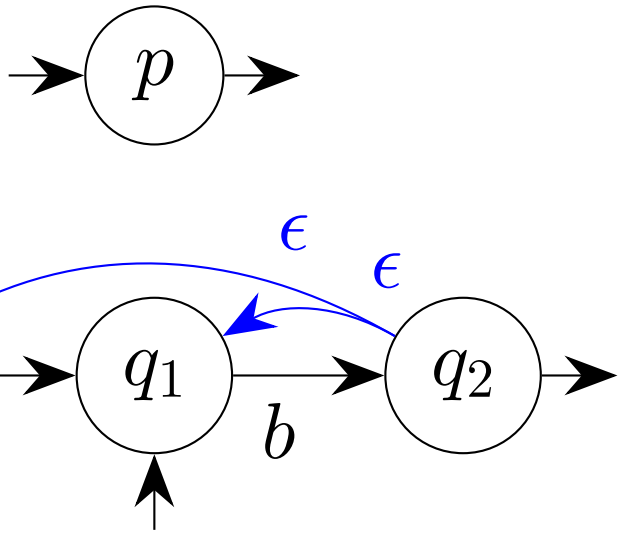
Stern für ENFA: Lösung

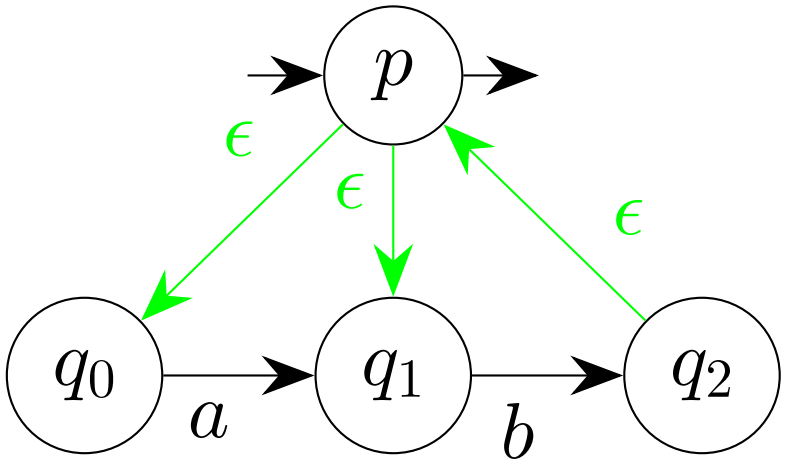
- Für $A = (\Sigma, Q, I, F, \delta, \sigma)$: baue ENFA für Lang$(A)^*$ als ...
- disjunkte ($p \notin Q$) Vereinigung

  von 

  und $A' = (\ldots, \sigma \cup (F \times I))$,

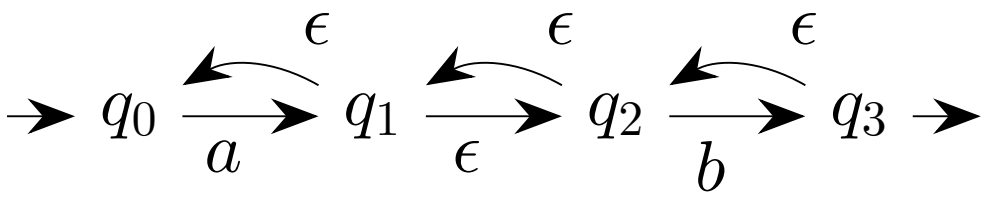
  also $(\Sigma, Q \cup \{p\}, I \cup \{p\}, F \cup \{p\}, \delta, \sigma \cup (F \times I))$
- oder $(\Sigma, Q \cup \{p\}, \{p\}, \{p\}, \delta, \sigma \cup (\{p\} \times I) \cup (F \times \{p\}))$



,


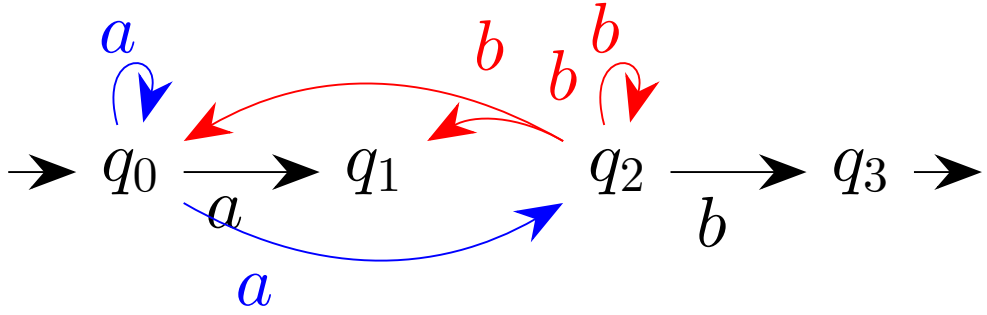
Von ENFA zu NFA (Epsilons entfernen)

- Gegeben ENFA $A = (\Sigma, Q, I, F, \delta, \sigma)$,

  Bsp: 

  gesucht NFA $A' = (\Sigma, Q, I', F', \delta')$: Lang(A) = Lang(A').

- Lösung: (multipliziere I und δ rechts mit σ^*)
  $I' = I \circ \sigma^*$, $F' = F$, $\forall c \in \Sigma : \delta'(c) = \delta(c) \circ \sigma^*$.
  Bsp: $\delta(a) = \{(q_0, q_1)\}$, $q_1 \circ \sigma^* = \{q_0, q_1, q_2\}$,
  $\delta'(a) = \{(q_0, q_0), (q_0, q_1), (q_0, q_2)\}$



- Beweis: $I' \circ \delta'^*(w) = (I \circ \sigma^*) \circ (\delta(w_1) \circ \sigma^* \ldots) = I \circ (\delta^*_\sigma(w))$

Vergleich von DFA, NFA, ENFA

- Satz: für jede Sprache L sind diese Aussagen äquivalent:

  (1) $\exists$ DFA A mit Lang$(A) = L$,

  (2) $\exists$ NFA A mit Lang$(A) = L$,

  (3) $\exists$ ENFA A mit Lang$(A) = L$,

- Beweis: $3 \Rightarrow 2$: Epsilons entfernen

  $2 \Rightarrow 1$: Potenzmengenkonstruktion

  $1 \Rightarrow 2$: trivial (DFA ist NFA, denn Funktion ist Relation)

  $2 \Rightarrow 3$: trivial (NFA ist ENFA, $\sigma = \emptyset$)

- Folgerung: Sprach-Äquivalenz ist entscheidbar für ENFA, NFA, DFA.

  Bew.: $(3) \Rightarrow (2) \Rightarrow (1)$, für DFA: sym. Diff. (Kreuzprodukt),

Vergleich von NFA und REG (Ansatz)

- Satz: für jede Sprache L gilt $(1) \Rightarrow (2)$ für

  (1) $\exists X \in \mathsf{REG} : \mathsf{Lang}(X) = L$

  (2) $\exists A \in \mathsf{NFA} : \mathsf{Lang}(A) = L$

- Beweis: (strukturelle Induktion über X)
  - Anfang: $\emptyset, \{\epsilon\}, \{[c]\}$ sind NFA-Sprachen.
  - Schritt: NFA-Sprachen sind abgeschlossen unter: Vereinigung, Verkettung, Stern (Umweg über ENFA).

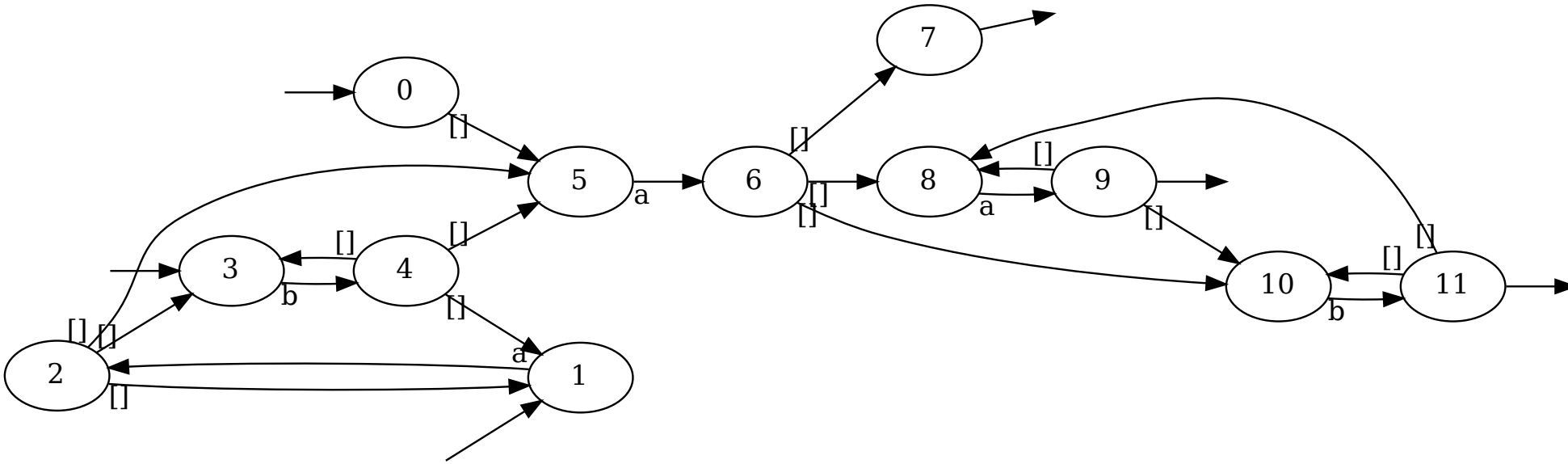
- es gibt für $(1) \Rightarrow (2)$ andere Konstruktionen (mit weniger Hilfszuständen) (V. M. Glushkov 1961, Taxonomie: B. W. Watson 1993)

- es gilt auch $(2) \Rightarrow (1)$ ... folgt

REG zu NFA (Beispiel)

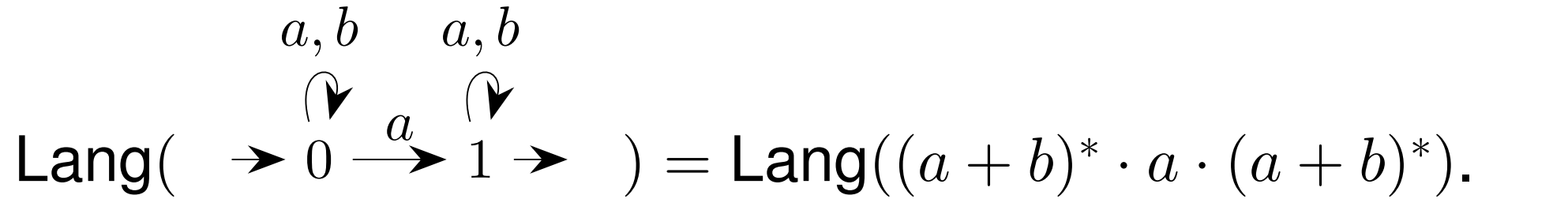
- nach vorigem Verfahren konstruierter ENFA

  für $(a+b)^*a(a+b)^*$

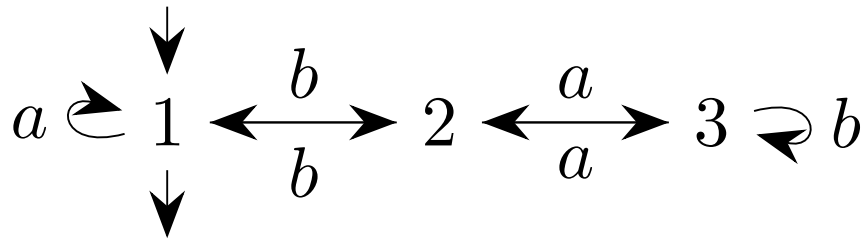


NFA zu REG: Motivation

- in einfachen Fällen haben NFA und dazu äquvalenter regulärer Ausdruck ähnliche Struktur:



- . . . aber nicht alle NFA sind so einfach, auch DFA nicht:



(durch 3 teilbare Binärzahlen)

NFA zu REG: Algorithmus

- für $Q = \{1, 2, \ldots, n\}$, $p, q \in Q, h \in \{0, 1, \ldots, n\}$

  $L_h(p, q) :=$
  $\{w \mid \exists p_1, \ldots, p_{k-1} \in \{1, \ldots, h\} : p \stackrel{w_1}{\to} p_1 \ldots p_{k-1} \stackrel{w_k}{\to} q\}$

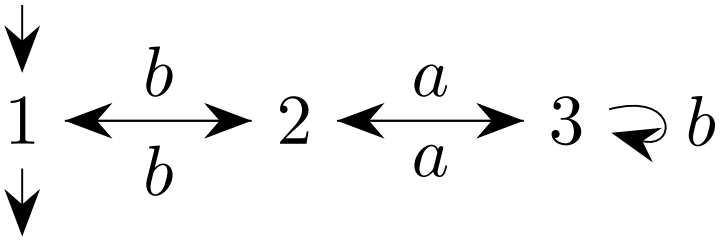
  Spezialfälle: $|w| = 0$ (dann $p = q$), $|w| = 1$ (dann $p \stackrel{w_1}{\to} q$).

- Satz: Lang$(A) = \bigcup_{p \in I, q \in F} L_n(p, q)$

- Konstruktion eines reg. Ausdrucks für $L_h(p, q)$ durch Induktion nach h

  – Start: $L_0(p, q) = \{\epsilon \mid p = q\} \cup \bigcup\{c \mid (p, q) \in \delta(c)\}$
  – Schritt: $L_{h+1}(p, q) =$
  $L_h(p, q) \cup L_h(p, h + 1) \cdot L_h(h + 1, h + 1)^* \cdot L_h(h + 1, q)$.

NFA zu REG: Beispiel

- 

- $L_0 = \begin{pmatrix} \epsilon + a & b & \emptyset \\ b & \epsilon & a \\ \emptyset & a & \epsilon + b \end{pmatrix}$, $L_1 = \begin{pmatrix} a^* & a^*b & \emptyset \\ ba^* & \epsilon + ba^*b & a \\ \emptyset & a & \epsilon + b \end{pmatrix}$,

$L_2 =$

$$\begin{pmatrix} a^* + a^*b(\epsilon + ba^*b)^*ba^* & a^*b(\epsilon + ba^*b)^* & a^*b(\epsilon + ba^*b)^*a \\ (\epsilon + ba^*b)^*ba^* & (\epsilon + ba^*b)^* & (\epsilon + ba^*b)^*a \\ a(\epsilon + ba^*b)^*ba^* & a(\epsilon + b^*ab)^* & \epsilon + b + a(\epsilon + ba^*b)^* \end{pmatrix}$$

$L_3(1,1) = L_2(1,1) \cup L_2(1,3) \cdot L_2(3,3)^* \cdot L_2(3,1) =$

$(a^* + a^*b(ba^*b)^*ba^*) + a^*b(ba^*b)^*a \cdot (b + a(ba^*b)^*a)^* \cdot a(ba^*b)^*ba^*$

Vergleich von NFA und REG (Resultat)

- Satz: für jede Sprache L gilt $(1) \Leftrightarrow (2)$ für

  (1) $\exists X \in \mathsf{REG} : \mathsf{Lang}(X) = L$

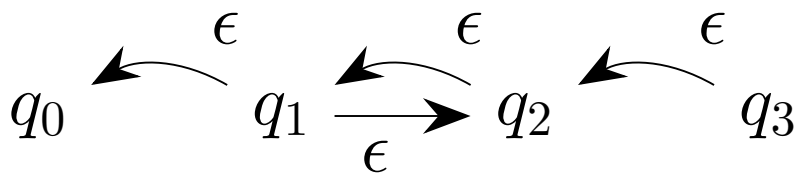
  (2) $\exists A \in \mathsf{NFA} : \mathsf{Lang}(A) = L$

- Beweis
  - $(1) \Rightarrow (2)$: strukturelle Induktion über X
  - $(2) \Rightarrow (1)$: Konstruktion von $L_h(p, q)$ durch Induktion über $h \in \{0, 1, \ldots, |Q|\}$

Aufgaben

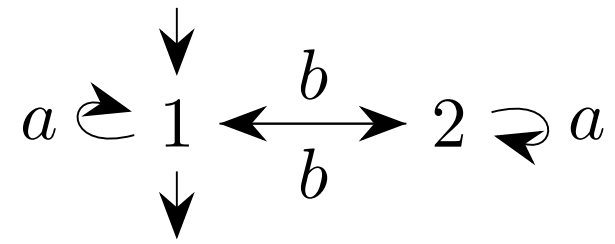
1. drei Algorithmen zur Bestimmung von σ^*.

   Führen Sie jeweils vor für diesen Graphen und diskutieren Sie Korrektheit und Kosten (abhängig von $|Q|$) im allgemeinen.



   (a) Berechne $S_0 = \mathsf{Id}_Q$, $S_{k+1} = S_k \cup S_k \circ \sigma$, bis $S_{k+1} = S_k$. Wieviele Schritte höchstens bis dahin?
   (b) Berechne $T_0 = \mathsf{Id}_Q$, $T_{k+1} = T_k^2 \cup \sigma$. Wieviele Schritte höchstens? (Beziehung zw. T_i und passendem S_k?)
   (c) Für jedes $q \in Q$ eine Tiefensuche im Graphen σ, um $\sigma^*(q)$ zu bestimmen.

2. Konstruieren Sie einen äquivalenten regulären Ausdruck

für  nach angegebenem Verfahren

- mit Zustands-Reihenfolge $1 < 2$,
- mit Zustands-Reihenfolge $2 < 1$.

Durch welche Umformungsregeln kann die Sprach-Äquivalenz der Ausdrücke bewiesen werden?

3. Zusatz: Auf Folie *NFA zu REG: Beispiel* wurden Vereinfachungen benutzt, z.B.
$L_2(1,2) = L_1(1,2) \cup L_1(1,2)L_1(2,2)^*L_1(2,2) =$
$L_1(1,2) \cdot (\epsilon + L_1(2,2)^+) = L_1(1,2)L_1(2,2)^*$, welche noch? verallgemeinern Sie das zu einer (in einigen Fällen)

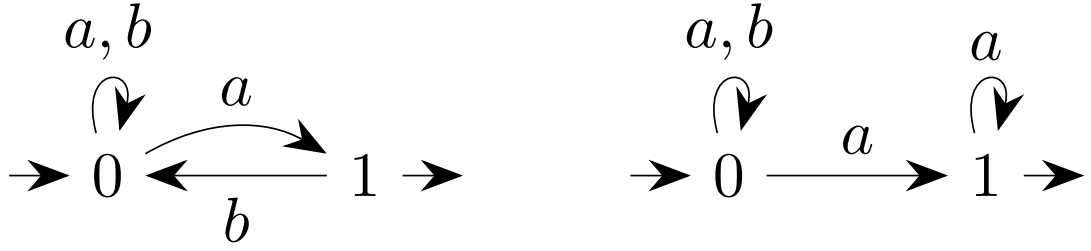
verbesserten Vorschrift zur Berechnung von $L_{h+1}(p, q)$.

Ergänzen Sie die Implementierung (im Quelltext-Repo zur Vorlesung), testen Sie Korrekheit, bewerten Sie die erzielte Einsparung.

Minimale DFA

Motivation

- für jedes $L \in \mathrm{Lang}(\mathrm{REG})$ gibt es (unendlich) viele $A \in \mathrm{NFA}$ mit $L = \mathrm{Lang}(A)$. gibt es *einen* kleinsten? Ja:
- für n in $1, 2, \ldots$ // bis wohin?
  für alle $A \in \mathrm{NFA}(\Sigma)$ mit $Q(A) = \{1, \ldots, n\}$ // wie viele?
  wenn $L = \mathrm{Lang}(A)$, // wie entscheiden?
  dann **return** A
- gibt es *den* (d.h., genau einen) kleinsten?
  - für NFA: nein,



  - für DFA: ja, und bessere Konstruktion

Die Nerode-Kongruenz einer Sprache

- Def: $u \sim_L v \iff \forall w \in \Sigma^* : (u \cdot w \in L \iff v \cdot w \in L)$
  benannt nach Anil Nerode (*Linear Automaton Transformations* Proc. AMS 1958)
- Bsp: $\Sigma = \{a, b\}, L = \{a^i b^k \mid i \geq k \geq 0\}$
  - $aa \not\sim_L bab$, denn für $w = \epsilon$ ist
    $u \cdot w = aa \cdot \epsilon = aa \in L, v \cdot w = bab \cdot \epsilon = bab \notin L$.
  - $a^2 b \sim_L a^4 b^3$.
  - $a^0, a^1, a^2, \ldots$ sind paarweise nicht $\sim_L$, denn . . . (Ü)
- $\forall L \subseteq \Sigma^*$: $\sim_L$ ist Äquivalenz-Relation auf Σ^*.
- die Äquivalenz-Klasse von u ist $[u]_{\sim_L} = \{v \mid u \sim_L v\}$.
  Bsp: $[a^2 b]_{\sim_L} = \{a^2 b, a^3 b^2, a^4 b^3, \ldots\} = \{a^{k+1} b^k \mid k > 0\}$
- die Menge aller Äquivalenzklassen ist $\Sigma^* / \sim_L$.

Eigenschaften der Nerode-Kongruenz

- Satz: für alle $u, v, s \in \Sigma^* : u \sim_L v \Rightarrow (u \cdot s) \sim_L (v \cdot s)$.

  Beweis: $(us)w \in L \iff u(sw) \in L \iff v(sw) \in L \iff (vs)w \in L$.

  Die Rechts-Multiplikation (Äq.-Klasse mal Wort) ist unabhängig von der Wahl des Vertreters

- Satz: $u \in L \Rightarrow [u]_{\sim_L} \subseteq L$,

  Beweis: $v \in [u]_{\sim_L} \Rightarrow u \sim_L v \Rightarrow u \cdot \epsilon \in L \iff v \cdot \epsilon \in L$.

- Satz: $u \notin L \Rightarrow [u]_{\sim_L} \cap L = \emptyset$

  Beweis: $v \in [u]_{\sim_L} \Rightarrow u \sim_L v \Rightarrow u \cdot \epsilon \notin L \iff v \cdot \epsilon \notin L$.

- jede $\sim_L$-Äquivalenzklasse liegt entweder ganz in L oder ganz außerhalb.

Der Äquivalenzklassen-Automat

- Def: der Nerode-Automat für $L \subseteq \Sigma^*$ ist der D(F?)A
  $A = (\Sigma, \Sigma^*/\sim_L, [\epsilon]_{\sim_L}, F, \delta)$
  mit $\delta([w]_{\sim_L}, c) = [w \cdot c]_{\sim_L}$ und $F = \{[w]_{\sim_L} \mid [w]_{\sim_L} \subseteq L\}$.
- Satz: Lang$(A) = L$. Beweis:
  1. $\delta^*(i, w) = [w]_{\sim_L}$ durch Induktion nach $|w|$.
  2. $w \in L \iff [w]_{\sim_L} \subseteq L$ nach Konstruktion und Satz
- Satz: wenn $\Sigma^*/\sim_L$ endlich, dann L regulär.
  Beweis: der Nerode-Automat von L ist ein DFA für L.
- Bsp: $L = (ab)^*$ über $\Sigma = \{a, b\}$. $\Sigma^*/\sim_L = \{[\epsilon], [a], [b]\}$.
- als nächstes: der Nerode-A. von L ist *der minimale* DFA für L und kann aus *jedem* DFA für L konstruiert werden.

Nerode-Kongruenz und DFA

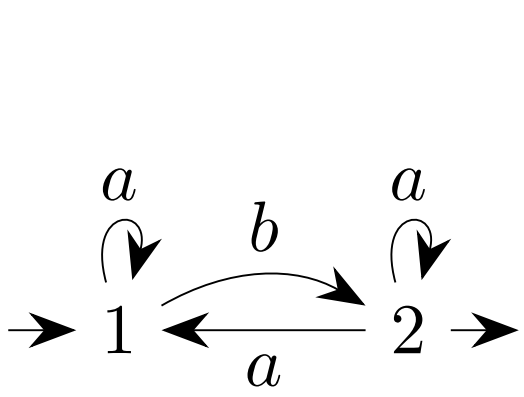
- Für DFA $A = (\Sigma, Q, i, F, \delta)$ mit $L = \mathsf{Lang}(A)$ und $q \in Q$ definieren wir $L_A(q) = \mathsf{Lang}(\Sigma, Q, i, \{q\}, \delta)$
  (die Sprache aller Pfade von i nach q)
- Satz: $\forall A \in \mathsf{DFA}, q \in Q(A), u, v \in L_A(q) : u \sim_L v$
  Beweis: $u \cdot w \in L \iff \delta^*(i, u \cdot w) \in F \iff$
  $\delta^*(\delta^*(i, u), w) \in F \iff \delta^*(q, w) \in F \iff$
  $\delta^*(\delta^*(i, v), w) \in F \iff \delta^*(i, v \cdot w) \in F \iff v \cdot w \in L$
- Satz: $|Q(A)| \geq |\Sigma^* / \sim_L|$,
  Beweis: für jede Klasse $C_1, C_2, \cdots \in \Sigma^* / \sim_L$
  betrachte ein kürzestes Wort $w_k \in C_k$ und $q_k = \delta(i, w_k)$.
  Diese $q_1, q_2, \ldots \in Q$ sind alle verschieden.
- Folgerung: der Nerode-A. von L ist der kleinste DFA für L
- Folgerung: wenn L regulär, dann $\Sigma^* / \sim_L$ endlich

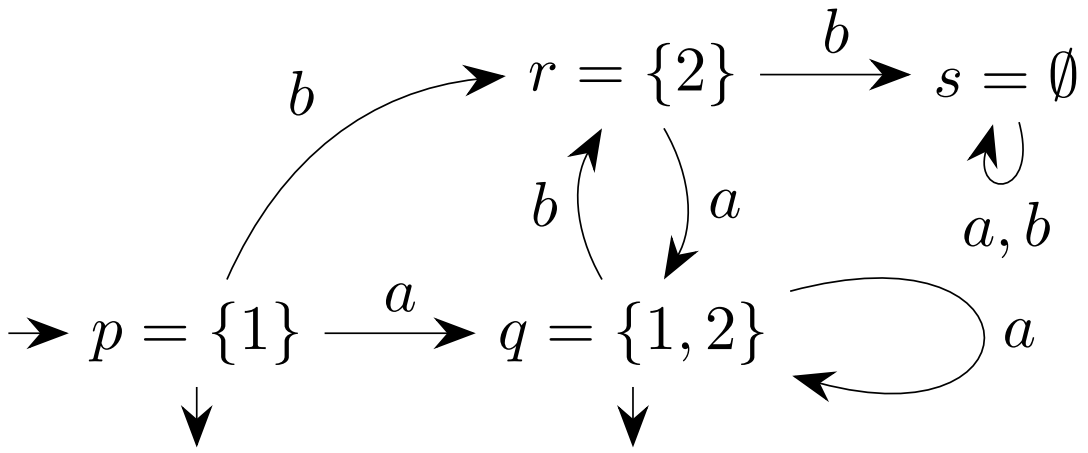
Konstruktion des minimalen DFA

- Aufgabe: für $A \in \mathrm{DFA}(\Sigma)$ mit $L = \mathrm{Lang}(A)$: bestimme die Zustände, die $\sim_L$-äquivalent sind,
- Lösung: Folge $R_0, \ldots, R_k$ von Äquivalenz-Rel. auf Q mit
  - Invariante: $R_i(p,q) \iff \forall u \in L_A(p), v \in L_A(q), w \in \Sigma^{\leq i} : (u \cdot w \in L \iff v \cdot w \in L)$.
  - Start: $R_0(p,q) = (p \in F \iff q \in F)$
  - Schr.: $R_{i+1}(p,q) = (R_i(p,q) \wedge \forall c \in \Sigma : R_i(\delta(p,c), \delta(q,c)))$
  - Schluß: wenn $R_{k+1} = R_k$. (Inv. gilt auch für alle $k' \geq k$)
- Satz: Korrektheit. Bew.: $\forall u \in L_A(p), v \in L_A(q), w \in \Sigma^*$: $(u \cdot w \in L \iff v \cdot w \in L)$, benutze Inv. für $k' = |w|$
- Satz: Termination. Beweis: schwach monoton fallende Folge endlicher Mengen $R_0 \supseteq \cdots \supseteq R_k = R_{k+1} = \ldots$

Konstruktion des minimalen DFA: Beispiel

- Potenzmengenkonstr.: DFA mit $Q = \{p, q, r, s\}$.



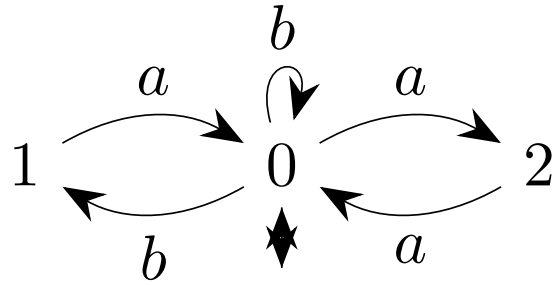


- $R_0 = \{(p,p),(p,q),(q,p),(q,q),(r,r),(r,s),(s,r),(s,s)\}$ die Äq.-Kl. Q/R_0 sind $\{F, Q \setminus F\} = \{\{p,q\},\{r,s\}\}$.
- es gilt $(r,s) \notin R_1$, denn es gilt $(\delta(r,\boxed{}), \delta(s,\boxed{})) \notin R_0$. gilt $(p,q) \in R_1$? Äq.-Klassen $Q/R_1 = \{\ldots, \{r\}, \{s\}\}$
- R_2? der DFA auf Q/R_2?

Aufgaben

1. (die einfache erste Aufgabe: Selbst-Test, ohne Punkt)
   Kann ein DFA eine leere Zustandsmenge haben? (Nein.)
   Ein NFA? (Ja.) Was ist die Sprache des leeren NFA?
   Geben Sie die Nerode-Kongruenz dieser Sprache an.
   Wenden Sie die Potenzmengenkonstruktion auf einen leeren NFA an. Überprüfen Sie die Sprach-Äquivalenz.
   Minimieren Sie den erhaltenen DFA nach dem Verfahren aus der Vorlesung.

2. Für diesen NFA (0 ist initial und final) den erreichbaren Teil des Potenzmengenautomaten bestimmen und dann minimieren:



3. Beschreiben Sie die Nerode-Äquivalenzklassen der Sprachen

(a) $L = \{a^{2 \cdot k} \mid k \in \mathbb{N}\}$ über $\Sigma = \{a\}$
(b) $L = \{a^{k^2} \mid k \in \mathbb{N}\}$ über $\Sigma = \{a\}$
(c) $L = \{a^i b^j \mid i \neq j\}$ über $\Sigma = \{a, b\}$

Geben Sie Beispiele (falls existieren) für

- $u \sim_L v$, $u \not\sim_L v$;
- für eine endliche $\sim_L$-Klasse, für eine unendliche;
- Gibt es unendlich viele $\sim_L$-Klassen?

Zusatz: eine Sprache L über Alphabet $\{a, b\}$ angeben, für die jede $\sim_L$-Klasse endlich ist.

4. Zusatz: Führen Sie die folgende Konstruktion für einen

kleinen NFA A (eines der Beispiele von den Folien oder eine Instanz Ihrer autotool-Aufgabe) vor:

$$A = A_0 \overset{\text{Spiegelbild}}{\longrightarrow} A_1 \overset{\text{Err. Pot.}}{\longrightarrow} A_2 \overset{\text{Spiegelbild}}{\longrightarrow} A_3 \overset{\text{Err. Pot.}}{\longrightarrow} A_4$$

(Err.Pot.: der erreichbare Teil d. Potenzmengenaut.)

Das Spiegelbild A^R eines NFA A erhält man durch Umkehren aller Pfeile und Vertauschen von I mit F. Damit gilt $\text{Lang}(A^R) = \text{Lang}(A)^R$, wobei das Spiegelbild einer Sprache $L^R = \{w^R \mid w \in L\}$ und das Spiegelbild eines Wortes $[w_1, w_2, \ldots, w_n]^R = [w_n, \ldots, w_2, w_1]$.

Begründen Sie $\text{Lang}(A_0) = \text{Lang}(A_4)$.

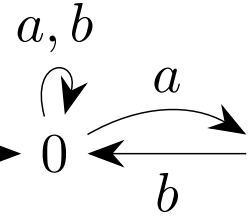
Nach Konstruktion ist A_4 ein DFA. Ein Satz von Brzozowski besagt: A_4 ist sogar minimal. Überprüfen Sie an Ihrem Beispiel.

Schleifensatz zum Nachweis der Nicht-Regularität

Motivation

- L regulär $\iff$ $\Sigma^*/\sim_L$ endlich. damit kann $L \notin \mathrm{Lang}(\mathrm{REG})$ nachgewiesen werden, aber: Untersuchung der $\sim_L$-Klassen ist evtl. aufwendig
- eine Abkürzung bringen Abschluß-Eigenschaften, Bsp. Dyck-Sprache D (Klammerwörter), $D \cap a^*b^* = \{a^k b^k \mid k \geq 0\} \notin \mathrm{Lang}(\mathrm{REG})$.
- jetzt: ein anderes Verfahren zum Nachweis von $L \notin \mathrm{Lang}(\mathrm{REG})$ in *einigen* Fällen (nicht in allen, die Aussage ist keine Äquivalenz)

Schleifensatz: Idee

- wenn $L = \text{Lang}(A)$ mit $A \in \text{NFA}$,
  dann für jedes Wort $w \in L$ mit $|w| \geq |Q(A)|$:
  jeder akzeptierende w-Pfad in A (von $i \in I$ zu $f \in F$)
  enthält einen wiederholten Zustand, also eine Schleife,
- diese Schleife kann beliebig oft durchlaufen werden
  (0 mal, 1 mal, 2 mal, 3 mal, ...)
  dadurch entstehen weitere akzeptierende Pfade
  deswegen sind dann noch weitere Wörter
  (für: 0 mal, 2 mal, 3 mal, ...) in L.
- $A =$ 

, $|Q(A)| = 2$, $w = aba$, $|w| \geq 2$,
  $I \ni 0 \xrightarrow{a} 1 \xrightarrow{b} 0 \xrightarrow{a} 1 \in F$, $I \ni (0 \xrightarrow{a} 1 \xrightarrow{b})^k 0 \xrightarrow{a} 1 \in F$,

Der Schleifensatz: Aussage und Beweis

- Satz: $\forall L \in \mathsf{Lang}(\mathsf{NFA}) : \exists n : \forall w \in L : |w| \geq n \Rightarrow \exists r, s, t \in \Sigma^* : w = r \cdot s \cdot t \wedge |r \cdot s| \leq n \wedge 1 \leq |s| \wedge \forall k \geq 0 : r \cdot s^k \cdot t \in L$.
- Beweis: $A = (\Sigma, Q, I, F, \delta) \in \mathsf{NFA}$ mit $L = \mathsf{Lang}(A)$,
  wähle $n = |Q|$. Für jedes $w \in L$ mit $|w| = l \geq n$:
  betrachte einen w-Pfad $I \ni q_0 \xrightarrow{w_1} q_1 \ldots q_{l-1} \xrightarrow{w_l} q_l \in F$.
  der Anfangs-Abschnitt $q_0, \ldots, q_n$ (der Länge $n+1$)
  enthält wenigstens ein Duplikat ($0 \leq i < j \leq n$ mit $q_i = q_j$)
  (Schubfachschluß, es gibt nur n verschiedene Zust.).
  dann $r = w[1 \ldots i], s = w[i+1 \ldots j], t = w[j+1 \ldots l]$.
  Dann ist für jedes $k \geq 0$ auch $rs^kt \in \mathsf{Lang}(A)$,
  denn $I \ni q_0 \xrightarrow{r} (q_i \xrightarrow{s} q_j)^k \xrightarrow{t} q_l \in F$
- englisch: *pumping lemma*, (auf- und ab-pumpen!)
- Def: $\mathsf{Pump}(L, n)$, falls $\forall w \in L : |w| \geq n \Rightarrow \ldots$

Der Schleifensatz: Beispiele

- $L = \{w : |w|_b \equiv 0 \pmod 2\}$. Es gilt $\mathsf{Pump}(L, 2)$.

  Beispiel: für $w = abab \in L$

  wähle $r = \epsilon, s = a, t = bab$, es gilt $rs^*t = \epsilon \cdot a^* \cdot bab \subseteq L$.

  Beispiel: für $w = bb \in L$

  wähle $r = \epsilon, s =, t = \epsilon$, es gilt $rs^*t = \epsilon \cdot (bb)^* \cdot \epsilon \subseteq L$.

- $D =$ Dyck-Sprache (Klammerwörter). Gilt $\mathsf{Pump}(D, 2)$?

  Jedes $w \in D$ mit $|w| \geq 2$ enthält ein Teilwort $s = ab$ (ein inneres Klammerpaar).

  Dann $w = r \cdot ab \cdot t$ und $rs^*t = r(ab)^*t \subseteq D$.

  Ja, aber: die Bedingung $|rs| \leq 2$ gilt nicht immer!

Pumpen im Weltraum: die Shadoks

- Satz: $\forall L \in \text{Lang}(\text{NFA}) : \exists n : \forall w \in L : |w| \geq n \Rightarrow \exists r, s, t \in \Sigma^* : w = r \cdot s \cdot t \wedge |r \cdot s| \leq n \wedge 1 \leq |s| \wedge \forall k \geq 0 : r \cdot s^k \cdot t \in L$.
- englisch: *pumping lemma*,
  aufpumpen: $k > 1$, abpumpen: $k = 0$.
- Def: $\text{Pump}(L, n)$, falls $\forall w \in L : |w| \geq n \Rightarrow \ldots$
  Def: $\text{Pump}(L)$, falls $\exists n \in \mathbb{N} : \text{Pump}(L, n)$.
- Jacques Rouxel: Les Shadoks (1968–1974) Die Shadoks und die Gibis wollen zur Erde, da ihre Planeten zu instabil sind. Die Gibi haben den Treibstoff Cosmogol 999. Die Shadoks wollen diesen stehlen, mit riesigen Kosmospumpen. Die Shadoks beginnen also zu pumpen.

Kontraposition des Schleifensatzes

- $L \in$ Lang(NFA) $\Rightarrow \exists n : \forall w : (w \in L \wedge |w| \geq n) \Rightarrow \exists r, s, t : \cdots \wedge \forall k : \ldots$ ist äquivalent zu seiner Kontraposition:
- $\neg(\exists n : \forall w : (w \in L \wedge |w| \geq n) \Rightarrow \exists r, s, t : \cdots \wedge \forall k : \ldots) \Rightarrow L \notin$ Lang(NFA).
- deren Voraussetzung ist äquivalent zu
  $\forall n : \exists w : \neg(w \in L \wedge |w| \geq n) \Rightarrow \exists r, s, t : \cdots \wedge \forall k : \ldots)$
  äq. $\forall n : \exists w : (w \in L \wedge |w| \geq n) \wedge \forall r, s, t : \neg \cdots \vee \forall k : \ldots)$
- für die Dyck-Sprache, und beliebiges n: wähle $w = a^n b^n$.
  Für beliebige Zerlegung $w = rst$ mit $|rs| \leq n$ und $|s| \geq 1$ gilt $s \in a^+$, wähle $k = 0$, dann $rs^k t \notin D$.
  Daraus folgt $D \notin$ Lang(NFA). (Das wußten wir schon, aber mit Schleifensatz brauchen wir $\sim_D$ nicht mehr.)

Schleifensatz ist keine Äquivalenz

- für $L = \{a^i b^j c^k \mid i = 0 \vee j = k\} = b^* c^* \cup a^+\{b^j c^j \mid j \geq 0\}$.

  gilt $\mathsf{Pump}(L, 1)$ (die Folgerung des Schleifensatzes)

  – falls $w \in b^* c^*$ und $|w| \geq 1$, dann $r = \epsilon$, $s =$ erster Buchstabe
  – falls $w \in a^+\{b^j c^j \mid j \geq 0\}$, dann $r = \epsilon, s = a$.

- die Voraussetzung des Schleifensatzes gilt aber nicht: L ist nicht regulär,

  Beweis: $\{ab^j \mid j \geq 0\}$ sind paarweise nicht äquivalent:

  für $j \neq j'$: $ab^j \cdot c^j \in L$, $ab^{j'} \cdot c^j \notin L$

Aufgaben

1. Geben Sie ein Beispiel an für reduzierte NFA A_1, A_2 mit 3 Zuständen und Lang(A_1) endlich, Lang(A_2) unendlich.

   Beweisen Sie: wenn ein NFA $A = (\Sigma, Q, I, F, \delta)$ mit n Zuständen ein Wort der Länge $\geq n$ akzeptiert, dann ist Lang(A) unendlich. (Hinweis: Schleifensatz, aufpumpen)

   Geben Sie ein Verfahren an, mit dem diese Eigenschaft effizient überprüft werden kann. Nehmen Sie dazu an, daß A reduziert ist. Auf welche Eigenschaft der Vereinigung der Übergangsrelationen $R = \bigcup_{a \in \Sigma} \delta(a)$ kommt es an? Wo wird die Reduziertheit benutzt?

2. Zeigen Sie, daß folgende Sprachen L die Eigenschaft $\exists n$: Pump(L, n) nicht erfüllen:

(a) $\{a^k b^{2k} \mid k \in \mathbb{N}\}$
(b) $\{a^k b a^k \mid k \in \mathbb{N}\}$
(c) $\{a^x b^y c^{x+y} \mid x, y \in \mathbb{N}\}$

3. Für die Sprache $L = \{a^x b^y c^{x+y} \mid x, y \in \mathbb{N}\}$: sind die folgenden Sprachen regulär?

(a) $L \cap a^* b^*$
(b) $L \cap b^* c^*$

Abschluß von REG unter Morphismen

Wdhlg/Motivation Abschluß-Eigenschaften

- Satz: $L = \{a^i b^j c^k \mid i = 0 \vee j = k\} \notin$ REG
  Beweis (indirekt): falls doch, dann
  $L' = L \cap ab^*c^* = \{a^1 b^k c^k \mid k \in \mathbb{N}\} \in$ REG
  aber: es gibt kein n mit $\mathsf{Pump}(L', n)$.
  Beweis dafür (indirekt): falls doch, für n, dann wähle
  $w = ab^n c^n$, für jede Zerlegung $w = rst$ mit $|rs| \leq n, |s| \geq 1$
  wähle $k = 0$
- für solche Beweise sind Abschluß-Eigenschaften von REG nützlich. Wir kennen schon: Abschluß unter $\cup, \cap, \cdot, ^*$, wir werden jetzt das a entfernen aus $\{a^1 b^k c^k \mid k \in \mathbb{N}\}$, entsteht $\{b^k c^k \mid k \in \mathbb{N}\} \notin$ REG.

Morphismen von Sprachen

- Def: ein Morphismus ϕ ordnet jedem Buchstaben $a \in \Sigma$ eine Wort $\phi(a) \in \Gamma^*$ zu
  und wird fortgesetzt auf Wörter durch Multiplikation:
  $\phi(\epsilon) = \epsilon, \phi(w_1 \cdot \ldots \cdot w_n) = \phi(w_1) \cdot \ldots \cdot \phi(w_n)$
  und auf Sprachen durch $\phi(L) = \{\phi(w) \mid w \in L\}$
- Bsp: $\Sigma = \{a, b, c\}, \Gamma = \{0, 1\}$,
  $\phi(a) = 01, \phi(b) = 1, \phi(c) = 0$
  $\phi(bab) = 1011$, $\phi((ca)^*) = (001)^*$
- jede solche Abb. $\phi : \Sigma^* \to \Gamma^*$ ist ein Homomorphismus ($=$ strukturerhaltende Abbildung) vom Monoid $(\Sigma^*, \epsilon, \cdot)$ in das Monoid $(\Gamma^*, \epsilon, \cdot)$, das bedeutet:
  - $\phi(\epsilon) = \epsilon$
  - $\forall u, v : \phi(u \cdot v) = \phi(u) \cdot \phi(v)$

Abschluß von REG unter Morphismem

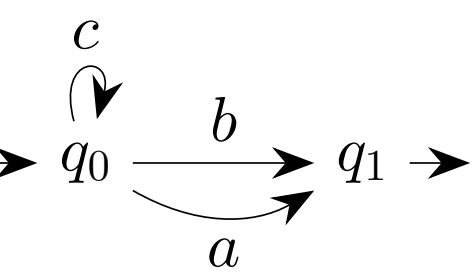
- Satz: wenn ϕ Morph. und $L \in$ REG, dann $\phi(L) \in$ REG.
- Beweis: wenn L regulär, dann gibt es reg. Ausdr. X mit Lang$(X) = L$.

  Ersetze in X jedes Vorkommen eines Buchstaben a durch das Wort $\phi(a)$, erhalte reg. Ausdr. X' mit Lang$(X') = \phi(L)$.
- alternativer Beweis: . . . NFA A mit Lang$(A) = L$, dann ersetze jede Kante $\stackrel{a}{\rightarrow}$ durch Pfad (mit evtl. neuen Zuständen) mit Beschriftung $\phi(a)$.

  Vorsicht: welche Art Automat entsteht dadurch?
- Anwendung: $\phi : a \mapsto \epsilon, b \mapsto a, c \mapsto b$, dann $\phi(\{ab^n c^n \mid n \in \mathbb{N}\}) = \{a^n b^n \mid n \in \mathbb{N}\} \notin$ REG.

Zus.: Abschluß unter inversen Morphismem

- Def: für Morph. ϕ, Sprache L: $\phi^{-1}(L) := \{w \mid \phi(w) \in L\}$
- Bsp: $\Sigma = \{a, b, c\}, \Gamma = \{0, 1\}, \phi(a) = 01, \phi(b) = 1, \phi(c) = 0$
  $\phi^{-1}(01) = \{a, cb\}$, $\phi^{-1}(0^*1) = c^*(a + b)$
- Satz: wenn ϕ Morph. und $L \in$ REG, dann $\phi^{-1}(L) \in$ REG.
- Bew: aus NFA $A = (\Gamma, Q, I, F, \delta_A)$ für L konstr. NFA
  $B = (\Sigma, Q, I, F, \delta_B)$ mit $\delta_B(a) = \delta_A^*(\phi(a))$. Dann
  $\phi^{-1}(L) =$ Lang(B).
- Bsp: $A =$ 

, $B =$ 

  $\delta_B(a) = \delta_A^*(01) = \delta_A(0) \circ \delta_A(1) = \{(q_0, q_1)\}$

Aufgaben

SS23: Aufgaben 2, 3, 4

1. Für die Sprache $L = \{a^x b^y c^{x+y} \mid x, y \in \mathbb{N}\}$: sind die folgenden Sprachen regulär?

(a) $\phi(L)$ für $\phi = \{(a, 0), (b, \epsilon), (c, 0)\}$
(b) $\phi(L)$ für $\phi = \{(a, 0), (b, 1), (c, \epsilon)\}$

2. gilt für alle Morphismen h und Sprachen L_1, L_2:

(a) $h(L_1 \cup L_2) = h(L_1) \cup h(L_2)$?
(b) $h(L_1 \cap L_2) = h(L_1) \cap h(L_2)$?
(c) $h(L_1 \setminus L_2) = h(L_1) \setminus h(L_2)$?
(d) $h(L_1 \cdot L_2) = h(L_1) \cdot h(L_2)$?

Wenn ja, Beweis; wenn nein, Gegenbeisp. mit $L_i \in$ REG.

3. Beweisen Sie: wenn ein DFA A mit n Zuständen alle Wörter der Länge $< n$ akzeptiert, dann ist Lang$(A) = \Sigma^*$.

   Hinweis: führen Sie einen indirekten Beweis: wenn ein längeres Wort $w \notin$ Lang(A), dann konstruieren Sie daraus einen Widerspruch zur Annahme.

   Warum funktioniert dieser Beweis nicht für NFA?

   Gibt es NFA, für welche die Aussage nicht gilt?

   Mit welcher (von n abhängigen) Schranke gilt eine entsprechende Aussage für NFA?

   Läßt sich diese Schranke realisieren?

4. Welche der folgenden Aussagen gelten für alle Sprachen $L \subseteq \Sigma^*$ und alle Morphismen $\phi : \Sigma^* \to \Gamma^*$?

(a) $\phi^{-1}(\phi(L)) = L$
(b) $\phi^{-1}(\phi(L)) \subseteq L$
(c) $\phi^{-1}(\phi(L)) \supseteq L$

Welche der folgenden Aussagen gelten für alle Sprachen $L \subseteq \Gamma^*$ und alle Morphismen $\phi : \Sigma^* \to \Gamma^*$?

(a) $\phi(\phi^{-1}(L)) = L$
(b) $\phi(\phi^{-1}(L)) \supseteq L$
(c) $\phi(\phi^{-1}(L)) \subseteq L$

Wenn ja, Beweis; wenn nein, Gegenbeispiel mit $L \in$ REG.

5. Geben Sie eine Abbildung $\phi : \Sigma^* \to \Gamma^*$ an mit $\phi(\epsilon) = \epsilon \wedge \neg\forall u, v \in \Sigma^* : \phi(u \cdot v) = \phi(u) \cdot \phi(v)$.

Gibt es eine Abb. $\phi : \Sigma^* \to \Gamma^*$ mit
$\phi(\epsilon) \neq \epsilon \wedge \forall u, v \in \Sigma^* : \phi(u \cdot v) = \phi(u) \cdot \phi(v)$?

Hinweise zur Notation

- Notation für Mengen: vollständig ist:
  $M = \{E \mid \lambda(x : T).P\}$
  nach dem Trennstrich steht eine Funktion, mit Name und Typ der abstrahierten Variablen, diese Deklaration ist in E und P sichtbar
  Bsp: $M = \{x^2 \mid \lambda(x : \mathbb{N}).(x > 2)\} = \{9, 16, \dots\}$
  Die Bedeutung dieser Definition ist:
  $e \in M \iff (\exists(x : T) : P \wedge e = E)$.
  Bsp: $16 \in M$, denn $\exists(x : \mathbb{N}) : (x > 2) \wedge 16 = x^2$.
  Vermeidung des Lambdas durch: $\{E \mid x \in T \wedge P\}$

oft noch kürzer: $\{E \mid P\}$, die abstrahierten Variablen sind dann die, welche in P (oder in E?) frei vorkommen. (Bsp: was bedeutet $\{x + y \mid x > 8\}$?)

- Ansatz für $h(L_1) \cap h(L_2) \stackrel{?}{\subseteq} h(L_1 \cap L_2)$ (schlägt fehl)
  - die Teilmengen-Relation $A \subseteq B$ ist definiert als $\forall x : (x \in A \Rightarrow x \in B)$
  - $x \in A$ bedeutet hier: $x \in h(L_1) \cap h(L_2)$,
  - ist äq. (nach Def. $\cap$) zu $x \in h(L_1) \wedge x \in h(L_2)$,
  - ist äq. (nach Def. $h(L)$ zu $x \in \{h(w) \mid w \in L_1\} \wedge x \in \{h(w) \mid w \in L_2\}$,
  - ist äq. (nach o.g. Def. Mengenschreibweise) zu $(\exists w : w \in L_1 \wedge x = h(w)) \wedge (\exists w : w \in L_2 : x = h(w))$
  - ist *nicht* äq. zu $(\exists w : (w \in L_1 \wedge w \in L_2) \wedge x = h(w))$ vgl. VL Modellierung: $(\exists w : P) \wedge (\exists w : Q) \not\equiv (\exists w : P \wedge Q)$

Anwendung NFA: Spursprachen

Motivation

- für beliebigen Ausdruck B, Anweisung A:
  die Programme P, Q, R sind äquivalent:

```
P : while (B) { A }
Q : if (B) { A ; while (B) { A } }
R : start: if (not B) goto end; A; goto start
```

- Beweis d. Konstruktion und Vergleich v. Spur-Sprachen:
  $\mathsf{Trace}(P) = \mathsf{Trace}(Q) = \mathsf{Trace}(R) = (B = 1; A)^*(B = 0)$

- Herausforderung: reguläre und nützliche Approximation der tatsächlichen Spursprache

Beispiel

- für dieses strukturierte Programm S (Ablaufsteuerung durch Schleifen)

```
{ foo : while (a) { bar : while (b)
      { p; if (c) break bar;
        q; if (c) continue foo; r; }}}
```

  ist äquiv. flaches Programm F gesucht (A.S. d. Sprünge)

- Versuch:

```
{start: if (!a) goto end; p;
  if (b || c) goto start; q; goto start; end : skip
```

- `ein Wort in Spur(F), aber nicht in Spur(S):`

```
 [(state [(c,True),(b,False),(a,True)],Execute p)]
```

Syntax und Semantik imperativer Programme

- Syntax: Anweisung ist: Zuweisung oder Sprung oder bedingter Sprung,

  Programm ist Folge von (markierten) Anweisungen.

  Bedingter Sprung ist: Bedingung und Sprungziel.

  alle Zuweisungen: $A_1, \ldots$, alle Bedingungen: $B_1, \ldots$.

- Semantik als Zustands-Übergangs-System:

  Zustandsmenge $=$ Position (PC) $\times$ Variablenbelegung

  Ausführung: $s_0 \to s_1 \to s_2 \to \ldots$, jeder Einzelschritt:

  optional Prüfen einer Bedingung, dann Ausführen einer Anweisung

Endliche Abstraktion der Zustandsmenge

- die tatsächliche Zustandsmenge ist sehr groß (alle Belegungen von Speicherstellen mit Maschinenzahlen)

- endliche Beschreibung durch Abstraktion: von Zustand s auf Wahrheitswerte der Bedingungen $B_1(s), B_2(s), \ldots$.

  „nur“ $2^{|B|}$ verschiedene abstrahierte Zustände.

- keine detaillierte Modellierung der Wirkung von A_i auf B_j: der (abstrahierte) Zustand nach Ausführen einer Anweisung ist unbestimmt

Abstraktion der Programm-Ausführung

- aus konkretem Programmtext P: konstruiere Epsilon-NFA $E(P)$ mit Übergängen:
  - Anweisung $p : A$ modelliert durch Übergänge $\{((p, s), (p + 1, s')) \mid s, s' \in Z\}$ in $\delta(A)$
  - Anweisung $p :$ goto q durch Spontan-Übergänge $\{((p, s), (q, s)) \mid s \in Z\}$
  - Anweisung $p :$ if B goto q durch Spontan-Übergänge $\{((p, s), (q, s)) \mid s \in Z \wedge B(s)\}$ $\cup\{((p, s), (p + 1, s)) \mid s \in Z \wedge \neg B(s)\}$
- berechne zustandsmarkierte Sprache des Automaten: Alphabet $Q \times \Sigma$, für $Q =$ Pos $\times\, 2^B$ und $\Sigma = A$, projiziere das Alphabet auf $2^B \times A$ (ignoriere Q) NB: diese Projektion ist ein Morphismus

Spursprachen—Details

- Korrektheit: wenn diese Spursprachen der abstrakten Programme übereinstimmen, dann sind die alle konkreten Instantiierungen der abstrakten Programme äquivalent.

- (nicht vollständig: es gibt konkrete äquivalente Programme, deren Abstraktionen nicht äq. sind)

- Implementierung benutzt die NFA-Bibliothek

  Quelltexte siehe
  `https://gitlab.imn.htwk-leipzig.de/autotool/all0/-/blob/master/collection/src/Flow/Common/Semantics.hs`

Kontextfreie Grammatiken

Motivation, Beispiel

- Dyck-Sprache D (korrekt geklammerte Wörter) ist nicht regulär, aber nützlich, zur Beschreibung der Syntax von Programmiersprachen mit Klammern/Blöcken
- $D = \text{Lang}(G)$ für $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$
  Bsp.: $S \to aSbS \to aaSbSbS \to aabSbS \to aabbS \to aabb$
- für CFG sind einige Eigenschaften entscheidbar
  Bsp.: Wortproblem: $w \stackrel{?}{\in} \text{Lang}(G)$, Leere: $\text{Lang}(G) \stackrel{?}{=} \emptyset$
- es gelten einige Abschluß-Eigenschaften
  Bsp: $L_1 \in \text{CFL} \wedge L_2 \in \text{REG} \Rightarrow (L_1 \cap L_2) \in \text{CFL}$
- es gibt äquivalentes Maschinenmodell (Keller-Automat)

Definition kontextfreie Grammatik

- Def (Syntax): kontextfreie Grammatik ist $G = (\Sigma, V, S, R)$

  mit Alphabet (Zeichenmenge) Σ, Variablenmenge V, Startvariable $S \in V$, Regelmenge $R \subseteq V \times (\Sigma \cup V)^*$

  alle diese Mengen sind endlich

- Bsp: $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$

- Def (Semantik):

  – $u \rightarrow_G v$ falls $\exists p, q \in \Sigma^*, (l, r) \in R : u = plq \wedge v = prq$
  – Lang$(G) = \{w \mid S \rightarrow_G^* w \wedge w \in \Sigma^*\}$

- Bsp: $aSbS \rightarrow_G aSb$, $aabb \in$ Lang(G), $ba \notin$ Lang(G)

Ableitungsbäume

- Def: Ableitungsbaum für $G = (\Sigma, V, S, R)$ ist gerichteter geordneter Baum mit Knoten-Markierung $m : K \to (V \cup \Sigma \cup \{\epsilon\})$
  - für jeden inneren Knoten k: $m(k) \in V$
  - für die Wurzel k: $m(k) = S$
  - für jedes Blatt k: $m(k) \in \Sigma$ oder $m(k) = \epsilon$
  - jedes Epsilon ist Einzelkind
  - für jeden inneren Knoten k mit Kind-Knoten $k_1, \ldots, k_n$: $(m(k), m(k_1) \cdot \ldots \cdot m(k_n)) \in R$
- Def: das Rand-Wort eines Ableitungsbaumes ist das Produkt der Blatt-Markierungen (von links nach rechts)
- Satz: $\forall w : w \in$ Lang$(G) \iff \exists$ Ableitungsbaum b für G : rand$(b) = w$.

Eine Charakterisierung der Dyck-Sprache

- Def: die Höhe $h : \{a, b\}^* \to \mathbb{Z} : w \mapsto |w|_a - |w|_b$.
  Bsp: $h(aabb) = 0, h(aab) = 1, h(b) = -1, h(ba) = 0$.

- Satz: $w \in D \iff h(w) = 0 \wedge \forall u \sqsubseteq w : h(u) \geq 0$.
  dabei ist $\sqsubseteq$ die Präfix-Relation.
  Begründung: jeder Präfix u von $w \in D$ enthält wenigstens soviele öffnende Klammern wie schließende.

- Satz: für $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$ gilt Lang$(G) = D$.
  Beweis (Plan):
  - Lang$(G) \subseteq D$: Induktion über G-Ableitungsbäume
  - $D \subseteq$ Lang(G): Induktion über Länge der Wörter.

$\mathbf{Lang}(G) \subseteq D$

- zu zeigen ist für jeden G-Ableitungsbaum b:

  $h(\mathsf{rand}(b)) = 0$ und $\forall u \sqsubseteq \mathsf{rand}(b) : h(u) \geq 0$.

  Induktion über alle Bäume k, die keine Blätter sind (sondern $m(k) = S$). Anfang: $S \to \epsilon$, Schritt: $S \to aSbS$.

  $\mathsf{rand}(k) = a \cdot \mathsf{rand}(k_1) \cdot b \cdot \mathsf{rand}(k_2)$

  $h(\mathsf{rand}(k)) = h(a \cdot \mathsf{rand}(k_1) \cdot b \cdot \mathsf{rand}(k_2)) =$
  $h(a) + h(\mathsf{rand}(k_1)) + h(b) + h(\mathsf{rand}(k_2)) = 1 + 0 - 1 + 0 = 0$

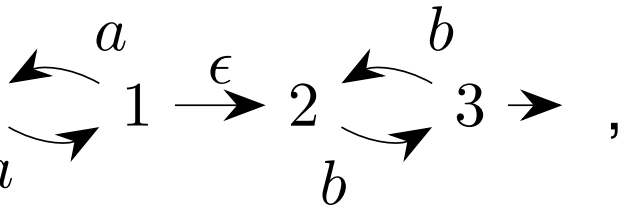
  für $u \sqsubseteq a \cdot \mathsf{rand}(k_1) \cdot b \cdot \mathsf{rand}(k_2)$: Fallunterscheidung:

  - $u = \epsilon$,
  - $u = a \cdot u_1$ mit $u_1 \sqsubseteq \mathsf{rand}(k_1)$,
  - $u = a \cdot \mathsf{rand}(k_1) \cdot b \cdot u_2$ mit $u_2 \sqsubseteq \mathsf{rand}(k_2)$,

$D \subseteq \mathbf{Lang}(G)$

- zu zeigen ist: wenn $h(w) = 0$ und $\forall u \sqsubseteq w : h(u) \geq 0$, dann gibt es einen G-Ableitungsbaum k mit $\mathsf{rand}(k) = w$.
- Induktion nach $|w|$. Anfang: $w = \epsilon$.
  Schritt: $w \neq \epsilon$:
  - Sei u der kürzeste nicht leere Präfix von w mit $h(u) = 0$. (Existiert, denn w selbst ist ein solcher Präfix)
  - Dann $u = au'b$ und $w = au'bv$. Wir zeigen $u' \in D$ und $v \in D$. (Betrachte dazu Präfixe $p \sqsubseteq u'$, $q \sqsubseteq v$.)
  - Dann gibt es (nach I.V.) Abl.-B. k_1 mit $\mathsf{rand}(k_1) = u'$ und k_2 mit $\mathsf{rand}(k_2) = v$.
  - Dann Ableitungsbaum $k = (S \to ak_1bk_2)$ mit $\mathsf{rand}(k) = w$.

REG $\subseteq$ CFL

- Satz: jede reguläre Sprache ist kontextfrei.
- Beweis: gegeben ist $A = (\Sigma, Q, I, F, \delta, \sigma) \in \mathsf{ENFA}(\Sigma)$.
  Wir nehmen zunächst an, daß $I = \{i\}$
  konstruieren $G = (\Sigma, V, S, R) \in \mathsf{CFG}$ mit $V = Q$, $S = i$
  $R = \{(p, aq) \mid (p, q) \in \delta(a)\} \cup \sigma \cup \{(f, \epsilon) \mid f \in F\}$.
  es gilt $\mathsf{Lang}(G) = \mathsf{Lang}(A)$.
- Bsp 

, $G = (\{a, b\}, \{0, 1, 2, 3\}, 0, R)$
  mit $R = \{(0, a1), \ldots, (1, 2), \ldots, (3, \epsilon)\}$
- für $|I| \neq 1$: neuer initialer Zustand q_0 mit Spontan-Übergängen $\{q_0\} \times I$. Leider! Das wäre einfacher für CFG mit Menge von Startvariablen.

Reguläre Grammatiken

- bei voriger Konstruktion für ENFA $\rightarrow$ CFG entstehen Regeln (l, r) mit $r \in \Sigma \cdot V \cup V \cup \{\epsilon\}$.
- Def: eine CFG mit Regeln $R \subseteq V \times (\Sigma \cdot V \cup V \cup \Sigma \cup \{\epsilon\})$ heißt *reguläre* Grammatik
- Satz: für jede reguläre Grammatik $G = (\Sigma, V, S, R)$ gilt: Lang$(G) \in$ REG.

  Beweis: Konstruktion eines ENFA $A = (\Sigma, Q, I, F, \delta, \sigma)$ mit Lang$(A) =$ Lang(G).

  $Q = V \cup \{q_f\}$ (zusätzlicher Finalzustand q_f)

  für Regel $(l, r) \in V \times \Sigma$: Übergang $(l, q_f) \in \delta(a)$.

  $I = \{S\}$, $F = \{q \mid (q, \epsilon) \in R\} \cup \{q_f\}$

CFL $\not\subseteq$ REG

- Satz: es gibt eine kontextfreie Sprache, die nicht regulär ist.

- Beweis: $L = \{a^n b^n \mid n \in \mathbb{N}\}$.
  - $L \notin$ REG mit Schleifensatz oder Nerode-Kongruenz.
  - $L \in$ CFL, denn $L =$ Lang(G) für
    $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSb)\}$

- es gibt viele weitere Sprachen in CFL $\setminus$ REG, z.B. die Dyck-Sprache

Eindeutige GFG

- Def: eine CFG G heißt *eindeutig*, wenn für jedes $w \in \text{Lang}(G)$ genau ein G-Ableitungsbaum b mit $w = \text{rand}(b)$ existiert.

  (andernfalls: mehrdeutig. (und nicht: uneindeutig))

- Bsp: $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$ ist eindeutig.

  Begründung: siehe Beweis für $D \subseteq \text{Lang}(G)$:

  „der kürzeste Präfix u“ ist eindeutig.

- eindeutige CFG sind fundamental für die Verarbeitung von Programmtexten: der Parser bestimmt den Syntaxbaum ($=$ Ableitungsbaum), die Semantik wird durch Induktion über diesen Baum definiert

Eine nicht offensichtliche CFL

- Def: $\Sigma = \{a, b\}$, $L = \{w \cdot w \mid w \in \Sigma^*\}$

  Bsp: $\epsilon, aa, bb, a^4, abab, baba, b^4, \ldots \in L$

- Def: $L' = \Sigma^* \setminus L$, Satz: L' ist kontextfrei.

  Bsp: $a, b, ab, ba, a^3, aab, \ldots, b^3, aaab, aabb, \ldots \in L'$

- später: CFL ist *nicht* unter Komplement abgeschlossen, aber in diesem Spezialfall funktionert es, mit einem Trick:

  wenn $w \in L'$ und $|w| = 2n$, dann existsiert $i \in [1 \ldots n]$ mit $a = w_i \neq w_{i+n} = b$ (oder andersherum)

  $w \in \Sigma^x a \Sigma^y \cdot \Sigma^x b \Sigma^y$ mit $x = i - 1, y = n - i$.

  Der Trick ist: $\Sigma^y \cdot \Sigma^x = \Sigma^x \cdot \Sigma^y$, also $w \in \Sigma^x a \Sigma^x \cdot \Sigma^y b \Sigma^y$

Zusatz: Potenzen und primitive Wörter

- die eben verwendete Sprache
  $L_2 = \{w \cdot w \mid w \in \Sigma^*\} = \{w^2 \mid w \in \Sigma^*\}$ ist die Sprache der zweiten Potenzen.

- Def: $L_k = \{w^k \mid w \in \Sigma^*\}$. Bsp: $a^6 \in L_3, a^5b \notin L_3$.

- Def: die Sprache der *primitiven* Wörter: $P = \Sigma^* \setminus \bigcup_{k \geq 2} L_k$

  Bsp: $a, b, ab, ba, aab, aba, abb, \ldots, aaab, \ldots \in P$

- ist P kontextfrei? — ist seit Jahrzehnten offen.

Aufgaben

SS 23: 1 bis 4, Zusatz: 5

1. Geben Sie eine CFG an für $\{a^x b^y c^{x+y} \mid x, y \in \mathbb{N}\}$

2. Geben Sie eine CFG an für $\{a^x b^y \mid x \neq y\}$.

   Geben Sie eine CFG an für $\{a, b\}^* \setminus \{a^x b^x \mid x \in \mathbb{N}\}$.

3. Für $\Sigma = \{a, b\}$, $E = \{w \mid w \in \Sigma^* \wedge h(w) = 0\}$,

   $G = (\Sigma, \{S\}, S, \{(S, \epsilon), (S, aSbS), (S, bSaS)\})$

   - zeigen Sie Lang$(G) \subseteq E$
   - gilt $E \subseteq$ Lang(G)?
   - ist G eindeutig?

4. Wenn die Konstruktion für REG $\subseteq$ CFL auf einen DFA angewendet wird: ensteht dadurch eine eindeutige Grammatik?

   Wenn bei dieser Konstruktion eine eindeutige Grammatik ensteht: war die Eingabe ein DFA?

   jeweils Beispiel angeben, Verallgemeinerung diskutieren.

5. Potenzen und primitive Wörter:

   (a) zeigen Sie $L_2 \notin$ REG (Schnitt mit geeigneter regulärer Sprache, dann Schleifensatz)
   (b) geben Sie eine CFG für $\Sigma^* \setminus L_3$ an.
   (c) beweisen Sie $P \notin$ REG. („Beweis“ durch Autorität: wenn $P \in$ REG, dann wegen REG $\subseteq$ CFL auch $P \in$ CFL, und dann wäre diese Frage nicht offen)

Normalformen CFG, Wortproblem

Erreichbar, produktiv, reduziert, leer

- Def: eine Variable v einer CFG $G = (\Sigma, V, S, R)$ heißt *erreichbar*, wenn $\exists p, q \in (V \cup \Sigma)^* : S \to_R^* pvq$.
- Def: eine Variable v einer CFG $G = (\Sigma, V, S, R)$ heißt *produktiv*, wenn $\exists w \in \Sigma^* : v \to_R^* w$.
- $(\{a, b\}, \{X, Y, Z\}, X, \{(X, aY), (Y, b), (Y, cXZ), (Z, bZb)\})$
- Satz: Lang$(G) \neq \emptyset$ gdw. Start-Variable ist produktiv.
- Def: $G \in$ CFG heißt *reduziert*, wenn jede Variable erreichbar und produktiv ist.
- Ü: gibt es zu jeder $G \in$ CFG eine reduzierte $G' \in$ CFG mit Lang$(G) =$ Lang(G')? — Bsp. $G = (\{a, b\}, \{S\}, S, \emptyset)$.

Reduktion

- Satz: es gibt einen Algorithmus, der die Menge der erreichbaren Variablen in $O(|G|)$ Zeit berechnet.

  Implementierung: Tiefensuche in dem Graphen der Relation $\{(X, Y) \mid (X, r) \in R, Y \in r\}$, beginnend bei S.

- Satz: . . . produktiven Variablen . . . (Übungsaufgabe?)

- Folgerung: Lang$(G) \stackrel{?}{=} \emptyset$ in welcher Zeit?

- Satz: zu jeder $G \in$ CFG mit Lang$(G) \neq \emptyset$ kann äquivalente reduzierte $G' \in$ CFG in . . . Zeit konstruiert werden

  Beweis: 1. alle nicht produktiven löschen, 2. alle (dann) nicht erreichbaren löschen. — *nicht andersherum*! (Ü)

Epsilon-Freiheit

- Def: G heißt *epsilon-frei*, wenn $\forall (l, r) \in R : r \neq \epsilon$.
- Bsp: $(\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$ nicht epsilon-frei
- Satz: für jede $G \in$ CFG existiert epsilon-freie $G' \in$ CFG mit Lang(G') = Lang$(G) \setminus \{\epsilon\}$.
- Beweis (Algorithmus):
  - bestimme $E = \{v \mid v \in V \wedge v \rightarrow_R^* \epsilon\}$. (Ü: wie genau?)
  - R' entsteht aus R durch Löschen aller Regeln (l, ϵ) und Hinzufügen aller Regeln, die aus vorhandenen Regeln durch Löschen von Variablen aus E entstehen.
- Bsp: $E = \{S\}$, $R' = \{(S, aSbS), (S, abS), (S, aSb), (S, ab)\}$

Kettenregeln

- Def: eine Regel $(l, r) \in V \times V$ heißt *Kettenregel*.
- Bsp: $(\{a, b\}, \{S, X, Y\}, S, R)$ mit
  $R = \{(S, X), (S, Y), (X, aX), (X, \epsilon), (Y, bY), (Y, \epsilon)\}$
- Satz: für jede $G \in$ CFG existiert kettenregelfreie $G' \in$ CFG mit Lang(G') = Lang(G).
- Beweis (Algorithmus): die Ketten-Relation ist $K = R \cap (V \times V)$. jede Kettenregel (l, r) wird ersetzt durch alle Regeln $\{(l, r') \mid \exists l' : (l, l') \in K^+ \wedge (l', r') \in R\}$.
- Bsp: $K = \{(S, X), (S, Y)\}$,
  $\{(S, aX), (S, \epsilon), (S, aY), (S, \epsilon), (X, aX), (X, \epsilon), (Y, bY), (Y, \epsilon)$

Chomsky-Normalform

- Def: $G \in$ CFG in Chomsky-NF, falls $\forall (l, r) \in R : r \in (\Sigma \cup V^2)$ (genau ein Zeichen oder genau zwei Variablen)
- Satz: zu jeder eps-freien kettenfreien $G \in$ CFG gibt es eine Chomsky-NF $G' \in$ CFG mit Lang$(G) =$ Lang(G').
- Beweis (Algorithmus):
  - neue Variablen v_a für $a \in \Sigma$ und Regeln (v_a, a)
  - für jede Regel (l, r) mit $r \in (V \cup \Sigma)^{\geq 2}$ in r jeden Buchstaben a durch Variable v_a ersetzen
  - jede Regel (l, r) mit $r = r_1 r_2 \ldots r_k \in V^k$ für $k > 2$: ersetzen durch Regeln (mit Hilfsvariablen $h_2, \ldots, h_{k-1}$) $(l, r_1 h_2), (h_2, r_2 h_3), \ldots, (h_{k-1}, r_{k-1} r_k)$
  - $(S, aSbS)$ zu $\{(A, a), (B, b), (S, Ah_2), (h_2, Sh_3), (h_3, BS)\}$

Das Wortproblem für CFG

- Def: das Wortproblem für $G \in$ CFG ist:
  Eingabe: $w \in \Sigma^*$, Ausg.: $w \in \text{Lang}(G)$ (Wahrheitswert)
- man kann das Wortproblem „nach Definition“ lösen, dazu muß man alle Ableitungen oder Ableitungsbäume für w aufzählen. Für Epsilon-freie G sind das endlich viele, aber evtl. sehr viele (exponentiell in $|w|$).
- deswegen ist es überraschend, daß $w \stackrel{?}{\in} \text{Lang}(G)$ in Polynomialzeit implementiert werden kann,
  für *beliebige* G (und nicht etwa nur für eindeutige oder auf andere Weise deterministische)
- Konstruktion ist ähnlich zu Wortproblem für NFA (mit Verwaltung von Zustandsmengen statt Zuständen)

Der CYK-Algorithmus für CFG-Wortproblem

- Idee: für jeden Abschnitt $w[i \ldots j]$ für $1 \leq i \leq j \leq |w|$
  bestimme die Menge $M_{i,j} = \{v \mid v \to_R^* w[i \ldots j]\}$
  dann $w \in \text{Lang}(G) \iff S \in M_{1,|w|}$.
- benutze dynamische Programmierung: von kleinen Abschnitten zu großen (Induktion über $(j - i)$)
- John Cocke, Jacob Schwartz (1970), Daniel Younger (1967), Tadao Kasami (1965), Itiroo Sakai (1962)
  einfache (und schnelle) Implementierung,
  falls G in Chomsky-Normalform vorliegt.
  $M_{i,i} = \{v \mid (v, w_i) \in R\}$; für $i < j$: $M_{i,j} = \{v \mid \exists h \in [i \ldots j-1], (v, pq) \in R, p \in M_{i,h}, q \in M_{h+1,j}\}$
- Kosten: $O(|w|^3)$ (i, j, h laufen über $[1 \ldots |w|]$)

Beispiel Chomsky-Nf, CYK-Algorithmus

- $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$
- epsilon-freie G' mit Lang$(G') = D \setminus \{\epsilon\}$:
  $G' = (\{a, b\}, \{S\}, S, \{(S, aSbS), (S, abS), (S, aSb), (S, ab)\})$
- Chomsky-Nf mit Nachnutzen von Hilfsvariablen
  $G'' = (\{a, b\}, \{S, A, B, H, I, K\}, S, R)$ mit
  $R = \{(A, a), (B, b), (S, AH), (H, SI), (I, BS), (S, AI),$
  $(S, AK), (K, SB), (S, AB)\})$
- $w = abab$,

$M_{i,j}$	$j=1$	2	3	4
$i=1$	$\{A\}$	$\{S\}$	$\emptyset$	$\{S\}$
2		$\{B\}$	$\emptyset$	$\{I\}$
3			$\{A\}$	$\{S\}$
4				$\{B\}$

  Bsp: $(S, AI) \in R, A \in M_{1,1}, I \in M_{2,4}$, deswegen $S \in M_{1,4}$

Aufgaben

SS 23: 1, 2, 4, 6 (wenigstens a, b) , 8 (a, b), Zusatz: 7

1. Geben Sie ein Verfahren an zur Berechnung der produktiven Variablen einer CFG. Führen Sie an einem Beispiel aus dem Skript oder einem selbst gewählten vor. Diskutieren sie seine Laufzeit.

   Geben Sie ein Beispiel an, für das durch dieses Verfahren:

   (a) alle nicht erreichbaren Variablen löschen
   (b) all (dann) nicht produktiven Variablen löschen

   keine reduzierte Grammatik entsteht.

2. Für die Grammatik $G = (\{a, b\}, \{S\}, S, \{(S, b), (S, aSS)\})$:
   geben Sie eine Chomsky-Normalform an.

   Entscheiden Sie $ababb \stackrel{?}{\in} \mathsf{Lang}(G)$ und $abbab \stackrel{?}{\in} \mathsf{Lang}(G)$ nach dem CYK-Algorithmus

3. Wie hängen die Kosten für den CYK-Algorithmus von $|\Sigma|, |V|, |R|$ ab? Geben Sie effiziente Datenstrukturen für die Berechnung der $M_{i,j}$ an. Vergleichen Sie mit der Implementierung in autotool.

   Das angegebene CYK-Verfahren liefert den Wahrheitswert von $w \in \mathsf{Lang}(G)$. Wie ist es zu erweitern, so daß ohne Mehrkosten auch ein Ableitungsbaum für w berechnet wird?

4. Für die vorige Grammatik $G = (\{a, b\}, \{S\}, S, \{(S, b), (S, aSS)\})$ und die Grammatik $H = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$ (für die Dyck-Sprache D): Überprüfen Sie $\mathsf{Lang}(G) = \mathsf{Lang}(H) \cdot b$ an Beispielen. Beweisen Sie diese Beziehung, indem Sie ein Verfahren angeben, das aus jedem G-Ableitungsbaum k einen H-Ableitungsbaum k' konstruiert mit $\mathsf{rand}(k) = \mathsf{rand}(k') \cdot b$, und umgekehrt.

5. Für die Dyck-Sprache D gilt $D = \phi(D)^R$, wenn $\cdot^R$ das Spiegeln bezeichnet und ϕ den Morphismus $a \leftrightarrow b$. Wendet man $\phi(\cdot)^R$ auf die Regeln der Grammatik H an, entsteht $H' = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, SaSb)\})$. Zeigen Sie $\mathsf{Lang}(H) = \mathsf{Lang}(H')$ durch Umformung von Ableitungsbäumen.

6. Für eine endliche Menge V: Wir betrachen die Sprache $A(V)$ der vollständig geklammerten allgemeingültigen aussagenlogischen Formeln mit Operatoren: Negation (einstellig), Disjunktion (zweistellig), Konjunktion (zweist.) über den aussagenlogischen Variablen V. Bsp: $((p \vee q) \vee ((\neg p) \wedge (\neg q))) \in A(\{p, q\})$, $(p \vee q) \notin A(\{p, q\})$. Zeigen Sie: jedes $A(V)$ ist kontextfrei. Hinweis: die Grammatik hat $2^{2^{|V|}}$ Variablen, jede Variable bezeichnet einen Werteverlauf einer $|V|$-stelligen Booleschen Fkt.

   Beschreiben Sie die Regelmenge für die Fälle:

   (a) $V = \emptyset$, (b) $V = \{p\}$, (c) allgemein

7. Geben Sie eine kontextfreie Grammatik an für das Komplement der Sprache $\{(a^n b)^n \mid n \in \mathbb{N}\}$

8. Geben Sie einen Algorithmus an, der bei Eingabe einer Grammatik $G = (\Sigma, V, S, R) \in$ CFG für jede Variable $v \in V$ entscheidet, ob die Sprache der aus v ableitbaren Wörter $L_v = \text{Lang}(G_v)$ mit $G_v = (\Sigma, V, \{v\}, R)$ die folgende Eigenschaft hat:

(a) $L_v \neq \emptyset$ (das ist tatsächlich der Inhalt einer anderen Übungsaufgabe—welcher?)

(b) $\epsilon \in L_v$

(c) $(L_v \setminus \{\epsilon\}) \neq \emptyset$

(d) L_v ist endlich

Welche Vereinfachungen sind möglich, wenn für die Eingabe vorausgesetzt wird:

- G ist reduziert
- G ist reduziert und in Chomsky-Normalform

(Nicht)Abschlußeigenschaft. von CFL

Einfache Fälle, Motivation

- Satz: wenn $L_1 \in$ CFL, $L_2 \in$ CFL, dann
  - $L_1 \cup L_2 \in$ CFL, $\quad L_1 \cdot L_2 \in$ CFL, $\quad L_1^* \in$ CFL

  Beweis: (offensichtliche) Konstruktion einer CFG.

- Frage: wenn $L_1 \in$ CFL, $L_2 \in$ CFL, dann $L_1 \cap L_2 \in$ CFL?

  Bsp: $L_1 = a^* \cdot \{b^n c^n \mid n \in \mathbb{N}\}$, $L_2 = \{a^n b^n \mid n \in \mathbb{N}\} \cdot c^*$, $L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$, dafür gelingt keine Konstruktion einer CFG—das müssen wir beweisen. Dazu zeigen wir einen Schleifensatz für CFL.

- später: wenn $L_1 \in$ CFL, $L_2 \in$ REG, dann $L_1 \cap L_2 \in$ CFL.

Einfache Fälle (Beispiel)

- Satz: $L_1 \in \mathsf{CFL} \wedge L_2 \in \mathsf{CFL} \Rightarrow L_1 \cdot L_2 \in \mathsf{CFL}$.
- Beweis: $L_i = \mathsf{Lang}(G_i)$ mit $G_i = (\Sigma, V_i, S_i, R_i)$.

  wir nehmen an $V_1 \cap V_2 = \emptyset$ (sonst umbenennen)

  konstruieren $G = (\Sigma, V, S, R)$, mit $S \notin V_1 \cup V_2$,

  $V = \{S\} \cup V_1 \cup V_2$, $R = \{(S, S_1 S_2)\} \cup R_1 \cup R_2$
- analog für Vereinigung und Stern: Übung.

Schleifensatz für CFL (Plan)

- (Whgl) Schleifensatz für REG: jede lange Rechnung eines NFA muß eine Zustand wiederholen, kann deswegen ab- und aufgepumpt werden.
- für CFG: betrachten lange Pfade in Ableitungsbäumen, diese müssen eine Variable wiederholen.
  Der Abschnitt des Pfades zwischen den Positionen der Duplikate erzeugt *zwei* Teilstücke des Randwortes (einen links, einen rechts). Diese *gleichzeitig* pumpen.
- (Whdlg) für REG: die Wiederholung findet spätestens nach $|Q|$ Zeichen *von links* statt.
- für CFG: die Wiederholung spätestens für eine Teilbaum der Höhe $|V|$.

Schleifensatz für CFL (Formulierung)

- Def: Pump-Eigenschaft für CFL:
  $\mathsf{Pump}_{\mathsf{CFL}}(L,n) := \forall w \in L : (|w| \geq n) \Rightarrow \exists p,q,r,s,t \in \Sigma^* :$
  $w = pqrst \wedge |qrs| \leq n \wedge |qs| \geq 1 \wedge \forall k \in \mathbb{N} : pq^krs^kt \in L.$
  $\mathsf{Pump}_{\mathsf{CFL}}(L) := \exists n \in \mathbb{N} : \mathsf{Pump}_{\mathsf{CFL}}(L,n).$
- Satz: $\forall L \in$ CFL : $\mathsf{Pump}_{\mathsf{CFL}}(L)$. (Beweis folgt später)
- Bsp: für die Sprache der Palindrome
  $L = \{w \mid w \in \{a,b\}^* \wedge w^R = w\}$ gilt $\mathsf{Pump}_{\mathsf{CFL}}(L,3)$.
  Bsp: $abaababaaba = aba \cdot a \cdot b \cdot a \cdot aba$ mit
  $\forall k : aba \cdot a^k \cdot b \cdot a^k \cdot aba \in L$
  Ü: es gilt sogar $\mathsf{Pump}(L,2)$
- Bsp: für die Dyck-Sprache $D \subseteq \{a,b\}^*$ gilt $\mathsf{Pump}_{\mathsf{CFL}}(D,2)$.
  Bsp: $aababbbab = aababb \cdot ab \cdot \epsilon \cdot \epsilon \cdot \epsilon$ mit
  $aababb \cdot (ab)^k \cdot \epsilon \cdot \epsilon^k \cdot \epsilon \in L$

Schleifensatz für CFL (Hilfssatz)

- Hilfssatz: für jede $G = (\Sigma, V, S, R) \in$ CFG in Chomsky-Nf, jeden G-Ableitungsbaum k gilt: $|\mathsf{rand}(k)| \leq 2^{\mathsf{height}(k)-1}$.

  Beweis: Induktion nach $h = \mathsf{height}(k)$,

  – I.A. ($h = 1$) Regel $(l, r) \in V \times \Sigma$ in der Wurzel,
  – I.S. ($h > 1$) Regel $(l, r) \in V \times V^2$ in der Wurzel

- Folgerung: für jede $G = (\Sigma, V, S, R) \in$ CFG in Chomsky-Nf, jeden G-Ableitungsbaum k, jedes $h \in \mathbb{N}$: wenn $|\mathsf{rand}(k)| \geq 2^h$, dann $\mathsf{height}(k) \geq h + 1$.

  Beweis: $2^{\mathsf{height}(k)-1} \geq |\mathsf{rand}(k)| \geq 2^h$.

Schleifensatz für CFL (Beweis)

- (Konkretisierung der Behauptung) Für jede CFG in Chomsky-Nf. $G = (\Sigma, V, S, R)$ gilt $\mathsf{Pump}(\mathsf{Lang}(G), 2^{|V|})$.
- Beweis: Sei $n = 2^{|V|}$, $w \in L = \mathsf{Lang}(G)$ mit $|w| \geq n$, k ein G-Ableitungsbaum mit $\mathsf{rand}(k) = w$.
  - k enthält wenigstens einen Teilbaum $k_1 \trianglelefteq k$ mit $\mathsf{height}(k_1) = |V| + 1$. Dann $|\mathsf{rand}(k_1)| \leq 2^{|V|} = n$
  - k_1 enthält Pfad mit $|V| + 1$ Kanten, also $|V| + 1$ Variablen (innere Knoten) (und einem Blatt).
  - unter diesen Variablen gibt es ein Duplikat v, d.h. Teilbäume $k_3 \triangleleft k_2 \trianglelefteq k_1$ mit $m(k_3) = m(k_2) = v$
  - $\mathsf{rand}(k) = p \cdot \mathsf{rand}(k_2) \cdot t = p \cdot q \cdot \mathsf{rand}(k_3) \cdot s \cdot t = pqrst$
  - wegen $k_2 \trianglelefteq k_1$ ist $|qrs| = |\mathsf{rand}(k_2)| \leq |\mathsf{rand}(k_1)| \leq n$
  - wegen $k_3 \triangleleft k_2$ (echter Teilbaum) ist $qs \neq \epsilon$, also $|qs| \geq 1$.

Schleifensatz für CFL (Anwendung)

- Für $L = \{a^k b^k c^k \mid k \in \mathbb{N}\}$ gilt $\neg$ Pump(L). Beweis (indirekt):

  Falls doch $\exists n$: Pump(L, n), dann wähle $w = a^n b^n c^n \in L$.

  Für jede Zerlegung $w = pqrst$ mit $|qrs| \leq n$ gilt:
  qrs enthält kein a oder qrs enthält kein c.

  Durch Abpumpen werden ein oder zwei Block-Größen (der drei Blöcke a^n, b^n, c^n) geändert, aber die dritte nicht.

  also $\neg\exists n$: Pump(L, n), also $L \notin$ CFL.

- Folgerung: es gilt *nicht* $\forall L_1, L_2 \in$ CFL : $(L_1 \cap L_2) \in$ CFL.

  Begründung (es genügt ein Gegenbeispiel)
  $L_1 = a^* \cdot \{b^k c^k \mid k \in \mathbb{N}\}$, $L_2 = \{a^k b^k \mid k \in \mathbb{N}\} \cdot c^*$,
  $L_1 \cap L_2 = \{a^k b^k c^k \mid k \in \mathbb{N}\} \notin$ CFL

Schleifensatz für CFL (Testfragen)

- Der Schleifensatz $\forall L \in \mathsf{CFL} : \mathsf{Pump}(L)$ wurde mittels G in Chomsky-Nf beweisen. Eine solche Nf. existiert aber nur für $L \setminus \{\epsilon\}$. Ist das ein Problem?

- Für die Dyck-Sprache D gilt $\mathsf{Pump}(D, 2)$, denn man kann ein beliebiges Teilwort ab pumpen.

  Welcher Wert für n in $\mathsf{Pump}(D, n)$ ergibt sich aus dem Beweis des Schleifensatzes?

- im Beweis des Schleifensatzes ist $w = p \cdot q \cdot r \cdot s \cdot t$ mit $r = \mathsf{rand}(k_3)$, also $r \neq \epsilon$. Damit kann $\mathsf{Pump}(L, n)$ verschärft werden zu $\ldots \wedge |r| \geq 1 \wedge \ldots$.

  Gibt es eine Quelle, in der das so dargestellt ist?

Schleifensatz für CFL (Anwendung 2)

- (Übung) für $L = \{a^x b^y a^x b^y \mid x, y \in \mathbb{N}\}$ gilt $\neg$ Pump(L).
- (Anwendung) ist $L_2 = \{ww \mid w \in \{a, b\}^*\}$ kontextfrei?

  Es gilt $L_2 \cap a^*b^*a^*b^* = L \notin$ CFL.

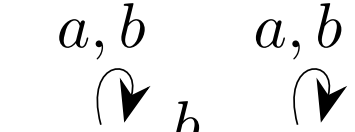
  Wir zeigen (nächste Folie), daß $L_2 \notin$ CFL folgt.
- (Vorsicht) $L_2 \cap a^*ba^*b = \{a^x ba^x b \mid x \in \mathbb{N}\} \in$ CFL
- (Erinnerung) $\{a, b\}^* \setminus L_2$ ist kontextfrei
- Folgerung: es gilt nicht:
  $\forall L : L \in \text{CFL}(\Sigma) \Rightarrow (\Sigma^* \setminus L) \in \text{CFL}(\Sigma)$

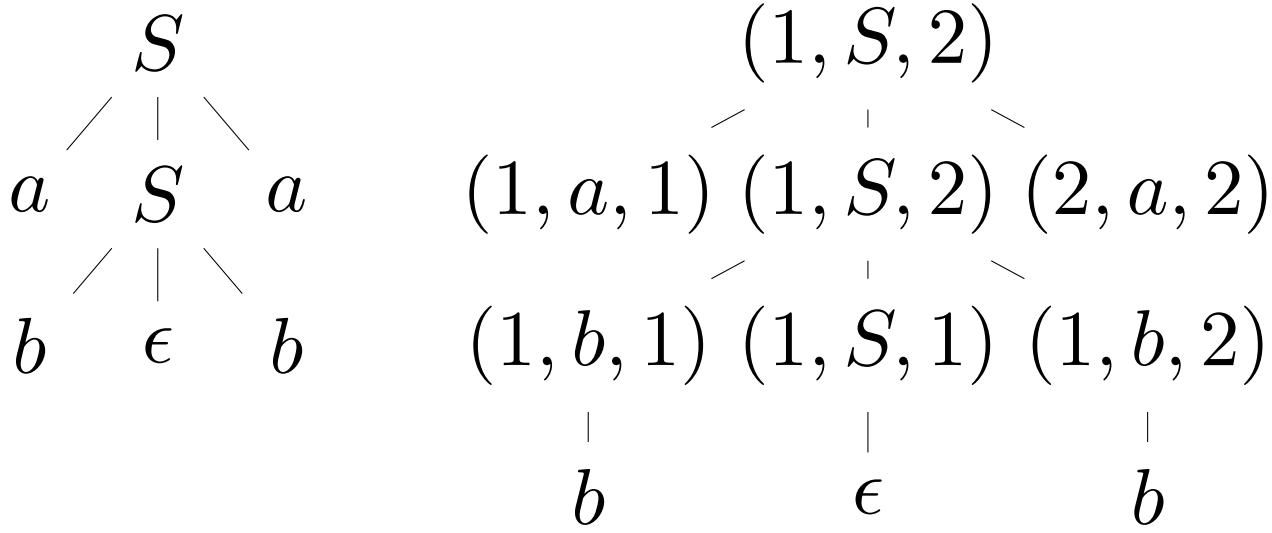
  CFL ist nicht abgeschlossen bzgl. Komplement.

Abschluß von CFL unter Schnitt mit REG

- Satz: $L_1 \in \mathsf{CFL} \wedge L_2 \in \mathsf{REG} \Rightarrow (L_1 \cap L_2) \in \mathsf{CFL}$.
- Beweis: $L_1 = \mathsf{Lang}(G)$ mit $G = (\Sigma, V, S, R) \in \mathsf{CFG}$;
  $L_2 = \mathsf{Lang}(A)$ mit $A = (\Sigma, Q, I, F, \delta) \in \mathsf{NFA}$.
  Konstruieren $G' = (\Sigma, V', S', R')$ mit $V' = Q \times (V \cup \Sigma) \times Q$,
  - Plan: $\mathsf{Lang}(G', (p, v, q)) = \mathsf{Lang}(G, v) \cap \mathsf{Lang}(A, p, q)$
    Bezeichnungen: $\mathsf{Lang}(G', v') := \mathsf{Lang}(\Sigma, V', v', R')$,
    $\mathsf{Lang}(A, p, q) := \mathsf{Lang}(\Sigma, Q, \{p\}, \{q\}, \delta)$,
  - Realisierung: $S' = \{(i, S, f) \mid i \in I, f \in F\}$, $R' =$
    $\{((p, a, q), a) \mid a \in \Sigma, (p, q) \in \delta(a)\} \cup$
    $\{((q_0, l, q_k), (q_0, r_1, q_1) \ldots (q_{k-1}, r_k, q_k)) \mid (l, r) \in R,$
    $r = r_1 \ldots r_k, q_i \in Q\}$. Für $k = 0$: Regel $(q_0, l, q_0) \to \epsilon$
  - (Start-Menge, Transform. auf Start-Symbol ist möglich)
  - Beweis durch Ind. über Ableitungsb. von G bzw. G'

Abschluß CFL unter Schnitt mit REG, Bsp

- $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSa), (S, bSb)\})$
- $A =$ 

, $I \ni 1 \xrightarrow{a} 1 \xrightarrow{b} 1 \xrightarrow{b} 2 \xrightarrow{a} 2 \in F$
- 

- $R' = 1a1 \to a, \ldots, 1S1 \to \epsilon, 2S2 \to \epsilon, 1S1 \to$
  $1a1\ 1S1\ 1S1, 1S1 \to 1a1\ 1S2\ 2S1, 1S1 \to 1a2\ 2S1\ 1S1, \ldots$

Zusammenfassung REG/CFG, Ausblick

- Wortproblem: REG: linear (DFA), CFG: kubisch (CYK)
- Sprache $\neq \emptyset$: produktiver Startzustand/Startsymbol
- Schleifensatz/Pump-Eigenschaft
  - REG: $\ldots w = rst, |rs| \leq n, |s| \geq 1, \forall k : rs^k t \in L$
  - CFL: $\ldots w = pqrst, |qrs| \leq n, |qs| \geq 1, \forall k : pq^k rs^k t \in L$
- Abschluß unter
  - Vereinigung, Verkettung, Stern: REG und CFL
  - Komplement, Durchschnitt: REG ja, CFL nein
  - Schnitt mit REG, Morphismus, inverser M.: REG u. CFL
- nächste VL: Sprachfamilien außerhalb von CFL, Maschinenmodelle (CFL: Kellerautomat)

Aufgaben

SS 23: 2, 3, 5, evtl: 4, 6

1. beweisen Sie den Abschluß von CFL unter Vereinigung und Stern (konstruieren Sie die Grammatiken, jeweils Beispiel und allgemein)
2. für $L = \{a^x b^y a^x b^y \mid x, y \in \mathbb{N}\}$ gilt $\neg\, \mathsf{Pump}_{\mathsf{CFL}}(L)$.
3. Für die Sprach-Operation

   $L_1 \odot L_2 := \{w_1 \odot w_2 \mid w_1 \in L_1 \wedge w_2 \in L_2 \wedge |w_1| = |w_2|\}$

   Geben Sie jeweils ein Beispiel an für

   (a) $L_1 \in \mathsf{REG}, L_2 \in \mathsf{REG}, L_1 \odot L_2 \in \mathsf{REG}$

   (b) $L_1 \in \mathsf{REG}, L_2 \in \mathsf{REG}, L_1 \odot L_2 \notin \mathsf{REG}$

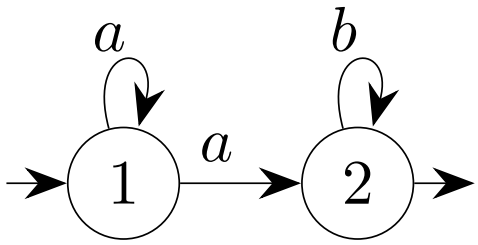
   (c) $L_1 \in \mathsf{CFL}, L_2 \in \mathsf{CFL}, L_1 \odot L_2 \notin \mathsf{CFL}$

   Zeigen Sie $L_1 \in \mathsf{REG} \wedge L_2 \in \mathsf{REG} \Rightarrow L_1 \odot L_2 \in \mathsf{CFL}$.

4. Die Syntax der Sprache Java ist durch eine kontextfreie Grammatik definiert (Java Language Specification, Chapter 19, `https://docs.oracle.com/javase/specs/jls/se20/html/jls-19.html`)
   - wie lautet das Startsymbol (aus dem man das übliche Hello-World-Programme ableiten kann)?
   - enthält diese Grammatik: Epsilon-Regeln, Ketten-R.?
   - Wie kann man das Hello-World-Programm pumpen? Geben Sie ein möglichst großes Programm an, das man *nicht* pumpen kann.

   Beachten Sie: hier geht es um die Menge der *syntaktisch* korrekten Programme. Die Menge der *semantisch* korrekten (d.h., syntaktisch korrekten und richtig typisierten) Programme ist *nicht* kontextfrei.

5. Konstruieren Sie eine CFG für den Durchschnitt von
   - $L_1 =$ Dyck-Sprache, repräsentiert durch CFG $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSbS)\})$,
   - mit $L_2 = a^+b^*$, repräsentiert durch NFA



   nach dem angegebenen Algorithmus. — Hinweise:
   - zuerst das erwartete Resultat voraussagen: beschreiben Sie $L_1 \cap L_2$.
   - Bestimmen Sie allgemein $|V'|$ und $|R'|$ in Abhängigkeit von $|\Sigma|, |V|, |R|, |Q|, \sum_a |\delta_a|$ (= Anzahl d. Kanten von A)
6. Zeigen Sie für $L = \{a^i b^j c^k d^l \mid i = 0 \vee (j = k = l)\}$
   - $\mathsf{Pump}_{\mathsf{CFL}}(L, 1)$
   - $L \notin$ CFL, denn es gibt $K \in$ REG mit $\neg\, \mathsf{Pump}_{\mathsf{CFL}}(L \cap K)$.

Kellerautomaten

Motivation, Plan

- Welches ist das passende Maschinenmodell für CFL?

  Es ist nicht NFA, denn $\mathsf{REG} \subsetneq \mathsf{CFL}$.

- Lösung: wir statten NFA mit zusätzlichem Speicher aus.
  - dieser ist unbegrenzt (beliebig lange Zeichenfolge) (sonst entsteht wieder ein NFA)
  - der Zugriff auf den Speicher ist eingeschränkt (push und pop— am linken Ende)
- Das Maschinenmodell wird nach dem Zusatzspeicher benannt: Kellerautomat (englisch: NPDA, nondet. pushdown automaton). Es charakterisiert CFL.

Definition

- Def: Ein NPDA $A = (\Sigma, Q, i, \Gamma, s, \delta)$ mit
  - Eingabe-Alphabet Σ, Zustandsmenge Q,
  - Initialzustand $i \in Q$, (eigentlich $I \subseteq Q$, aber wir haben bei CFG (leider) auch nur eine Startvariable),
  - Keller-Alphabet Γ, Startsymbol $s \in \Gamma$
  - Übergangsrelation $\delta \subseteq ((\Sigma \cup \{\epsilon\}) \times Q \times \Gamma) \times (Q \times \Gamma^*)$
- Def: Konfigurationsmenge des NPDA ist
  $C = \Sigma^* \times Q \times \Gamma^*$ (Rest der Eingabe, Zustand, Kellerinhalt)
- Def: die Schritt-Relation $\rightarrow_A$ auf C ist
  $\{((a \cdot w, q, t \cdot v), (w, q', u \cdot v)) \mid ((a, q, t), (q', u)) \in \delta\}$
- Lang$(A) = \{w \mid \exists q \in Q : (w, i, [s]) \rightarrow_A^* (\epsilon, q, \epsilon)\}$
  (wir akzeptieren durch leeren Keller)

Beispiel: PDA für Palindrome

- $L = \{w \cdot w^R \mid w \in \{a,b\}^*\}$ (Palindrome gerader Länge)
- PDA A kopiert Eingabe in den Keller, schaltet nichtdeterminstisch um zum Vergleich (und Verbrauch) des Kellers mit weiterer Eingabe.

  $\Sigma = \{a,b\}, \Gamma = \{s,a,b\}, Q = \{0,1\},$

  $\delta = \{((c,0,t),(0,ct)) \mid c \in \Sigma, t \in \Gamma\}$ ... push

  $\cup \{((\epsilon,0,t),(1,t)) \mid t \in \Gamma\}$ ... umschalten

  $\cup \{((c,1,c),(1,\epsilon)) \mid c \in \Sigma\}$ ... pop und Vergleich

  $\cup \{((\epsilon,1,s),(1,\epsilon))\}$ ... fertig
- Zustand kann im obersten Kellersymbol notiert werden, Lang(A) = Lang(A'), $|Q'| = 1, \Gamma' = (\Gamma \times Q) \cup \Gamma, s' = (s,i)$

CFG zu PDA

- Satz: $\forall G \in$ CFG : $\exists A \in$ PDA : Lang$(A) =$ Lang(G)

- Beweis (Konstruktion): $Q = \{0\}, \Gamma = V \cup \Sigma, s = S$

  $\delta = \{((\epsilon, 0, l), (0, r)) \mid (l, r) \in R(G)\}$
  $\cup \{((c, 0, c), (0, \epsilon)) \mid c \in \Sigma\}$.

- Beispiel: $G = (\{a, b\}, \{S\}, S, \{(S, \epsilon), (S, aSa), (S, bSb)\})$
  (Lang$(G) =$ gerade Palindrome)

  $\delta =$
  $\{((\epsilon, 0, S), (0, \epsilon)), ((\epsilon, 0, S), (0, aSa)), ((\epsilon, 0, S), (0, bSb)),$
  $((a, 0, a), (0, \epsilon)), ((b, 0, b), (0, \epsilon))\}$

PDA zu CFG

- Satz: $\forall A \in$ PDA : $\exists G \in$ CFG : Lang$(A) =$ Lang(G)
- Beweis (Plan): $V = Q \times \Gamma \times Q$, Spezifikation:
  Lang$(G, (p, t, q)) =$ die in G aus (p, t, q) ableitbaren Wörter $= \{w \mid ((w, p, t) \to_A^* (\epsilon, q, \epsilon)\}$ A beginnt in p, liest w, erreicht q, verbraucht dabei Kellerzeichen t
- Startsymbole $\{(i, s, q) \mid q \in Q\}$,
  dann Lang$(A) = \bigcup_{q \in Q}$ Lang$(G, (i, s, q))$
- für $((a, q, t), (q', k)) \in \delta$ mit $a \in \Sigma \cup \{\epsilon\}$, $k = k_1 k_2 \ldots k_n$:
  Regeln $(q, t, q_n) \to a(q', k_1, q_1)(q_1, k_2, q_2) \ldots (q_{n-1}, k_n, q_n)$ für $q_i \in Q$.
  Spezialfall $k = \epsilon$ ($n = 0, q_0 = q'$), Regel $(q, t, q') \to a$
- Ü: auf den PDA (2 Zust) f. gerade Palindrome anwenden

Ausblick, Anwendung von PDA

- man kann DPDA (*deterministische* PDA) definieren (so daß $\rightarrow_A$ eindeutig ist, jede Konfiguration hat höchstens einen Nachfolger)
- erhält damit ein Linearzeit-Verfahren für das Wortproblem

  Anwendung: Konstruktion des AST eines Programmtextes, das sollte wirklich linear sein (anstatt kubisch mit CYK)
- . . . für solche CFL, die durch DPDA darstellbar sind (das sind nicht alle)
- Einzelheiten: VL Compilerbau, Abschnitt syntaktische Analyse (Parsing)

Ausblick: Kombinator-Parser

- Daan Leijen et al., parsec: Monadic parser combinators, `https://hackage.haskell.org/package/parsec`
- ```
  dyck :: Parser ()
  dyck =
        do char 'a' ; dyck ; char 'b' ; dyck
    <|> return ()
  ```
  entspricht Grammatik mit den Regeln $\{(S, aSbS), (S, \epsilon)\}$
- ist eingebettete DSL (domainspezifische Sprache), die Grammatik (die Parser) sind durch Ausdrücke der Host-Sprache (Haskell) beschrieben und ihre Verknüpfung auch (durch sog. Kombinatoren).

  Im Bsp. sind das Semikolon ein Kombinator (Verkettung) und <|> (Vereinigung)
  ```

Zusatz: Anwendung: Singleton CFGs

- betrachten CFG G mit $|\text{Lang}(G)| = 1$
- Bsp: $L = \{0101001\} = \text{Lang}(G_0) = \text{Lang}(G_1)$ mit

  $G_0 = (\{0,1\}, \{S\}, \{(S \to 0101001)\})$

  $G_1 = (\{0,1\}, \{S,T\}, \{S \to TT0T, T \to 01\})$,
- (nur für diese Anwendung) Def: Größe einer CFG G ist $|G| := \sum\{|r| : (l \to r) \in R\}$ (Summe der Längen der rechten Seiten), $|G_0 = 7| > |G_1| = 4 + 2 = 6$.
- Anwendungen:
  - wörterbuchbasierte Kompression ($R =$ Wörterbuch)
  - subword tokenisation für generative KI

Singleton CFG: Beispiele

- logarithmische Kompression:
  $X_0 = a, X_1 = X_0X_0, X_2 = X_1X_1, \ldots$ mit $|X_i| = 2^i$

- Folgen von Wörtern mit interessanten Eigenschaften

  $F_0 = 0, F_1 = 1, \forall i \geq 0 : F_{i+2} = F_{i+1}F_i$; Bsp: $F_4 = 01001010$.

  kein F_i enthält 11, kein F_i enthält 000,

  $\forall i \geq 2$: F_i ohne letzte zwei Buchstaben ist Palindrom

- $X_0 = 0, Y_0 = 1, \forall i \geq 0 : X_{i+1} = X_iY_i, Y_{i+1} = Y_iX_i$

  $\lim_i X_i$ ist Lösung von

```
xs = False : tail (xs >>= \ x -> [x, not x]
```

Singleton CFG: Konstruktion

- Spezifikation: Eingabe: w, Ausgabe: CFG G
  mit Lang$(G) = \{w\}$ und $|G|$ möglichst klein
- naheliegende Implementierung (greedy): wiederholt:
  ein in w häufigstes benachbartes Paar bestimmen,
  jedes Vorkommen durch eine neue Variable ersetzen.
- Bsp: $w = 01010010010$, $|w| = 11$
  01 kommt 4-mal vor, $X \to 01, w_1 = XX0X0X0$,
  $X0$ 3-mal, $Y \to X0, w_2 = XYYY$. $|G| = 2 + 2 + 4 = 8$
- Ü: Verfahren ist nicht optimal

Aufgaben

1. einen PDA für $\{a^x b^y c^{x+y} \mid x, y \in \mathbb{N}\}$ angeben
2. die Konstruktion aus *PDA zu CFG* auf den PDA (mit 2 Zust.) für gerade Palindrome anwenden
3. Kombinator-Parser für die Lukasiewicz-Sprache mit der Grammatik $G = (\{a, b\}, \{S\}, S, \{(S, b), (S, aSS)\})$.
4. die Palindrom-Eigenschaft der Wörter F_i beweisen
5. beweise $F_{i+1} = \phi(F_i)$ mit $\phi : 0 \mapsto 1, 1 \mapsto 01$
6. für das greedy-Verfahren für Singleton CFG:
   ein w angeben, für das keine optimale SCFG entsteht
   wie groß kann der Approximationsfehler werden?

Ausblick: Berechenbarkeit

Automaten mit RAM

- (Wdhlg) NFA: endlicher Automat (ohne Speicher)
  PDA: endlicher Automat mit Kellerspeicher
- (jetzt) endlicher Automat mit RAM (random access memory, Speicher ohne Zugriffsbeschränkung)

  Zustand des Automaten z.B. realisiert als die Nr. der nächsten Anweisung (die Zeilen-Nr. im Programmtext)

  Zustand des Speichers ist Funkt. von Adresse nach Wert
- Semantik definiert mit Relation $\rightarrow_A$ auf Konfigurationen,
  $\mathsf{Lang}(A) = \{w \mid \exists c : \mathsf{encode}(w) \rightarrow_A^* c \in F\}$
- deterministisch, falls $\rightarrow_A$ links-total und rechts-eindeutig

Def. und Varianten der Berechenbarkeit

- unterschiedliche Maschinenmodelle M (Realisierung von Programm- und Speicherstruktur): u.a. Termersetzung, Lambda-Kalkül, imperativ (RAM) flach (goto), strukturiert (while), Wortersetzung (Turing-Maschine)
- Def: Fkt. $f : \mathbb{N}^k \to \mathbb{N}$ heißt M-*berechenbar*, falls $\exists A \in M$: $\forall x \in \mathbb{N}^k, y \in \mathbb{N} : f(x) = y \iff \exists c :$ encode$(x) \to_A^* c \in F \wedge$ decode$(c) = y$
- These von Church und Turing: alle vernünftigen M sind äquivalent (bestimmen die gleiche Menge berechenbarer Funktionen)
  Beweis: Compilerbau (semantik-erhaltende Übersetzung zwischen Programmen)
  diese *These* ist eigentlich die Definition von „vernünftig“

Invarianten der Berechenbarkeit (Interpreter)

- Satz: für jedes vernünftige Maschinenmodell M
  kann man einen Interpreter in M selbst schreiben
- Begründung: der Interpreter ist ein Programm
  $I(x, y) :=$ die Ausgabe von Programm x bei Eingabe y.
  I realisiert die Semantik des Programms (Programmtextes) x,
  z.B. durch eine Schleife (solange Konfiguration nicht final)
  und innerhalb davon Fallunterscheidung nach dem aktuellen Befehl in der aktuellen Konfiguration.
- das Programm für I heißt das *universelle Programm* (für M), weil es das Verhalten jedes M-Programms nachbilden kann

Grenzen der Berechenbarkeit (Halteproblem)

- Satz: die folgende Funktion ist nicht berechenbar:
  $f(x) :=$ wenn $I(x,x)\downarrow$ (Rechnung hält nach endlich vielen Schritten) dann 1, sonst ($I(x,x)\uparrow$, hält nicht) 0.
- Beweis (indirekt): wenn f berechenbar, dann gibt es ein Programm g (das benutzt f als Unterprogr.) mit

  ```
  g(x) = if f(x) then while(true){} else return 0
  ```

  Wir betrachten die Anwendung von g auf g selbst (Interpretation des Programms g bei Eingabe g)
  $I(g,g)\downarrow \iff$ der Zweig `return 0` wurde ausgeführt $\iff f(g)=0 \iff I(g,g)\uparrow$, Widerspruch.
- Def: das *Halteproblem* ist die Menge von Zahlen
  $K := \{x \mid I(x,x)\downarrow\} = \mathsf{dom}(\lambda a.I(x,x)) \subseteq \mathbb{N}$
  Satz: K ist nicht entscheidbar ($= \chi_K$ nicht berechenbar)

Ein realistisches Maschinenmodell

- Programmablaufsteuerung durch bedingte Sprünge (endlich viele Sprungziele $\Rightarrow$ endlicher Automat)
- Speicher (s: Adresse $(\mathbb{N}) \to$ Wert),
  Wertebereiche jeder einzelnen Speicherstelle:
  - $\mathbb{N}$: ist unrealistisch (denn $\mathbb{N}$ ist unbeschränkt)
  - realisitischer ist: endliche Menge Σ

  Kosten (Zeit) für einen Speicherzugriff $s[i]$
  - für jedes i sofort: ist unrealistisch (Zugriff auf große oder entfernte Adressen sollte teurer sein)
  - realistischer ist: es gibt ein *aktuelles* i, man kann $s[i]$ benutzen und dann i um 1 verschieben
- dieses Modell heißt Turing-Maschine (TM). Der Speicher ist ein Wort (Band) $s \in \Sigma^*$ mit einem Zeiger i (Kopf)

TM, Konfigurationen, CFL

- Def: Konfiguration einer TM (Band s, Kopf bei i)
  $(s[1 \ldots i-1], (s[i], q), s[i+1 \ldots n]) \in \Sigma^* \times (\Sigma \times Q) \times \Sigma^*$
- $x \in \mathsf{Lang}(M)$ gdw. existiert Konfigurationsfolge
  $\mathsf{encode}(x) = c_0 \to_M c_1 \to_M \cdots \to_M c_n \in \mathsf{final}$
- Menge dieser Folgen, notiert als $c_0|c_1^R|c_2|c_3^R| \ldots$,
  ist $\underbrace{\mathsf{encode}(x) \cdot \{c^R d \mid c \to_M d\}^*}_{\mathsf{Lang}(G_1(M,x)) \in \mathsf{CFL}} \cap \underbrace{\{cd^R \mid c \to_M d\}^* \cdot \mathsf{final}}_{\mathsf{Lang}(G_2(M)) \in \mathsf{CFL}}$
- Folgerung: für $G_1, G_2 \in$ CFG ist nicht entscheidbar,
  ob $\emptyset = \mathsf{Lang}(G_1) \cap \mathsf{Lang}(G_2)$.
  Beweis: M eine TM für $\lambda x.I(x,x)$. Dann $x \in \mathsf{dom}(M)$
  gdw. $\emptyset \neq \mathsf{Lang}(G_1(M,x)) \cap \mathsf{Lang}(G_2(M))$,
  aber $\mathsf{dom}(M)$ nicht entscheidbar, Widerspruch.
- vgl.: $\emptyset \stackrel{?}{=} \mathsf{Lang}(A_1) \cap \mathsf{Lang}(A_2)$ für $A_i \in$ NFA

Historische und didaktische Einordnung

- Alan Turing (1912–1954) *On Computable Numbers, with an Application to the Entscheidungsproblem*, 1936 (Semantik, universelle Maschine, Unentscheidbarkeit, Äquivalenz zu Berechenbarkeit im Lambda-Kalkül)
- TM: globaler Zustand (Band), globale Sprünge: unhandlich (nicht modular) (Gegensatz: λ-Kalkül)

  $\Rightarrow$ maschinenunabhängie Berechenbarkeitstheorie
- die (unverändert gültigen) Zwecke der TM:
  - Beweis der Unentscheidbarkeit von Eigenschaften formaler Sprachen
  - Ressourcen-Messung (Zeit, Platz) $\Rightarrow$ Komplexität von Algorithmen und Entscheidungsproblemen

- Quellen Biographie: https://mathshistory.st-andrews.ac.uk/Biographies/Turing/

  On Computable Numbers: http://www.comlab.ox.ac.uk/activities/ieg/e-library/sources/tp2-ie.pdf

Zusammenfassung

REG und CFL

- Definitionen (reguläre Ausdruck, kontextfreie Grammatik)
- äquivalente Definitionen (DFA, NFA, ENFA; NPDA) mit konstruktiven Beweisen (Compiler)
- Algorithmen für Wortproblem (für jede Definition), für weitere Sprach-Eigensch. (ist leer, enthält ϵ, ist endlich)
- Normalformen (Minimierung, Chomsky) mit Beweisen
- Abschluß unter Sprach-Operationen, mit
  - (falls ja) konstruktive Beweise (Algorithmen)
  - (falls nein) Gegenbsp. und Schleifensatz

Die schönsten Aufgaben

- gelöste
  - $\mathsf{Pump}_{\mathsf{REG}}(\{a^i b^j \mid i \neq j\})$
- und nicht gelöste
  - kleine CFG für $\Sigma^* \setminus \{w^2 \mid w \in \Sigma^*\}$
  - CFG für allgemeingültige aussagenlogische Formeln
- und nicht gestellte:

  die Sprache $L \subseteq \{0,1\}^*$ der Wörter, die durch Solitär-Regeln $\{110 \to 001, 011 \to 100\}$ bis auf einen Steine abgeräumt werden können: ist regulär
- (die sind alle zu schwer für die Klausur)

Plan

- KW 14: Einleitung, Wörter, Sprachen
- KW 15: reguläre Ausdrücke, Kleene-Algebra, Äquivalenzen
- KW 16: deterministische endliche Automaten, Abschluß-Eigenschaften
- KW 17: nichtdeterministische Automaten, Potenzmengen-K., Potenzmenge on-the-fly
- KW 18: Umrechnung zwischen Automat und reg. Ausdruck, NFA mit spontanen Übergängen
- KW 19: Nerode-Kongruenz, Minimierung von Automaten.

- KW 21: Schleifensatz
- KW 22: Morphismen, Zusatz: Spursprachen von Programmen
- KW 23: Wortersetzung, Grammatiken, CFG
- KW 24: Chomsky-NF, CYK
- KW 25: Schnitt mit REG, Schleifensatz, Nicht-Abschluß-Eig.
- KW 26: Kellerautomaten (Parser), Maschinenmodelle
- KW 27: Zusammenfassung, Ausblick Berechenbarkeit, nicht entscheidbare Probleme für CFG,