Einführung in die Theoretische Informatik

Stefan Kratsch

Institut für Informatik
Humboldt-Universität zu Berlin

WS 2019/20

Inhalt der Vorlesung

Themen dieser VL:

- **Welche Rechenmodelle eignen sich zur Lösung welcher algorithmischen Problemstellungen?** **Automatentheorie**
- **Welche algorithmischen Probleme sind überhaupt lösbar?** **Berechenbarkeitstheorie**
- **Welcher Aufwand ist zur Lösung eines geg. algorithmischen Problems nötig?** **Komplexitätstheorie**

Themen der VL Algorithmen und Datenstrukturen:

- Wie lassen sich praktisch relevante Problemstellungen möglichst effizient lösen? Algorithmik

Themen der VL Logik in der Informatik:

- Mathem. Grundlagen der Informatik, Beweise führen, Modellierung
  Aussagenlogik, Prädikatenlogik

Lernziele

- Überblick über die wichtigsten Rechenmodelle (Automaten) wie z.B.
  - endliche Automaten
  - Kellerautomaten
  - Turingmaschinen
  - Registermaschinen
  - Schaltkreise
- Charakterisierung der Klassen aller mit diesen Rechenmodellen lösbaren Probleme durch
  - unterschiedliche Typen von formalen Grammatiken
  - Abschlusseigenschaften unter geeigneten Sprachoperationen
  - Reduzierbarkeit auf typische Probleme (Vollständigkeit)
- Erkennen von Grenzen der Berechenbarkeit
- Klassifikation wichtiger algorithmischer Probleme nach ihrer Komplexität

Maschinenmodelle

- Rechenmaschinen spielen in der Informatik eine zentrale Rolle
- Es gibt viele unterschiedliche math. Modelle für Rechenmaschinen
- Diese können sich in ihrer Berechnungskraft unterscheiden
- Die Turingmaschine (TM) ist ein universales Berechnungsmodell, da sie alle anderen bekannten Rechenmodelle simulieren kann
- Wir betrachten zunächst Einschränkungen des TM-Modells, die vielfältige praktische Anwendungen haben, wie z.B.
  - endliche Automaten (DFA, NFA)
  - Kellerautomaten (PDA, DPDA) etc.

Der Algorithmenbegriff

- Der Begriff Algorithmus geht auf den persischen Gelehrten Muhammed Al Chwarizmi (8./9. Jhd.) zurück
- Ältester bekannter nicht-trivialer Algorithmus:
  Euklidischer Algorithmus zur Berechnung des ggT (300 v. Chr.)
- Von einem Algorithmus wird erwartet, dass er bei jeder zulässigen Problemeingabe nach endlich vielen Rechenschritten eine korrekte Ausgabe liefert
- Eine wichtige Rolle spielen Entscheidungsprobleme, bei denen jede Eingabe nur mit ja oder nein beantwortet wird
- Die (maximale) Anzahl der Rechenschritte bei allen möglichen Eingaben ist nicht beschränkt, d.h. mit wachsender Eingabelänge kann auch die Rechenzeit beliebig anwachsen
- Die Beschreibung eines Algorithmus muss jedoch endlich sein
- Problemeingaben können Zahlen, Formeln, Graphen etc. sein
- Diese werden über einem Eingabealphabet Σ kodiert

Alphabet, Wort, Sprache

Definition

- Ein Alphabet ist eine geordnete endliche Menge

  $$\Sigma = \{a_1, \ldots, a_m\}, \; m \geq 1$$

  von Zeichen a_i
- Eine Folge $x = x_1 \ldots x_n \in \Sigma^n$ von Zeichen heißt Wort
- Die Länge von $x = x_1 \ldots x_n \in \Sigma^n$ ist n und wird mit $|x|$ bezeichnet
- Die Menge aller Wörter über Σ ist

  $$\Sigma^* = \bigcup_{n \geq 0} \Sigma^n$$
- Das (einzige) Wort der Länge $n = 0$ ist das leere Wort, welches wir mit ε bezeichnen, d.h. $\Sigma^0 = \{\varepsilon\}$
- Jede Teilmenge $L \subseteq \Sigma^*$ heißt Sprache über dem Alphabet Σ

Beispiele für Sprachen

Beispiel

- Sprachen über Σ sind beispielsweise $\emptyset, \Sigma^*, \Sigma$ und $\{\varepsilon\}$
- $\emptyset$ enthält keine Wörter und heißt leere Sprache
- Σ^* enthält dagegen alle Wörter über Σ
- Σ enthält alle Wörter über Σ der Länge 1
- $\{\varepsilon\}$ enthält nur das leere Wort, ist also einelementig
- Sprachen, die genau ein Wort enthalten, werden auch als Singletonsprachen bezeichnet
- in der Informatik spielen Programmiersprachen eine wichtige Rolle

Operationen auf Sprachen

- Da Sprachen Mengen sind, können wir sie bzgl. Inklusion vergleichen
- Zum Beispiel gilt $\varnothing \subseteq \{\varepsilon\} \subseteq \Sigma^*$
- Wir können Sprachen auch vereinigen, schneiden und komplementieren
- Seien A und B Sprachen über Σ. Dann ist
  - $A \cap B = \{x \in \Sigma^* \mid x \in A \wedge x \in B\}$ der Schnitt von A und B
  - $A \cup B = \{x \in \Sigma^* \mid x \in A \vee x \in B\}$ die Vereinigung von A und B, und
  - $\overline{A} = \{x \in \Sigma^* \mid x \notin A\}$ das Komplement von A

Konkatenation von Wörtern

Definition

Die Konkatenation von zwei Wörtern $x = x_1 \dots x_n$ und $y = y_1 \dots y_m$ ist das Wort $x \circ y = x_1 \dots x_n y_1 \dots y_m$, das wir auch einfach mit xy bezeichnen

Beispiel

- Für $x = aba$ und $y = abab$ erhalten wir $xy = abaabab$ und $yx = abababa$
- Die Konkatenation ist also nicht kommutativ
- Allerdings ist $\circ$ assoziativ, d.h. es gilt $x(yz) = (xy)z$
  Daher können wir hierfür auch einfach xyz schreiben
- Es gibt auch ein neutrales Element, da $x\varepsilon = \varepsilon x = x$ ist
- Eine algebraische Struktur $(M, \square, e)$ mit einer assoziativen Operation $\square : M \times M \to M$ und einem neutralen Element e heißt Monoid
- $(\Sigma^*, \circ, \varepsilon)$ ist also ein Monoid

Spezielle Sprachoperationen

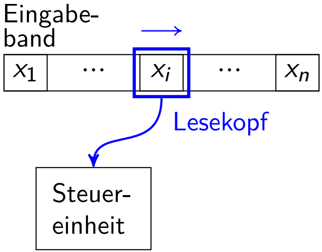
Neben den Mengenoperationen Schnitt, Vereinigung und Komplement gibt es auch spezielle Sprachoperationen

Definition

- Das Produkt (Verkettung, Konkatenation) der Sprachen A und B ist
  $$AB = \{xy \mid x \in A, y \in B\}$$
- Ist $A = \{x\}$ eine Singletonsprache, so schreiben wir für $\{x\}B$ auch einfach xB
- Die n-fache Potenz A^n einer Sprache A ist induktiv definiert durch
  $$A^n = \begin{cases} \{\varepsilon\}, & n = 0, \\ A^{n-1}A, & n > 0 \end{cases}$$
- Die Sternhülle von A ist $A^* = \bigcup_{n \geq 0} A^n$
- Die Plushülle von A ist $A^+ = \bigcup_{n \geq 1} A^n = AA^*$

Algorithmische Erkennung von Sprachen

- Ein einfaches Rechenmodell zum Erkennen von Sprachen ist der endliche Automat:



- Ein endlicher Automat
  - nimmt zu jedem Zeitpunkt genau einen von endlich vielen Zuständen an
  - macht bei Eingaben der Länge n genau n Rechenschritte und
  - liest in jedem Schritt genau ein Eingabezeichen

Formale Definition eines endlichen Automaten

Definition

- Ein endlicher Automat (kurz: DFA; *Deterministic Finite Automaton*) wird durch ein 5-Tupel $M = (Z, \Sigma, \delta, q_0, E)$ beschrieben, wobei
  - $Z \neq \varnothing$ eine endliche Menge von Zuständen
  - Σ das Eingabealphabet
  - $\delta : Z \times \Sigma \to Z$ die Überführungsfunktion
  - $q_0 \in Z$ der Startzustand und
  - $E \subseteq Z$ die Menge der Endzustände ist
- Die von M akzeptierte (oder erkannte) Sprache ist

$$L(M) = \left\{ x_1 \dots x_n \in \Sigma^* \;\middle|\; \begin{array}{l} \text{es gibt } q_1, \dots, q_{n-1} \in Z, q_n \in E \text{ mit} \\ \delta(q_i, x_{i+1}) = q_{i+1} \text{ für } i = 0, \dots, n-1 \end{array} \right\}$$

- Eine Zustandsfolge $q_0, q_1, \dots, q_n$ heißt Rechnung von $M(x_1 \dots x_n)$, falls $\delta(q_i, x_{i+1}) = q_{i+1}$ für $i = 0, \dots, n-1$ gilt
- Sie heißt akzeptierend, falls $q_n \in E$ ist, und andernfalls verwerfend

Die Klasse der regulären Sprachen

Frage

Welche Sprachen lassen sich durch endliche Automaten erkennen und welche nicht?

Definition

Eine von einem DFA akzeptierte Sprache wird als regulär bezeichnet. Die zugehörige Sprachklasse ist

$$\mathrm{REG} = \{L(M) \mid M \text{ ist ein DFA}\}$$

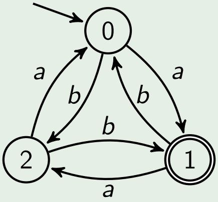
DFAs beherrschen Modulare Arithmetik

Beispiel

Sei $M_3 = (Z, \Sigma, \delta, 0, E)$ ein DFA mit $Z = \{0, 1, 2\}$, $\Sigma = \{a, b\}$, $E = \{1\}$ und der Überführungsfunktion

δ	0	1	2
a	1	2	0
b	2	0	1

Graphische Darstellung:



Endzustände werden durch einen doppelten Kreis und der Startzustand wird durch einen Pfeil gekennzeichnet ◁

Frage: Welche Wörter akzeptiert M_3?

- Ist $w_1 = aba \in L(M_3)$? Ja (akzeptierende Rechnung: $0, 1, 0, 1$)
- Ist $w_2 = abba \in L(M_3)$? Nein (verwerfende Rechnung: $0, 1, 0, 2, 0$)

DFAs beherrschen Modulare Arithmetik

Behauptung

Die von M_3 erkannte Sprache ist

$$L(M_3) = \{x \in \{a, b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 1\}, \quad \text{wobei}$$

- $\#_a(x)$ die Anzahl der Vorkommen von a in x bezeichnet und
- $i \equiv_m j$ (in Worten: i ist kongruent zu j modulo m) bedeutet, dass $i - j$ durch m teilbar ist

Beweis der Behauptung durch Induktion über die Länge von x

Wir betrachten zunächst das Erreichbarkeitsproblem für DFAs

Das Erreichbarkeitsproblem für DFAs

Frage

Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA und sei $x = x_1 \dots x_n \in \Sigma^*$. Welchen Zustand erreicht M nach Lesen der Eingabe x?

Antwort

- nach 0 Schritten: den Startzustand q_0
- nach 1 Schritt: den Zustand $\delta(q_0, x_1)$
- nach 2 Schritten: den Zustand $\delta(\delta(q_0, x_1), x_2)$

  $\vdots$
- nach n Schritten: den Zustand $\delta(\dots\delta(\delta(q_0, x_1), x_2), \dots x_n)$

Definition

- Bezeichne $\hat{\delta}(q, x)$ denjenigen Zustand, in dem sich M nach Lesen von x befindet, wenn M im Zustand q gestartet wird
- Dann können wir die Funktion

  $$\hat{\delta} : Z \times \Sigma^* \to Z$$

  induktiv über die Länge von x wie folgt definieren:
  Für $q \in Z$, $x \in \Sigma^*$ und $a \in \Sigma$ sei

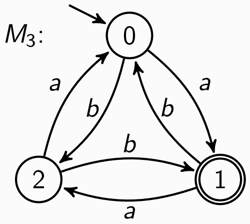
  $$\begin{aligned} \hat{\delta}(q, \varepsilon) &= q, \\ \hat{\delta}(q, xa) &= \delta(\hat{\delta}(q, x), a) \end{aligned}$$

- Die von M erkannte Sprache lässt sich nun elegant durch

  $$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(q_0, x) \in E\}$$

  beschreiben

DFAs beherrschen Modulare Arithmetik



Behauptung

Für alle $x \in \{a, b\}^*$ gilt:

$$x \in L(M_3) \Leftrightarrow \#_a(x) - \#_b(x) \equiv_3 1$$

Beweis

- 1 ist der einzige Endzustand von M
- Daher ist $L(M_3) = \{x \in \{a, b\}^* \mid \hat{\delta}(0, x) = 1\}$
- Obige Behauptung ist also äquivalent zu

$$\forall x \in \{a, b\}^*\colon \hat{\delta}(0, x) = 1 \Leftrightarrow \#_a(x) - \#_b(x) \equiv_3 1$$

- Folglich reicht es, für alle $x \in \{a, b\}^*$ folgende Kongruenz zu zeigen:

$$\hat{\delta}(0, x) \equiv_3 \#_a(x) - \#_b(x)$$

DFAs beherrschen Modulare Arithmetik

Induktionsbehauptung: Für alle $x \in \{a,b\}^n$ gilt $\hat{\delta}(0,x) \equiv_3 \#_a(x) - \#_b(x)$

Induktionsanfang ($n = 0$): klar, da $\hat{\delta}(0,\varepsilon) = \#_a(\varepsilon) = \#_b(\varepsilon) = 0$ ist

Induktionsschritt ($n \rightsquigarrow n+1$): Sei $x = x_1 \dots x_{n+1} \in \{a,b\}^{n+1}$ gegeben

- Nach Induktionsvoraussetzung (IV) gilt für $x' = x_1 \dots x_n$:

$$\hat{\delta}(0,x') \equiv_3 \#_a(x') - \#_b(x')$$

- Zudem gilt

$$\begin{aligned}\delta(i, x_{n+1}) &\equiv_3 \begin{cases} i+1, & x_{n+1} = a \\ i-1, & x_{n+1} = b \end{cases} \\ &= i + \#_a(x_{n+1}) - \#_b(x_{n+1}) && (*)\end{aligned}$$

- Somit folgt

$$\begin{aligned}\hat{\delta}(0,x) &= \delta(\hat{\delta}(0,x'), x_{n+1}) \\ &\equiv_3 \hat{\delta}(0,x') + \#_a(x_{n+1}) - \#_b(x_{n+1}) && (*) \\ &\equiv_3 \#_a(x') - \#_b(x') + \#_a(x_{n+1}) - \#_b(x_{n+1}) && (IV) \\ &\equiv_3 \#_a(x) - \#_b(x)\end{aligned}$$

□

Singletonsprachen sind regulär

Vereinbarung

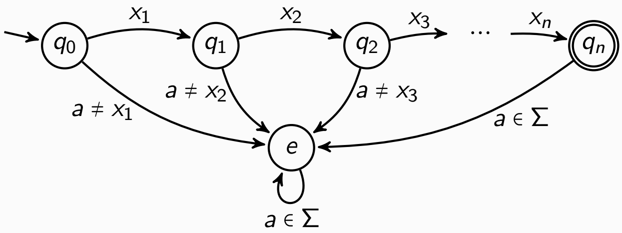
Für das Folgende sei $\Sigma = \{a_1, \ldots, a_m\}$ ein fest gewähltes Alphabet

Beobachtung 1

Alle Sprachen, die nur ein Wort $x = x_1 \ldots x_n \in \Sigma^*$ enthalten, sind regulär

Beweis

Folgender DFA M erkennt die Sprache $L(M) = \{x\}$:



□

REG ist unter Komplement abgeschlossen

Beobachtung 2

Ist $L \in \mathrm{REG}$, so ist auch die Sprache $\overline{L} = \Sigma^* \setminus L$ regulär

Beweis

- Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA mit $L(M) = L$
- Dann wird das Komplement $\overline{L}$ von L von dem DFA $\overline{M} = (Z, \Sigma, \delta, q_0, Z \setminus E)$ akzeptiert

$\square$

Definition

Für eine Sprachklasse $\mathcal{C}$ bezeichne *co*-$\mathcal{C}$ die Klasse $\{\overline{L} \mid L \in \mathcal{C}\}$ aller Komplemente von Sprachen in $\mathcal{C}$

Korollar

co-REG = REG

REG ist unter Schnitt abgeschlossen

Beobachtung 3

Sind $L_1, L_2 \in \mathrm{REG}$, so ist auch die Sprache $L_1 \cap L_2$ regulär

Beweis

- Seien $M_i = (Z_i, \Sigma, \delta_i, q_i, E_i)$, $i = 1, 2$, DFAs mit $L(M_i) = L_i$.
- Dann wird der Schnitt $L_1 \cap L_2$ von dem DFA

  $$M = (Z_1 \times Z_2, \Sigma, \delta, (q_1, q_2), E_1 \times E_2)$$

  mit

  $$\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$$

  erkannt.
- M wird auch als Kreuzproduktautomat bezeichnet

□

REG ist unter Vereinigung abgeschlossen

Beobachtung 4

Die Vereinigung $L_1 \cup L_2$ von regulären Sprachen L_1 und L_2 ist regulär

Beweis

Es gilt $L_1 \cup L_2 = \overline{(\overline{L_1} \cap \overline{L_2})}$. $\square$

Frage

Wie sieht der zugehörige DFA aus?

Antwort

$$M' = (Z_1 \times Z_2, \Sigma, \delta, (q_1, q_2), (E_1 \times Z_2) \cup (Z_1 \times E_2)).$$

Abschlusseigenschaften von Sprachklassen

Definition

- Ein (k-stelliger) Sprachoperator ist eine Abbildung *op*, die k Sprachen $L_1, \ldots, L_k$ auf eine Sprache $op(L_1, \ldots, L_k)$ abbildet.
- Eine Sprachklasse $\mathcal{K}$ heißt unter *op* abgeschlossen, wenn gilt:

$$L_1, \ldots, L_k \in \mathcal{K} \Rightarrow op(L_1, \ldots, L_k) \in \mathcal{K}$$

- Der Abschluss von $\mathcal{K}$ unter *op* ist die (bzgl. Inklusion) kleinste Sprachklasse $\mathcal{K}'$, die $\mathcal{K}$ enthält und unter *op* abgeschlossen ist

Beispiel

- Der 2-stellige Schnittoperator $\cap$ bildet L_1 und L_2 auf $L_1 \cap L_2$ ab.
- Der Abschluss der Singletonsprachen unter $\cap$ besteht aus allen Singletonsprachen und der leeren Sprache.
- Der Abschluss der Singletonsprachen unter $\cup$ besteht aus allen nichtleeren endlichen Sprachen.
- Der Abschluss der Singletonsprachen unter $\cap$, $\cup$ und Komplement besteht aus allen endlichen und co-endlichen Sprachen. $\triangleleft$

REG ist unter Mengenoperationen abgeschlossen

Korollar

Die Klasse REG der regulären Sprachen ist unter folgenden Operationen abgeschlossen:

- Komplement
- Schnitt
- Vereinigung

Wie umfangreich ist REG?

Folgerung

- Aus den Beobachtungen folgt, dass alle endlichen und alle co-endlichen Sprachen regulär sind
- Da die reguläre Sprache

  $$L(M_3) = \{x \in \{a, b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 1\}$$

  weder endlich noch co-endlich ist, haben wir damit allerdings noch nicht alle regulären Sprachen erfasst

Wie umfangreich ist REG?

Nächstes Ziel

Zeige, dass REG unter Produktbildung und Sternhülle abgeschlossen ist

Problem

Bei der Konstruktion eines DFA für das Produkt L_1L_2 bereitet es Schwierigkeiten, den richtigen Zeitpunkt für das Ende der Simulation von M_1 und den Start der Simulation von M_2 zu finden

Lösungsidee

Ein nichtdeterministischer Automat (NFA) kann den richtigen Zeitpunkt „raten“

Verbleibendes Problem

Zeige, dass auch NFAs nur reguläre Sprachen erkennen

Nichtdeterministische endliche Automaten

Definition

- Ein nichtdet. endl. Automat (kurz: NFA; *Nondet. Finite Automaton*)

  $$N = (Z, \Sigma, \Delta, Q_0, E)$$

  ist genau so aufgebaut wie ein DFA, nur dass er
  - eine Menge $Q_0 \subseteq Z$ von Startzuständen hat und
  - die Überführungsfunktion folgende Form hat

    $$\Delta : Z \times \Sigma \to \mathcal{P}(Z)$$

  Hierbei bezeichnet $\mathcal{P}(Z)$ die Potenzmenge (also die Menge aller Teilmengen) von Z; diese wird oft auch mit 2^Z bezeichnet
- Die von einem NFA N akzeptierte (oder erkannte) Sprache ist

  $$L(N) = \left\{ x_1 \dots x_n \in \Sigma^* \;\middle|\; \begin{array}{c} \text{es gibt } q_0 \in Q_0, q_1, \dots, q_{n-1} \in Z, q_n \in E \\ \text{mit } q_{i+1} \in \Delta(q_i, x_{i+1}) \text{ für } i = 0, \dots, n-1 \end{array} \right\}$$

- Eine Zustandsfolge $q_0, \dots, q_n$ heißt Rechnung von $N(x_1 \dots x_n)$, falls $q_0 \in Q_0$ und $q_{i+1} \in \Delta(q_i, x_{i+1})$ für $i = 0, \dots, n-1$ gilt

Eigenschaften von NFAs

- Ein NFA N kann bei einer Eingabe x also nicht nur eine, sondern mehrere verschiedene Rechnungen parallel ausführen
- Ein Wort x gehört genau dann zu $L(N)$, wenn $N(x)$ mindestens eine akzeptierende Rechnung hat
- Im Gegensatz zu einem DFA, der jede Eingabe zu Ende liest, kann ein NFA N „stecken bleiben"
- Dieser Fall tritt ein, wenn N in einen Zustand q gelangt, in dem er das nächste Eingabezeichen x_i wegen

  $$\Delta(q, x_i) = \varnothing$$

  nicht verarbeiten kann

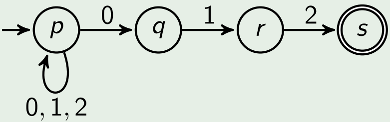
Eigenschaften von NFAs

Beispiel

- Betrachte den NFA $N = (Z, \Sigma, \Delta, Q_0, E)$ mit $Z = \{p, q, r, s\}$, $\Sigma = \{0, 1, 2\}$, $Q_0 = \{p\}$, $E = \{s\}$ und der Überführungsfunktion

Δ	p	q	r	s
0	$\{p, q\}$	$\varnothing$	$\varnothing$	$\varnothing$
1	$\{p\}$	$\{r\}$	$\varnothing$	$\varnothing$
2	$\{p\}$	$\varnothing$	$\{s\}$	$\varnothing$

Graphische Darstellung:



- Ist $w_1 = 012 \in L(N)$? Ja (akzeptierende Rechnung: p, q, r, s)
  Es gibt aber auch verwerfende Rechnungen bei Eingabe w_1: p, p, p, p
- Ist $w_2 = 021 \in L(N)$? Nein, da es keine akzeptierenden Rechnungen gibt
- Es gilt $L(N) = \{x012 \mid x \in \Sigma^*\}$ ◁

Ein NFA für das Produkt von regulären Sprachen

Beobachtung 5

Seien $N_i = (Z_i, \Sigma, \Delta_i, Q_i, E_i)$ NFAs mit $L(N_i) = L_i$ für $i = 1, 2$. Dann wird auch das Produkt $L_1 L_2$ von einem NFA erkannt

Beweis

- Wir können $Z_1 \cap Z_2 = \varnothing$ annehmen
- Dann gilt $L(N) = L_1 L_2$ für den NFA $N = (Z_1 \cup Z_2, \Sigma, \Delta, Q_1, E)$ mit

$$\Delta(p, a) = \begin{cases} \Delta_1(p, a), & p \in Z_1 \setminus E_1, \\ \Delta_1(p, a) \cup \bigcup_{q \in Q_2} \Delta_2(q, a), & p \in E_1, \\ \Delta_2(p, a), & p \in Z_2 \end{cases}$$

und

$$E = \begin{cases} E_2, & Q_2 \cap E_2 = \varnothing, \\ E_1 \cup E_2, & \text{sonst} \end{cases}$$

Ein NFA für das Produkt von regulären Sprachen

- Dann gilt $L(N) = L_1 L_2$ für den NFA $N = (Z_1 \cup Z_2, \Sigma, \Delta, Q_1, E)$ mit

$$\Delta(p,a) = \begin{cases} \Delta_1(p,a), & p \in Z_1 \setminus E_1, \\ \Delta_1(p,a) \cup \bigcup_{q \in Q_2} \Delta_2(q,a), & p \in E_1, \\ \Delta_2(p,a), & p \in Z_2 \end{cases}$$

und $E = E_2$, falls $Q_2 \cap E_2 = \emptyset$, bzw. $E = E_1 \cup E_2$ sonst.

Beweis von $L_1 L_2 \subseteq L(N)$:

Seien $x = x_1 \cdots x_k \in L_1, y = y_1 \cdots y_l \in L_2$ und seien $q_0, \ldots, q_k$ und $p_0, \ldots, p_l$ akzeptierende Rechnungen von $N_1(x)$ und $N_2(y)$

Dann ist $q_0, \ldots, q_k, p_1, \ldots, p_l$ eine akz. Rechnung von $N(xy)$, da

- $q_0 \in Q_1$ und $p_l \in E_2$ ist, und
- im Fall $l \geq 1$ wegen $q_k \in E_1$, $p_0 \in Q_2$ und $p_1 \in \Delta_2(p_0, y_1)$ zudem $p_1 \in \Delta(q_k, y_1)$ und
- im Fall $l = 0$ wegen $q_k \in E_1$ und $p_l \in Q_2 \cap E_2$ zudem $q_k \in E$ ist

Ein NFA für das Produkt von regulären Sprachen

- Dann gilt $L(N) = L_1 L_2$ für den NFA $N = (Z_1 \cup Z_2, \Sigma, \Delta, Q_1, E)$ mit

$$\Delta(p,a) = \begin{cases} \Delta_1(p,a), & p \in Z_1 \setminus E_1, \\ \Delta_1(p,a) \cup \bigcup_{q \in Q_2} \Delta_2(q,a), & p \in E_1, \\ \Delta_2(p,a), & p \in Z_2 \end{cases}$$

und $E = E_2$, falls $Q_2 \cap E_2 = \varnothing$, bzw. $E = E_1 \cup E_2$ sonst.

Beweis von $L(N) \subseteq L_1 L_2$:

Sei $x = x_1 \cdots x_n \in L(N)$ und sei $q_0, \ldots, q_n$ eine akz. Rechnung von $N(x)$

Dann gilt $q_0 \in Q_1$, $q_n \in E$, $q_0, \ldots, q_i \in Z_1$ und $q_{i+1}, \ldots, q_n \in Z_2$ für ein $i \leq n$

Wir zeigen, dass $q_0, \ldots, q_i$ eine akz. Rechnung von $N_1(x_1 \cdots x_i)$ und $q, q_{i+1}, \ldots, q_n$ für ein $q \in Q_2$ eine akz. Rechnung von $N_2(x_{i+1} \cdots x_n)$ ist:

- Im Fall $i < n$ impliziert der Übergang $q_{i+1} \in \Delta(q_i, x_{i+1})$, dass $q_i \in E_1$ und $q_{i+1} \in \Delta_2(q, x_{i+1})$ für ein $q \in Q_2$ ist. Zudem ist $q_n \in E \cap Z_2 = E_2$
- Im Fall $i = n$ ist $q_n \in E \cap Z_1$, was $q_n \in E_1$ und $Q_2 \cap E_2 \neq \varnothing$ impliziert

□

Ein NFA für die Sternhülle einer regulären Sprache

Beobachtung 6

Ist $N = (Z, \Sigma, \Delta, Q_0, E)$ ein NFA, so wird auch die Sprache $L(N)^*$ von einem NFA erkannt

Beweis

Die Sprache $L(N)^*$ wird von dem NFA

$$N' = (Z \cup \{q_{neu}\}, \Sigma, \Delta', Q_0 \cup \{q_{neu}\}, E \cup \{q_{neu}\})$$

mit

$$\Delta'(p,a) = \begin{cases} \Delta(p,a), & p \in Z \setminus E, \\ \Delta(p,a) \cup \bigcup_{q \in Q_0} \Delta(q,a), & p \in E, \\ \varnothing, & p = q_{neu} \end{cases}$$

erkannt. $\square$

Überblick

Ziel

Zeige, dass REG unter Produktbildung und Sternhülle abgeschlossen ist

Problem

Bei der Konstruktion eines DFA für das Produkt $L_1 L_2$ bereitet es Schwierigkeiten, den richtigen Zeitpunkt für den Übergang von (der Simulation von) M_1 zu M_2 zu finden

Lösungsidee (bereits umgesetzt)

Ein nichtdeterministischer Automat (NFA) kann den richtigen Zeitpunkt für den Übergang „raten“

Noch zu zeigen

NFAs erkennen genau die regulären Sprachen

NFAs erkennen genau die regulären Sprachen

Satz (Rabin und Scott)

$\mathrm{REG} = \{L(N) \mid N \text{ ist ein NFA}\}$

Beweis von $\mathrm{REG} \subseteq \{L(N) \mid N \text{ ist ein NFA}\}$

Diese Inklusion ist klar, da jeder DFA $M = (Z, \Sigma, \delta, q_0, E)$ in einen äquivalenten NFA

$$N = (Z, \Sigma, \Delta, Q_0, E)$$

transformiert werden kann, indem wir $\Delta(q, a) = \{\delta(q, a)\}$ und $Q_0 = \{q_0\}$ setzen. $\square$

Für die umgekehrte Inklusion ist das Erreichbarkeitsproblem für NFAs von zentraler Bedeutung

Das Erreichbarkeitsproblem für NFAs

Frage

Sei $N = (Z, \Sigma, \Delta, Q_0, E)$ ein NFA und sei $x = x_1 \dots x_n$ eine Eingabe. Welche Zustände sind in i Schritten erreichbar?

Antwort

- in 0 Schritten: alle Zustände in Q_0
- in einem Schritt: alle Zustände in

$$Q_1 = \bigcup_{q \in Q_0} \Delta(q, x_1)$$

- in i Schritten: alle Zustände in

$$Q_i = \bigcup_{q \in Q_{i-1}} \Delta(q, x_i)$$

Simulation von NFAs durch DFAs

Idee

- Wir können einen NFA $N = (Z, \Sigma, \Delta, Q_0, E)$ durch einen DFA $M = (Z', \Sigma, \delta, q_0', E')$ simulieren, der in seinem Zustand die Information speichert, in welchen Zuständen sich N momentan befinden könnte
- Die Zustände von M sind also Teilmengen Q von Z (d.h. $Z' = \mathcal{P}(Z)$) mit Q_0 als Startzustand (d.h. $q_0' = Q_0$) und der Endzustandsmenge

$$E' = \{Q \subseteq Z \mid Q \cap E \neq \varnothing\}$$

- Die Überführungsfunktion $\delta : \mathcal{P}(Z) \times \Sigma \to \mathcal{P}(Z)$ von M berechnet dann für einen Zustand $Q \subseteq Z$ und ein Zeichen $a \in \Sigma$ die Menge

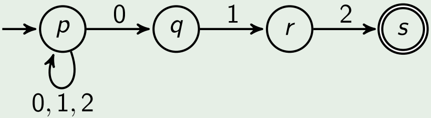
$$\delta(Q, a) = \bigcup_{q \in Q} \Delta(q, a)$$

  aller Zustände, in die N gelangen kann, wenn N ausgehend von einem beliebigen Zustand $q \in Q$ das Zeichen a liest
- M wird auch als der zu N gehörige Potenzmengenautomat bezeichnet

Simulation von NFAs durch DFAs

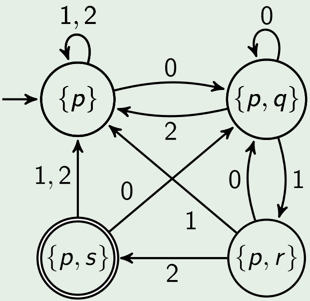
Beispiel

- Betrachte den NFA N



- Ausgehend von $Q_0 = \{p\}$ liefert δ dann die folgenden Werte:

δ	0	1	2
$\{p\}$	$\{p,q\}$	$\{p\}$	$\{p\}$
$\{p,q\}$	$\{p,q\}$	$\{p,r\}$	$\{p\}$
$\{p,r\}$	$\{p,q\}$	$\{p\}$	$\{p,s\}$
$\{p,s\}$	$\{p,q\}$	$\{p\}$	$\{p\}$



◁

Simulation von NFAs durch DFAs

Bemerkung

- Im obigen Beispiel werden für die Konstruktion des Potenzmengenautomaten nur 4 der insgesamt

  $$\|\mathcal{P}(Z)\| = 2^{\|Z\|} = 2^4 = 16$$

  Zustände benötigt, da die übrigen 12 Zustände nicht erreichbar sind. (Hierbei bezeichnet $\|A\|$ die Mächtigkeit einer Menge A.)
- Es gibt jedoch Beispiele, bei denen alle $2^{\|Z\|}$ Zustände benötigt werden (siehe Übungen)

NFAs erkennen genau die regulären Sprachen

Beweis von $\{L(N) \mid N \text{ ist ein NFA}\} \subseteq \mathrm{REG}$

- Sei $N = (Z, \Sigma, \Delta, Q_0, E)$ ein NFA und sei $M = (\mathcal{P}(Z), \Sigma, \delta, Q_0, E')$ der zugehörige Potenzmengenautomat mit $\delta(Q, a) = \bigcup_{q \in Q} \Delta(q, a)$ und $E' = \{Q \subseteq Z \mid Q \cap E \neq \varnothing\}$
- Dann folgt die Korrektheit von M mittels folgender Behauptung, die wir auf der nächsten Folie beweisen.

Behauptung

$\hat{\delta}(Q_0, x)$ enthält genau die von N nach Lesen von x erreichbaren Zustände

- Für alle Wörter $x \in \Sigma^*$ gilt

$$
\begin{aligned}
x \in L(N) \quad &\Leftrightarrow \quad N \text{ kann nach Lesen von } x \text{ einen Endzustand erreichen} \\
&\overset{\textit{Beh.}}{\Leftrightarrow} \quad \hat{\delta}(Q_0, x) \cap E \neq \varnothing \\
&\Leftrightarrow \quad \hat{\delta}(Q_0, x) \in E' \\
&\Leftrightarrow \quad x \in L(M)
\end{aligned}
$$

$\square$

Beweis der Behauptung

Behauptung

$\hat{\delta}(Q_0, x)$ enthält genau die von N nach Lesen von x erreichbaren Zustände

Beweis durch Induktion über die Länge n von x

$n = 0$: klar, da $\hat{\delta}(Q_0, \varepsilon) = Q_0$ ist

$n \rightsquigarrow n + 1$: Sei $x = x_1 \dots x_{n+1}$ gegeben. Nach IV enthält

$$Q_n = \hat{\delta}(Q_0, x_1 \dots x_n)$$

die Zustände, die N nach Lesen von $x_1 \dots x_n$ erreichen kann. Wegen

$$\hat{\delta}(Q_0, x) = \delta(Q_n, x_{n+1}) = \bigcup_{q \in Q_n} \Delta(q, x_{n+1})$$

enthält dann aber $\hat{\delta}(Q_0, x)$ die Zustände, die N nach Lesen von x erreichen kann. □

Abschlusseigenschaften der Klasse REG

Satz (Rabin und Scott)

$\mathrm{REG} = \{L(N) \mid N \text{ ist ein NFA}\}$

Korollar

Die Klasse REG der regulären Sprachen ist unter folgenden Operationen abgeschlossen:

- Komplement
- Schnitt
- Vereinigung
- Produkt
- Sternhülle

Überblick

Nächstes Ziel

Zeige, dass REG als Abschluss der endlichen Sprachen unter Vereinigung, Produkt und Sternhülle charakterisierbar ist

Bereits gezeigt:

Jede Sprache, die mittels der Operationen Vereinigung, Produkt und Sternhülle (sowie Schnitt und Komplement) angewandt auf endliche Sprachen darstellbar ist, ist regulär

Noch zu zeigen:

Jede reguläre Sprache lässt sich aus endlichen Sprachen mittels Vereinigung, Produkt und Sternhülle erzeugen

Konstruktive Charakterisierung von REG

Induktive Definition der Menge RA_Σ aller regulären Ausdrücke über Σ

Die Symbole $\varnothing$, ϵ und a ($a \in \Sigma$) sind reguläre Ausdrücke über Σ, die

- die leere Sprache $L(\varnothing) = \varnothing$
- die Sprache $L(\epsilon) = \{\varepsilon\}$ und
- für jedes $a \in \Sigma$ die Sprache $L(a) = \{a\}$ beschreiben

Sind α und β reguläre Ausdrücke über Σ, die die Sprachen $L(\alpha)$ und $L(\beta)$ beschreiben, so sind auch $\alpha\beta$, $(\alpha|\beta)$ und $(\alpha)^*$ reguläre Ausdrücke über Σ, die folgende Sprachen beschreiben:

- $L(\alpha\beta) = L(\alpha)L(\beta)$
- $L\big((\alpha|\beta)\big) = L(\alpha) \cup L(\beta)$
- $L\big((\alpha)^*\big) = L(\alpha)^*$

Bemerkung

RA_Σ ist eine Sprache über dem Alphabet $\Gamma = \Sigma \cup \{\varnothing, \epsilon, |, ^*, (,)\}$

Reguläre Ausdrücke

Beispiel

Die regulären Ausdrücke $(\epsilon)^*$, $(\varnothing)^*$, $(0|1)^*00$ und $(0|(\epsilon 0|\varnothing(1)^*))$ beschreiben folgende Sprachen:

γ	$(\epsilon)^*$	$(\varnothing)^*$	$(0\|1)^*00$	$(0\|(\epsilon 0\|\varnothing(1)^*))$
$L(\gamma)$	$\{\varepsilon\}$	$\{\varepsilon\}$	$\{x00 \mid x \in \{0,1\}^*\}$	$\{0\}$

$\triangleleft$

Vereinbarungen

- Um Klammern zu sparen, definieren wir folgende Präzedenzordnung: Der Sternoperator * bindet stärker als der Produktoperator und dieser wiederum stärker als der Vereinigungsoperator
- Für $(0|(\epsilon 0|\varnothing(1)^*))$ können wir also kurz $0|\epsilon 0|\varnothing 1^*$ schreiben
- Da der reguläre Ausdruck $\gamma\gamma^*$ die Sprache $L(\gamma)^+$ beschreibt, verwenden wir γ^+ als Abkürzung für den Ausdruck $\gamma\gamma^*$

Charakterisierung von REG durch reguläre Ausdrücke

Satz

REG = $\{L(\gamma) \mid \gamma$ ist ein regulärer Ausdruck$\}$

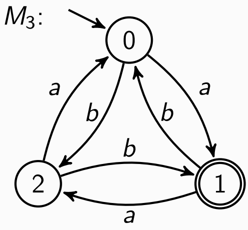
Beweis der Inklusion von rechts nach links.

Klar, da

- die Basisausdrücke $\emptyset$, ϵ und a, $a \in \Sigma^*$, reguläre Sprachen beschreiben und
- die Sprachklasse REG unter Produkt, Vereinigung und Sternhülle abgeschlossen ist. $\square$

Charakterisierung von REG durch reguläre Ausdrücke

Für die umgekehrte Inklusion betrachten wir zunächst den DFA M_3.



Frage

Wie lässt sich die Sprache

$$L(M_3) = \{x \in \{a,b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 1\}$$

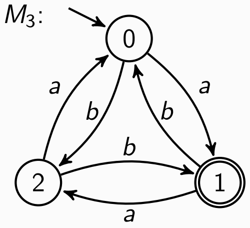
durch einen regulären Ausdruck beschreiben?

Antwort

- Sei $L_{p,q}$ die Sprache aller Wörter x, die M_3 vom Zustand p in den Zustand q überführen (d.h. $L_{p,q} = \{x \in \{a,b\}^* \mid \hat{\delta}(p,x) = q\}$)
- Weiter sei $L_{p,q}^{\neq r}$ die Sprache aller Wörter $x = x_1 \cdots x_n \in L_{p,q}$, die hierzu nur Zustände ungleich r benutzen (d.h. $\hat{\delta}(p, x_1 \cdots x_i) \neq r$ für $i = 1, \dots, n-1$)
- Dann gilt $L(M_3) = L_{0,1} = L_{0,0} L_{0,1}^{\neq 0}$, wobei $L_{0,0} = (L_{0,0}^{\neq 0})^*$ ist

Charakterisierung von REG durch reguläre Ausdrücke

Antwort (Fortsetzung)



- Dann ist $L(M_3) = L_{0,0} L_{0,1}^{\neq 0} = (L_{0,0}^{\neq 0})^* L_{0,1}^{\neq 0}$
- $L_{0,1}^{\neq 0}$ und $L_{0,0}^{\neq 0}$ lassen sich durch folgende reguläre Ausdrücke beschreiben:

$$\gamma_{0,1}^{\neq 0} = (a|bb)(ab)^*$$

$$\gamma_{0,0}^{\neq 0} = a(ab)^*(aa|b) \mid b(ba)^*(a|bb) \mid \epsilon$$

- Also ist $L(M_3)$ durch folgenden regulären Ausdruck beschreibbar:

$$\gamma_{0,1} = (a(ab)^*(aa|b) \mid b(ba)^*(a|bb))^*(a|bb)(ab)^*$$

Charakterisierung von REG durch reguläre Ausdrücke

Satz

REG = $\{L(\gamma) \mid \gamma$ ist ein regulärer Ausdruck$\}$

Beweis der Inklusion von links nach rechts.

- Wir konstruieren zu einem DFA $M = (Z, \Sigma, \delta, q_0, E)$ einen regulären Ausdruck γ mit $L(\gamma) = L(M)$.
- Wir nehmen an, dass $Z = \{1, \ldots, m\}$ und $q_0 = 1$ ist.
- Dann lässt sich $L(M)$ als Vereinigung

  $$L(M) = \bigcup_{q \in E} L_{1,q}$$

  von Sprachen der Form $L_{p,q} = \{x \in \Sigma^* \mid \hat{\delta}(p, x) = q\}$ darstellen
- Es reicht also, reguläre Ausdrücke für die Sprachen $L_{p,q}$ mit $1 \le p, q \le m$ anzugeben

Charakterisierung von REG durch reguläre Ausdrücke

Satz

$\mathrm{REG} \subseteq \{L(\gamma) \mid \gamma \text{ ist ein regulärer Ausdruck}\}$

Beweis (Fortsetzung)

- Es reicht also, reguläre Ausdrücke für die Sprachen $L_{p,q}$ mit $1 \leq p, q \leq m$ anzugeben
- Hierzu betrachten wir für $r = 0, \ldots, m$ die Sprachen

$$L_{p,q}^{\leq r} = \left\{x_1 \ldots x_n \in L_{p,q} \,\middle|\, \text{für } i = 1, \ldots, n-1 \text{ ist } \hat{\delta}(p, x_1 \ldots x_i) \leq r\right\},$$

  die wir auch einfach mit $L_{p,q}^r$ bezeichnen
- Wegen $L_{p,q} = L_{p,q}^m$ reicht es, reguläre Ausdrücke für die Sprachen $L_{p,q}^r$ mit $1 \leq p, q \leq m$ und $0 \leq r \leq m$ anzugeben
- Wir zeigen induktiv über r, dass die Sprachen $L_{p,q}^r$ durch reguläre Ausdrücke beschreibbar sind

Satz

REG $\subseteq \{L(\gamma) \mid \gamma$ ist ein regulärer Ausdruck$\}$

Beweis (Schluss)

$r = 0$: In diesem Fall sind die Sprachen

$$L^0_{p,q} = \begin{cases} \{a \in \Sigma \mid \delta(p,a) = q\}, & p \neq q, \\ \{a \in \Sigma \mid \delta(p,a) = q\} \cup \{\varepsilon\}, & \text{sonst} \end{cases}$$

endlich, also durch reg. Ausdrücke $\gamma^0_{p,q}$ beschreibbar

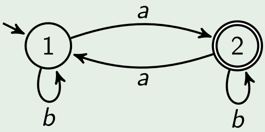
$r \rightsquigarrow r+1$: Nach IV existieren reguläre Ausdrücke $\gamma^r_{p,q}$ für die Sprachen $L^r_{p,q}$. Wegen

$$L^{r+1}_{p,q} = L^r_{p,q} \cup L^r_{p,r+1}(L^r_{r+1,r+1})^* L^r_{r+1,q}$$

sind dann $\gamma^{r+1}_{p,q} = \gamma^r_{p,q} | \gamma^r_{p,r+1}(\gamma^r_{r+1,r+1})^* \gamma^r_{r+1,q}$ reguläre Ausdrücke für die Sprachen $L^{r+1}_{p,q}$. $\square$

Beispiel

- Betrachte den DFA M



- Da M nur einen Endzustand $q = 2$ und insgesamt $m = 2$ Zustände besitzt, folgt

$$L(M) = \bigcup_{q \in E} L_{1,q} = L_{1,2} = L_{1,2}^2$$

Beispiel (Fortsetzung)

- Um reguläre Ausdrücke $\gamma_{p,q}^r$ für die Sprachen $L_{p,q}^r$ zu bestimmen, benutzen wir für $r \geq 0$ die Rekursionsformel

$$\gamma_{p,q}^{r+1} = \gamma_{p,q}^r | \gamma_{p,r+1}^r (\gamma_{r+1,r+1}^r)^* \gamma_{r+1,q}^r$$

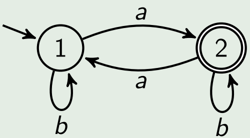
- Damit erhalten wir

$$\begin{aligned}
\gamma_{1,2}^2 &= \gamma_{1,2}^1 | \gamma_{1,2}^1 (\gamma_{2,2}^1)^* \gamma_{2,2}^1 \\
\gamma_{1,2}^1 &= \gamma_{1,2}^0 | \gamma_{1,1}^0 (\gamma_{1,1}^0)^* \gamma_{1,2}^0 \\
\gamma_{2,2}^1 &= \gamma_{2,2}^0 | \gamma_{2,1}^0 (\gamma_{1,1}^0)^* \gamma_{1,2}^0
\end{aligned}$$

- Für die Berechnung von $\gamma_{1,2}^2$ werden also nur die regulären Ausdrücke $\gamma_{1,1}^0$, $\gamma_{1,2}^0$, $\gamma_{2,1}^0$, $\gamma_{2,2}^0$, $\gamma_{2,2}^1$ und $\gamma_{1,2}^1$ benötigt

Beispiel (Fortsetzung)

DFA M



Rekursionsformeln

$$L^0_{p,p} = \{c \in \Sigma \mid \delta(p,c) = p\} \cup \{\varepsilon\}$$

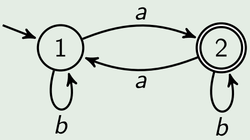
$$L^0_{p,q} = \{c \in \Sigma \mid \delta(p,c) = q\} \text{ für } p \neq q$$

$$\gamma^{r+1}_{p,q} = \gamma^r_{p,q} | \gamma^r_{p,r+1} (\gamma^r_{r+1,r+1})^* \gamma^r_{r+1,q}$$

r	p,q			
	1, 1	1, 2	2, 1	2, 2
0	$\gamma^0_{1,1}$	$\gamma^0_{1,2}$	$\gamma^0_{2,1}$	$\gamma^0_{2,2}$
1	-	$\gamma^1_{1,2}$	-	$\gamma^1_{2,2}$
2	-	$\gamma^2_{1,2}$	-	-

Beispiel (Fortsetzung)

DFA M



Rekursionsformel

$$L^0_{1,1} = \{c \in \Sigma \mid \delta(1, c) = 1\} \cup \{\varepsilon\} = \{\varepsilon, b\}$$

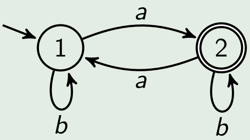
$\rightsquigarrow$ $\gamma^0_{1,1} = \epsilon|b$

r	p, q			
	$1, 1$	$1, 2$	$2, 1$	$2, 2$
0	$\epsilon\|b$	$\gamma^0_{1,2}$	$\gamma^0_{2,1}$	$\gamma^0_{2,2}$
1	-	$\gamma^1_{1,2}$	-	$\gamma^1_{2,2}$
2	-	$\gamma^2_{1,2}$	-	-

Charakterisierung von REG durch reguläre Ausdrücke

Beispiel (Fortsetzung)

DFA M



Rekursionsformel

$$L^0_{1,2} = \{c \in \Sigma \mid \delta(1,c) = 2\} = \{a\}$$

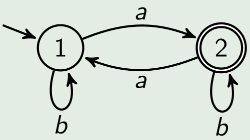
$$\rightsquigarrow \quad \gamma^0_{1,2} = a$$

r	p, q			
	1, 1	1, 2	2, 1	2, 2
0	$\epsilon \vert b$	a	$\gamma^0_{2,1}$	$\gamma^0_{2,2}$
1	-	$\gamma^1_{1,2}$	-	$\gamma^1_{2,2}$
2	-	$\gamma^2_{1,2}$	-	-

Charakterisierung von REG durch reguläre Ausdrücke

Beispiel (Fortsetzung)

DFA M



Rekursionsformel

$$L^0_{2,1} = \{c \in \Sigma \mid \delta(2,c) = 1\} = \{a\}$$

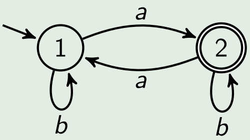
$$\leadsto \quad \gamma^0_{2,1} = a$$

r	p, q			
	1, 1	1, 2	2, 1	2, 2
0	$\epsilon \mid b$	a	a	$\gamma^0_{2,2}$
1	-	$\gamma^1_{1,2}$	-	$\gamma^1_{2,2}$
2	-	$\gamma^2_{1,2}$	-	-

Charakterisierung von REG durch reguläre Ausdrücke

Beispiel (Fortsetzung)

DFA M



Rekursionsformel

$$L^0_{2,2} = \{c \in \Sigma \mid \delta(2,c) = 2\} \cup \{\varepsilon\} = \{\varepsilon, b\}$$

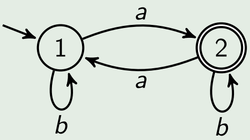
$$\rightsquigarrow \quad \gamma^0_{2,2} = \epsilon|b$$

r	p, q			
	1, 1	1, 2	2, 1	2, 2
0	$\epsilon\|b$	a	a	$\epsilon\|b$
1	-	$\gamma^1_{1,2}$	-	$\gamma^1_{2,2}$
2	-	$\gamma^2_{1,2}$	-	-

Charakterisierung von REG durch reguläre Ausdrücke

Beispiel (Fortsetzung)

DFA M



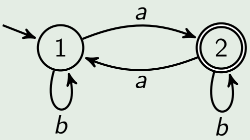
Rekursionsformel

$$\begin{aligned}\gamma_{1,2}^{1} &= \gamma_{1,2}^{0} | \gamma_{1,1}^{0} (\gamma_{1,1}^{0})^{*} \gamma_{1,2}^{0} \\ &= a | (\epsilon | b)(\epsilon | b)^{*} a \\ &\equiv b^{*} a\end{aligned}$$

r	p, q			
	1, 1	1, 2	2, 1	2, 2
0	$\epsilon \vert b$	a	a	$\epsilon \vert b$
1	-	$b^{*} a$	-	$\gamma_{2,2}^{1}$
2	-	$\gamma_{1,2}^{2}$	-	-

Beispiel (Fortsetzung)

DFA M



Rekursionsformel

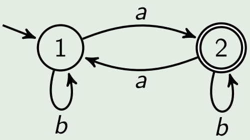
$$\gamma_{2,2}^{1} = \gamma_{2,2}^{0} | \gamma_{2,1}^{0} (\gamma_{1,1}^{0})^* \gamma_{1,2}^{0}$$
$$= (\epsilon|b)|a(\epsilon|b)^* a$$
$$\equiv \epsilon|b|ab^* a$$

r	p, q: 1, 1	1, 2	2, 1	2, 2
0	$\epsilon\|b$	a	a	$\epsilon\|b$
1	-	$b^* a$	-	$\epsilon\|b\|ab^* a$
2	-	$\gamma_{1,2}^{2}$	-	-

Charakterisierung von REG durch reguläre Ausdrücke

Beispiel (Fortsetzung)

DFA M



Rekursionsformel

$$\gamma_{1,2}^{2} = \gamma_{1,2}^{1} | \gamma_{1,2}^{1} (\gamma_{2,2}^{1})^{*} \gamma_{2,2}^{1}$$
$$= b^{*}a | b^{*}a(\epsilon|b|ab^{*}a)^{*}(\epsilon|b|ab^{*}a)$$
$$\equiv b^{*}a(b|ab^{*}a)^{*}$$

r	p,q			
	1, 1	1, 2	2, 1	2, 2
0	$\epsilon\|b$	a	a	$\epsilon\|b$
1	-	$b^{*}a$	-	$\epsilon\|b\|ab^{*}a$
2	-	$b^{*}a(b\|ab^{*}a)^{*}$	-	-

◁

Charakterisierungen der Klasse REG

Korollar

Für jede Sprache L sind folgende Aussagen äquivalent:

- L ist regulär (d.h. es gibt einen DFA M mit $L = L(M)$)
- es gibt einen NFA N mit $L = L(N)$
- es gibt einen regulären Ausdruck γ mit $L = L(\gamma)$
- L lässt sich mit den Operationen Vereinigung, Produkt und Sternhülle aus endlichen Sprachen gewinnen
- L lässt sich mit den Operationen Vereinigung, Schnitt, Komplement, Produkt und Sternhülle aus endlichen Sprachen gewinnen

Ausblick

- Als nächstes wenden wir uns der Frage zu, wie sich die Anzahl der Zustände eines DFA minimieren lässt
- Da hierbei Äquivalenzrelationen eine wichtige Rolle spielen, befassen wir uns zunächst mit Relationalstrukturen

Relationalstrukturen

Definition

- Sei A eine nichtleere Menge, R ist eine k-stellige Relation auf A, wenn $R \subseteq A^k = \underbrace{A \times \cdots \times A}_{k\text{-mal}} = \{(a_1, \ldots, a_k) \mid a_i \in A \text{ für } i = 1, \ldots, k\}$ ist
- Für $i = 1, \ldots, n$ sei R_i eine k_i-stellige Relation auf A. Dann heißt $(A; R_1, \ldots, R_n)$ Relationalstruktur
- Die Menge A heißt der Individuenbereich, die Trägermenge oder die Grundmenge der Relationalstruktur

Bemerkung

- Wir werden hier hauptsächlich den Fall $n = 1$, $k_1 = 2$, also (A, R) mit $R \subseteq A \times A$ betrachten
- Man nennt dann R eine (binäre) Relation auf A
- Oft wird für $(a, b) \in R$ auch die Infix-Schreibweise aRb benutzt

Relationalstrukturen

Beispiel

- (F, M) mit $F = \{f \mid f \text{ ist Fluss in Europa}\}$ und
$$M = \{(f, g) \in F \times F \mid f \text{ mündet in } g\}$$
- (U, B) mit $U = \{x \mid x \text{ ist Berliner}\}$ und
$$B = \{(x, y) \in U \times U \mid x \text{ ist Bruder von } y\}$$
- $(\mathcal{P}(M), \subseteq)$, wobei M eine beliebige Menge und $\subseteq$ die Inklusionsrelation auf den Teilmengen von M ist
- (A, Id_A) mit $Id_A = \{(x, x) \mid x \in A\}$ (die Identität auf A)
- $(\mathbb{R}, \leq)$
- $(\mathbb{Z}, |)$, wobei $|$ die "teilt"-Relation bezeichnet (d.h. $a|b$, falls ein $c \in \mathbb{Z}$ mit $b = ac$ existiert)

◁

Mengentheoretische Operationen auf Relationen

- Da Relationen Mengen sind, können wir den Schnitt, die Vereinigung, die Differenz und das Komplement von Relationen bilden:

$$\begin{aligned} R \cap S &= \{(x,y) \in A \times A \mid xRy \wedge xSy\} \\ R \cup S &= \{(x,y) \in A \times A \mid xRy \vee xSy\} \\ R - S &= \{(x,y) \in A \times A \mid xRy \wedge \neg xSy\} \\ \overline{R} &= (A \times A) - R \end{aligned}$$

- Sei $\mathcal{M} \subseteq \mathcal{P}(A \times A)$ eine beliebige Menge von Relationen auf A. Dann sind der Schnitt über $\mathcal{M}$ und die Vereinigung über $\mathcal{M}$ folgende Relationen:

$$\begin{aligned} \bigcap \mathcal{M} &= \bigcap_{R \in \mathcal{M}} R = \{(x,y) \mid \forall R \in \mathcal{M} : xRy\} \\ \bigcup \mathcal{M} &= \bigcup_{R \in \mathcal{M}} R = \{(x,y) \mid \exists R \in \mathcal{M} : xRy\} \end{aligned}$$

Weitere Operationen auf Relationen

Definition

- Die transponierte (konverse) Relation zu R ist

  $$R^T = \{(y, x) \mid xRy\}$$

- R^T wird oft auch mit R^{-1} bezeichnet
- Zum Beispiel ist $(\mathbb{R}, \leq^T) = (\mathbb{R}, \geq)$
- Das Produkt (oder die Komposition) zweier Relationen R und S ist

  $$R \circ S = \{(x, z) \in A \times A \mid \exists y \in A : xRy \wedge ySz\}$$

Beispiel

Ist B die Relation "ist Bruder von", V "ist Vater von", M "ist Mutter von" und $E = V \cup M$ "ist Elternteil von", so ist $B \circ E$ die Onkel-Relation $\triangleleft$

Das Relationenprodukt

Notation

- Für $R \circ S$ wird auch $R\,;S$, $R \cdot S$ oder einfach RS geschrieben.
- Für $\underbrace{R \circ \cdots \circ R}_{\text{n-mal}}$ schreiben wir auch R^n. Dabei ist $R^0 = Id$

Vorsicht!

Das Relationenprodukt R^n darf nicht mit dem kartesischen Produkt

$$\underbrace{R \times \cdots \times R}_{\text{n-mal}}$$

verwechselt werden

Vereinbarung

Wir vereinbaren, dass R^n das n-fache Relationenprodukt bezeichnen soll, falls R eine Relation ist

Eigenschaften von Relationen

Definition

Sei R eine Relation auf A. Dann heißt R

reflexiv,	falls $\forall x \in A : xRx$	(also $Id_A \subseteq R$)
irreflexiv,	falls $\forall x \in A : \neg xRx$	(also $Id_A \subseteq \overline{R}$)
symmetrisch,	falls $\forall x, y \in A : xRy \Rightarrow yRx$	(also $R \subseteq R^T$)
asymmetrisch,	falls $\forall x, y \in A : xRy \Rightarrow \neg yRx$	(also $R \subseteq \overline{R^T}$)
antisymmetrisch,	falls $\forall x, y \in A : xRy \wedge yRx \Rightarrow x = y$	(also $R \cap R^T \subseteq Id$)
konnex,	falls $\forall x, y \in A : xRy \vee yRx$	(also $A \times A \subseteq R \cup R^T$)
semikonnex,	falls $\forall x, y \in A : x \neq y \Rightarrow xRy \vee yRx$	(also $\overline{Id} \subseteq R \cup R^T$)
transitiv,	falls $\forall x, y, z \in A : xRy \wedge yRz \Rightarrow xRz$	(also $R^2 \subseteq R$)

gilt

Überblick über Relationalstrukturen

Äquivalenz- und Ordnungsrelationen

	refl.	sym.	trans.	antisym.	asym.	konnex	semikon.
Äquivalenzrelation	✓	✓	✓				
(Halb-)Ordnung	✓		✓	✓			
Striktordnung			✓		✓		
lineare Ordnung			✓	✓		✓	
lin. Striktord.			✓		✓		✓
Quasiordnung	✓		✓				

Bemerkung

In der Tabelle sind nur die definierenden Eigenschaften durch ein "✓" gekennzeichnet. Das schließt nicht aus, dass noch weitere Eigenschaften vorliegen

Eigenschaften von Relationen

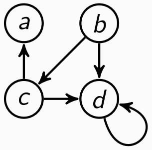
Beispiel

- Die Relation "ist Schwester von" ist zwar in einer reinen Damengesellschaft symmetrisch, i.a. jedoch weder symmetrisch noch asymmetrisch noch antisymmetrisch.
- Die Relation "ist Geschwister von" ist zwar symmetrisch, aber weder reflexiv noch transitiv und somit keine Äquivalenzrelation.
- $(\mathbb{R}, <)$ ist irreflexiv, asymmetrisch, transitiv und semikonnex und somit eine lineare Striktordnung.
- $(\mathbb{R}, \leq)$ und $(\mathcal{P}(M), \subseteq)$ sind reflexiv, antisymmetrisch und transitiv und somit Ordnungen.
- $(\mathbb{R}, \leq)$ ist auch konnex und somit eine lineare Ordnung.
- $(\mathcal{P}(M), \subseteq)$ ist zwar im Fall $\|M\| \leq 1$ konnex, aber im Fall $\|M\| \geq 2$ weder semikonnex noch konnex. $\triangleleft$

Darstellung von endlichen Relationen

Graphische Darstellung

$$A = \{a, b, c, d\}$$
$$R = \{(b, c), (b, d), (c, a), (c, d), (d, d)\}$$



- Eine Relation R auf einer (endlichen) Menge A kann durch einen gerichteten Graphen (kurz Digraphen) $G = (A, R)$ mit Knotenmenge A und Kantenmenge R veranschaulicht werden
- Hierzu stellen wir jedes Element $x \in A$ als einen Knoten dar und verbinden jedes Knotenpaar $(x, y) \in R$ durch eine gerichtete Kante (Pfeil)
- Zwei durch eine Kante verbundene Knoten heißen adjazent oder benachbart

Darstellung von endlichen Relationen

Definition

Sei R eine binäre Relation auf A

- Die Menge der Nachfolger bzw. Vorgänger von x ist

$$R[x] = \{y \in A \mid xRy\} \text{ bzw. } R^{-1}[x] = \{y \in A \mid yRx\}$$

- Der Ausgangsgrad eines Knotens x ist $\deg^+(x) = \|R[x]\|$
- Der Eingangsgrad von x ist $\deg^-(x) = \|R^{-1}[x]\|$
- Ist R symmetrisch, so können wir die Pfeilspitzen auch weglassen
- In diesem Fall heißt $\deg(x) = \deg^-(x) = \deg^+(x)$ der Grad von x und $R[x] = R^{-1}[x]$ die Nachbarschaft von x in G
- G ist schleifenfrei, falls R irreflexiv ist
- Ist R irreflexiv und symmetrisch, so nennen wir $G = (A, R)$ einen (ungerichteten) Graphen
- Eine irreflexive und symmetrische Relation R wird meist als Menge der ungeordneten Paare $E = \big\{\{a, b\} \mid aRb\big\}$ notiert

Darstellung von endlichen Relationen

Matrixdarstellung (Adjazenzmatrix)

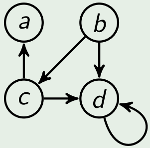
Eine Relation R auf $A = \{a_1, \ldots, a_n\}$ lässt sich auch durch die boolesche $(n \times n)$-Matrix $M_R = (m_{ij})$ darstellen mit

$$m_{ij} = \begin{cases} 1, & a_i R a_j \\ 0, & \text{sonst} \end{cases}$$

Beispiel

Die Relation $R = \{(b,c),(b,d),(c,a),(c,d),(d,d)\}$ auf $A = \{a,b,c,d\}$ hat beispielsweise die Matrixdarstellung

$$M_R = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



◁

Darstellung von endlichen Relationen

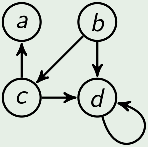
Listendarstellung (Adjazenzlisten)

R lässt sich auch durch eine Tabelle darstellen, die jedem Element $x \in A$ seine Nachfolger in Form einer Liste zuordnet

Beispiel

Die Relation $R = \{(b,c),(b,d),(c,a),(c,d),(d,d)\}$ auf $A = \{a,b,c,d\}$ lässt sich beispielsweise durch folgende Adjazenzlisten darstellen:

x:	$R[x]$
a:	-
b:	c, d
c:	a, d
d:	d



◁

Berechnung des Relationenprodukts

Berechnung von $R \circ S$

- Sind $M_R = (r_{ij})$ und $M_S = (s_{ij})$ boolesche $(n \times n)$-Matrizen für R und S, so erhalten wir für $T = R \circ S$ die Matrix $M_T = (t_{ij})$ mit

$$t_{ij} = \bigvee_{k=1,\dots,n} (r_{ik} \wedge s_{kj})$$

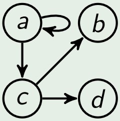
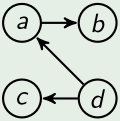
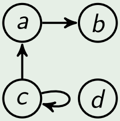
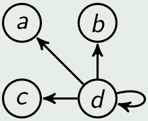
- Die Nachfolgermenge $T[x]$ von x bzgl. der Relation $T = R \circ S$ berechnet sich zu

$$T[x] = \bigcup_{y \in R[x]} S[y]$$

Das Relationenprodukt

Beispiel

Betrachte die Relationen R = $\{(a,a),(a,c),(c,b),(c,d)\}$ und $S = \{(a,b),(d,a),(d,c)\}$ auf der Menge $A = \{a,b,c,d\}$.

Relation	R	S	$R \circ S$	$S \circ R$
Digraph				
Adjazenz-matrix	$\begin{matrix}1&0&1&0\\0&0&0&0\\0&1&0&1\\0&0&0&0\end{matrix}$	$\begin{matrix}0&1&0&0\\0&0&0&0\\0&0&0&0\\1&0&1&0\end{matrix}$	$\begin{matrix}0&1&0&0\\0&0&0&0\\1&0&1&0\\0&0&0&0\end{matrix}$	$\begin{matrix}0&0&0&0\\0&0&0&0\\0&0&0&0\\1&1&1&1\end{matrix}$
Adjazenz-listen	a: a, c b: - c: b, d d: -	a: b b: - c: - d: a, c	a: b b: - c: a, c d: -	a: - b: - c: - d: a, b, c, d

◁

Hüllenoperatoren

Frage

Welche Paare muss man zu einer Relation R mindestens hinzufügen, damit R transitiv wird?

Antwort

- Es ist klar, dass der Schnitt von transitiven Relationen wieder transitiv ist
- Die transitive Hülle von R ist

$$R^+ = \bigcap \{ S \subseteq A \times A \mid S \text{ ist transitiv und } R \subseteq S \}$$

- R^+ ist also eine transitive Relation, die R enthält
- Da R^+ zudem in jeder Relation mit diesen Eigenschaften enthalten ist, gibt es keine transitive Relation mit weniger Paaren, die R enthält
- Da auch die Reflexivität und die Symmetrie bei der Schnittbildung erhalten bleiben, lassen sich nach demselben Muster weitere Hüllenoperatoren definieren

Weitere Hüllenoperatoren

Definition

Sei R eine Relation auf A

- Die reflexive Hülle von R ist

$$h_{\mathrm{refl}}(R) = \bigcap\{S \subseteq A \times A \mid S \text{ ist reflexiv und } R \subseteq S\}$$

- Die symmetrische Hülle von R ist

$$h_{\mathrm{sym}}(R) = \bigcap\{S \subseteq A \times A \mid S \text{ ist symmetrisch und } R \subseteq S\}$$

- Die reflexiv-transitive Hülle von R ist

$$R^* = \bigcap\{S \subseteq A \times A \mid S \text{ ist reflexiv, transitiv und } R \subseteq S\}$$

- Die Äquivalenzhülle von R ist

$$h_{\mathrm{äq}}(R) = \bigcap\{E \subseteq A \times A \mid E \text{ ist eine Äquivalenzrelation mit } R \subseteq E\}$$

Transitive und reflexive Hülle

Satz

$h_{\text{refl}}(R) = R \cup Id_A, \quad h_{\text{sym}}(R) = R \cup R^T, \quad R^+ = \bigcup_{n \geq 1} R^n, \quad R^* = \bigcup_{n \geq 0} R^n$

Beweis

Siehe Übungen. $\square$

Transitive und reflexive Hülle

Bemerkung. Sei $G = (V, E)$ ein Digraph.

- Ein Paar (a, b) ist genau dann in der reflexiv-transitiven Hülle E^* von E enthalten, wenn es ein $n \geq 0$ gibt mit aE^nb
- Dies bedeutet, dass es Elemente $x_0, \dots, x_n \in V$ gibt mit

$$x_0 = a,\ x_n = b \ \text{ und } \ (x_{i-1}, x_i) \in E \text{ für } i = 1, \dots, n$$

- $x_0, \dots, x_n$ heißt Weg der Länge n von a nach b in G
- G heißt zusammenhängend, wenn es in G für je zwei Knoten a und b einen Weg von a nach b oder einen Weg von b nach a gibt
- G heißt stark zusammenhängend, wenn es in G von jedem Knoten a einen Weg zu jedem Knoten b gibt

Äquivalenzrelationen

Definition

(A, R) heißt Äquivalenzrelation, wenn R eine reflexive, symmetrische und transitive Relation auf A ist

Beispiel

- Auf der Menge aller Geraden im $\mathbb{R}^2$ die Parallelität
- Auf der Menge aller Menschen "im gleichen Jahr geboren wie"
- Auf $\mathbb{Z}$ die Relation "gleicher Rest bei Division durch m". $\triangleleft$

Äquivalenzklassen

Definition

- Ist E eine Äquivalenzrelation, so nennt man die Nachbarschaft $E[x]$ die von x repräsentierte Äquivalenzklasse und bezeichnet sie auch mit $[x]_E$ (oder einfach mit $[x]$, falls E aus dem Kontext ersichtlich ist):

$$[x]_E = [x] = E[x] = \{y \mid xEy\}$$

- Eine Menge $S \subseteq A$ heißt Repräsentantensystem, falls sie genau ein Element aus jeder Äquivalenzklasse enthält
- Die Menge aller Äquivalenzklassen von E wird Quotienten- oder Faktormenge von A bzgl. E genannt und mit A/E bezeichnet:

$$A/E = \{[x]_E \mid x \in A\}$$

- Die Anzahl $\|A/E\|$ der Äquivalenzklassen von E wird auch als der Index von E (kurz: $\mathit{index}(E)$) bezeichnet

Äquivalenzrelationen

Beispiel

Für die weiter oben betrachteten Äquivalenzrelationen erhalten wir folgende Klasseneinteilungen:

- Für die Parallelität auf der Menge aller Geraden im $\mathbb{R}^2$:
  alle Geraden mit derselben Richtung (oder Steigung) bilden jeweils eine Äquivalenzklasse
- Ein Repräsentantensystem wird beispielsweise durch die Menge aller Ursprungsgeraden gebildet
- Für die Relation "im gleichen Jahr geboren wie" auf der Menge aller Menschen: jeder Jahrgang bildet eine Äquivalenzklasse
- Für die Relation "gleicher Rest bei Division durch m" auf $\mathbb{Z}$:
  jede der m Restklassen $[0], [1], \ldots, [m-1]$ mit

  $$[r] = \{a \in \mathbb{Z} \mid a \bmod m = r\}$$

  bildet eine Äquivalenzklasse
- Repräsentantensystem: $\{0, 1, \ldots, m-1\}$. $\triangleleft$

Bemerkungen

- Die kleinste Äquivalenzrelation auf A ist die Identität Id_A, die größte ist die Allrelation $A \times A$
- Die Äquivalenzklassen der Identität enthalten jeweils nur ein Element, d.h. $[x]_{Id_A} = \{x\}$ für alle $x \in A$
- Die Allrelation erzeugt dagegen nur eine Äquivalenzklasse, nämlich $[x]_{A \times A} = A$ für alle $x \in A$
- Die Identität Id_A hat nur ein Repräsentantensystem, nämlich A
- Dagegen kann jede Singletonmenge $\{x\}$ mit $x \in A$ als Repräsentantensystem für die Allrelation $A \times A$ fungieren

Partition einer Menge

Wie wir sehen werden, bilden die Äquivalenzklassen eine Zerlegung von A

Definition

Eine Familie $\{B_i \mid i \in I\}$ von nichtleeren Teilmengen $B_i \subseteq A$ heißt Partition (oder Zerlegung) der Menge A, falls gilt:

- die Mengen B_i überdecken A, d.h. $A = \bigcup_{i \in I} B_i$ und
- die Mengen B_i sind paarweise disjunkt, d.h. für je zwei verschiedene Mengen $B_i \neq B_j$ gilt $B_i \cap B_j = \varnothing$

Verfeinerung und Vergröberung von Äquivalenzrelationen

Bemerkungen

- Für zwei Äquivalenzrelationen $E \subseteq E'$ sind auch die Äquivalenzklassen $[x]_E$ von E in den Klassen $[x]_{E'}$ von E' enthalten
- Folglich ist jede Äquivalenzklasse von E' die Vereinigung von (evtl. mehreren) Äquivalenzklassen von E
- Im Fall $E \subseteq E'$ sagt man auch, E bewirkt eine feinere Zerlegung von A als E'
- Demnach ist die Identität die feinste und die Allrelation die gröbste Äquivalenzrelation

Äquivalenzrelationen und Partitionen

Satz

Sei E eine Relation auf A. Dann sind folgende Aussagen äquivalent:

1. E ist eine Äquivalenzrelation auf A
2. es gibt eine Partition $\{B_i \mid i \in I\}$ von A mit $xEy \Leftrightarrow \exists i \in I : x, y \in B_i$

Beweis.

1 impliziert 2: Sei E eine Äquivalenzrelation auf A. Dann bildet $\{E[x] \mid x \in A\}$ eine Partition von A mit der gewünschten Eigenschaft:

- Da E reflexiv ist, gilt xEx und somit $x \in E[x]$, d.h. $A = \bigcup_{x \in A} E[x]$
- Ist $E[x] \cap E[y] \neq \emptyset$ und $u \in E[x] \cap E[y]$, so folgt $E[x] = E[y]$:

$$z \in E[x] \Leftrightarrow xEz \overset{xEu}{\Leftrightarrow} uEz \overset{yEu}{\Leftrightarrow} yEz \Leftrightarrow z \in E[y]$$

- Zudem gilt

$$\exists z \in A : x, y \in E[z] \Leftrightarrow \exists z : z \in E[x] \cap E[y] \Leftrightarrow E[x] = E[y] \overset{y \in E[y]}{\Leftrightarrow} xEy$$

Äquivalenzrelationen und Partitionen

Satz

Sei E eine Relation auf A. Dann sind folgende Aussagen äquivalent:

1. E ist eine Äquivalenzrelation auf A
2. es gibt eine Partition $\{B_i \mid i \in I\}$ von A mit $xEy \Leftrightarrow \exists i \in I : x, y \in B_i$

Beweis.

2 impliziert 1: Existiert umgekehrt eine Partition $\{B_i \mid i \in I\}$ von A mit $xEy \Leftrightarrow \exists i \in I : x, y \in B_i$, so ist E

- reflexiv, da zu jedem $x \in A$ eine Menge B_i mit $x \in B_i$ existiert,
- symmetrisch, da aus $x, y \in B_i$ auch $y, x \in B_i$ folgt, und
- transitiv, da aus $x, y \in B_i$ und $y, z \in B_j$ wegen $y \in B_i \cap B_j$ die Gleichheit $B_i = B_j$ und somit $x, z \in B_i$ folgt. $\square$

Ordnungsrelationen

Definition

(A, R) heißt Ordnung (auch Halbordnung oder partielle Ordnung), wenn R eine reflexive, antisymmetrische und transitive Relation auf A ist

Beispiel

- $(\mathcal{P}(M), \subseteq)$, $(\mathbb{Z}, \leq)$, $(\mathbb{R}, \leq)$, $(\mathbb{N}, |)$, sind Ordnungen. $(\mathbb{Z}, |)$ ist keine Ordnung, aber eine Quasiordnung.
- Ist R eine Relation auf A und $B \subseteq A$, so ist $R_B = R \cap (B \times B)$ die Einschränkung von R auf B
- Einschränkungen von (linearen) Ordnungen sind ebenfalls (lineare) Ordnungen
- Beispielsweise ist $(\mathbb{Q}, \leq)$ die Einschränkung von $(\mathbb{R}, \leq)$ auf $\mathbb{Q}$ und $(\mathbb{N}, |)$ die Einschränkung von $(\mathbb{Z}, |)$ auf $\mathbb{N}$. $\triangleleft$

Darstellung einer Ordnung durch ein Hasse-Diagramm

- Sei $\leq$ eine Ordnung auf A und sei $<$ die Relation $\leq \setminus Id_A$, d.h.

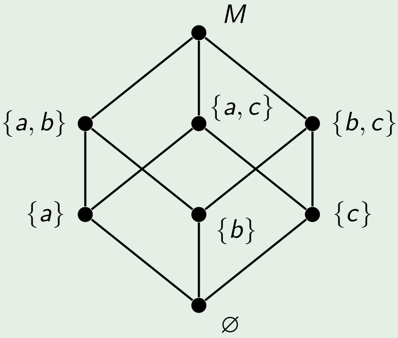
  $$x < y \Leftrightarrow x \leq y \wedge x \neq y$$

- Ein Element $x \in A$ heißt unterer Nachbar von y (kurz: $x \lessdot y$), falls $x < y$ gilt und kein $z \in A$ existiert mit $x < z < y$
- $\lessdot$ ist also die Relation $< \setminus <^2$
- Um die Ordnung $(A, \leq)$ in einem Hasse-Diagramm darzustellen, wird nur der Digraph der Relation $(A, \lessdot)$ gezeichnet
- Weiterhin wird im Fall $x \lessdot y$ der Knoten y oberhalb des Knotens x gezeichnet, so dass auf die Pfeilspitzen verzichtet werden kann

Das Hasse-Diagramm für $(\mathcal{P}(M); \subseteq)$

Beispiel

Die Inklusion $\subseteq$ auf $\mathcal{P}(M)$ mit $M = \{a, b, c\}$ lässt sich durch folgendes Hasse-Diagramm darstellen:

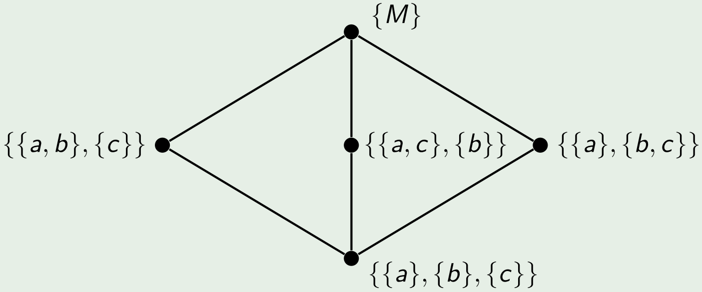


◁

Das Hasse-Diagramm der Feiner-Relation

Beispiel

Die "feiner als" Relation auf der Menge aller Partitionen von $M = \{a, b, c\}$ ist durch folgendes Hasse-Diagramm darstellbar:

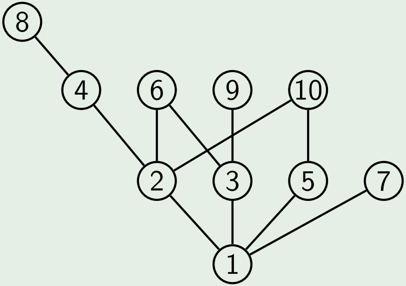


◁

Das Hasse-Diagramm der "teilt"-Relation

Beispiel

Die Einschränkung der "teilt"-Relation auf die Menge $\{1, 2, \ldots, 10\}$ ist durch folgendes Hasse-Diagramm darstellbar:



◁

Maximale, minimale, größte und kleinste Elemente

Definition

Sei $\leq$ eine Ordnung auf A und sei b ein Element in einer Teilmenge $B \subseteq A$

- b heißt kleinstes Element oder Minimum von B ($b = \min B$), falls gilt:

$$\forall b' \in B : b \leq b'$$

- b heißt größtes Element oder Maximum von B ($b = \max B$), falls gilt:

$$\forall b' \in B : b' \leq b$$

- b heißt minimal in B, falls es in B kein kleineres Element gibt:

$$\forall b' \in B : b' \leq b \Rightarrow b' = b$$

- b heißt maximal in B, falls es in B kein größeres Element gibt:

$$\forall b' \in B : b \leq b' \Rightarrow b = b'$$

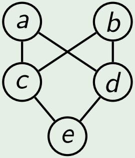
Bemerkung

Wegen der Antisymmetrie kann es in B höchstens ein kleinstes und höchstens ein größtes Element geben

Maximale, minimale, größte und kleinste Elemente

Beispiel

Betrachte folgende Ordnung



B	minimal in B	maximal in B	$\min B$	$\max B$
$\{a, b\}$	a, b	a, b	-	-
$\{c, d\}$	c, d	c, d	-	-
$\{a, b, c\}$	c	a, b	c	-
$\{a, b, c, e\}$	e	a, b	e	-
$\{a, c, d, e\}$	e	a	e	a

Obere und untere Schranken

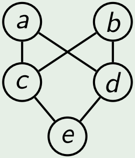
Definition

Sei $\leq$ eine Ordnung auf A und sei $B \subseteq A$. Dann heißt

- ein Element $u \in A$ mit $u \leq b$ für alle $b \in B$ untere Schranke von B
- ein Element $o \in A$ mit $b \leq o$ für alle $b \in B$ obere Schranke von B
- B nach oben beschränkt, wenn B eine obere Schranke hat
- B nach unten beschränkt, wenn B eine untere Schranke hat
- B beschränkt, wenn B nach oben und nach unten beschränkt ist

Obere und untere Schranken

Beispiel (Fortsetzung)



B	minimal	maximal	min	max	untere Schranken	obere Schranken
$\{a, b\}$	a, b	a, b	-	-	c, d, e	-
$\{c, d\}$	c, d	c, d	-	-	e	a, b
$\{a, b, c\}$	c	a, b	c	-	c, e	-
$\{a, b, c, e\}$	e	a, b	e	-	e	-
$\{a, c, d, e\}$	e	a	e	a	e	a

Infima und Suprema

Definition

Sei $\leq$ eine Ordnung auf A und sei $B \subseteq A$

- Besitzt B eine größte untere Schranke i, d.h. besitzt die Menge U aller unteren Schranken von B ein größtes Element i, so heißt i das Infimum von B ($i = \inf B$):

$$(\forall b \in B : b \geq i) \wedge [\forall u \in A : (\forall b \in B : b \geq u) \Rightarrow u \leq i]$$

- Besitzt B eine kleinste obere Schranke s, d.h. besitzt die Menge O aller oberen Schranken von B ein kleinstes Element s, so heißt s das Supremum von B ($s = \sup B$):

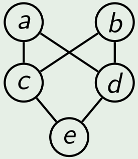
$$(\forall b \in B : b \leq s) \wedge [\forall o \in A : (\forall b \in B : b \leq o) \Rightarrow s \leq o]$$

Bemerkung

B kann nicht mehr als ein Supremum und ein Infimum haben

Infima und Suprema

Beispiel (Schluss)



B	minimal	maximal	min	max	untere Schranken	obere Schranken	inf	sup
$\{a, b\}$	a, b	a, b	-	-	c, d, e	-	-	-
$\{c, d\}$	c, d	c, d	-	-	e	a, b	e	-
$\{a, b, c\}$	c	a, b	c	-	c, e	-	c	-
$\{a, b, c, e\}$	e	a, b	e	-	e	-	e	-
$\{a, c, d, e\}$	e	a	e	a	e	a	e	a

◁

Bemerkung

- In einer endlichen linearen Ordnung $(A; \leq)$ besitzt jede nichtleere Teilmenge $B \subseteq A$ ein Maximum und ein Minimum sowie ein Supremum und ein Infimum, wobei $\sup B = \max B$ und $\inf B = \min B$
- Zudem ist $\sup \varnothing = \min A$ und $\inf \varnothing = \max A$
- Dagegen müssen in einer unendlichen linearen Ordnung nicht einmal beschränkte Teilmengen ein Supremum oder Infimum besitzen
- So hat in der linear geordneten Menge $(\mathbb{Q}, \leq)$ die Teilmenge

  $$B = \{x \in \mathbb{Q} \mid x^2 \leq 2\} = \{x \in \mathbb{Q} \mid x^2 < 2\}$$

  weder ein Supremum noch ein Infimum
- Dagegen hat in $(\mathbb{R}, \leq)$ jede beschränkte Teilmenge B ein Supremum und ein Infimum (aber möglicherweise kein Maximum oder Minimum)

Abbildungen

Definition. Sei R eine binäre Relation auf einer Menge M.

- R heißt rechtseindeutig, falls für alle $x, y, z \in M$ gilt:

  $$xRy \wedge xRz \Rightarrow y = z$$

- R heißt linkseindeutig, falls für alle $x, y, z \in M$ gilt:

  $$xRz \wedge yRz \Rightarrow x = y$$

- Der Nachbereich $N(R)$ und der Vorbereich $V(R)$ von R sind

  $$N(R) = \bigcup_{x \in M} R[x] \quad \text{und} \quad V(R) = \bigcup_{x \in M} R^T[x]$$

- R ist also genau dann rechtseindeutig, wenn jedes Element $x \in M$ höchstens einen Nachfolger hat, also $R[x]$ höchstens einelementig ist,
- und genau dann linkseindeutig, wenn jedes Element $x \in M$ höchstens einen Vorgänger hat, also $R^{-1}[x]$ höchstens einelementig ist

Abbildungen

Abbildungen ordnen jedem Element ihres Definitionsbereichs genau ein Element zu

Definition

Eine rechtseindeutige Relation R mit $V(R) = A$ und $N(R) \subseteq B$ heißt Abbildung oder Funktion von A nach B (kurz $R : A \to B$)

Bemerkung

- Wie üblich werden wir Abbildungen meist mit kleinen Buchstaben $f, g, h, \ldots$ bezeichnen und für $(x, y) \in f$ nicht xfy sondern $f(x) = y$ oder $f : x \mapsto y$ schreiben
- Ist $f : A \to B$ eine Abbildung, so wird der Vorbereich $V(f) = A$ der Definitionsbereich und die Menge B der Wertebereich oder Wertevorrat von f genannt
- Der Nachbereich $N(f)$ wird als Bild von f bezeichnet

Abbildungen

Definition

Sei $f : A \to B$ eine Abbildung

- Im Fall $N(f) = B$ heißt f surjektiv
- Ist f linkseindeutig, so heißt f injektiv
- In diesem Fall impliziert $f(x) = f(y)$ die Gleichheit $x = y$
- Eine injektive und surjektive Abbildung heißt bijektiv
- Ist f injektiv, so ist auch $f^{-1} : N(f) \to A$ eine Abbildung, die als die zu f inverse Abbildung bezeichnet wird

Bemerkung

Man beachte, dass der Definitionsbereich $V(f^{-1}) = N(f)$ von f^{-1} nur dann gleich B ist, wenn f auch surjektiv, also eine Bijektion ist

Homomorphismen

Definition

Seien (A_1, R_1) und (A_2, R_2) Relationalstrukturen

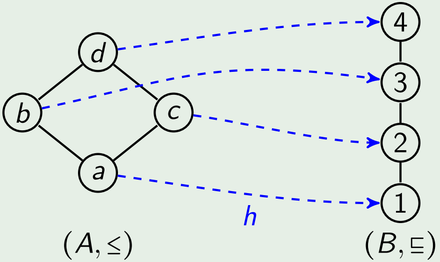
- Eine Abbildung $h : A_1 \to A_2$ heißt Homomorphismus, falls für alle $a, b \in A_1$ gilt:

$$aR_1 b \Rightarrow h(a) R_2 h(b)$$

- Sind (A_1, R_1) und (A_2, R_2) Ordnungen, so spricht man auch von Ordnungshomomorphismen oder einfach von monotonen Abbildungen
- Injektive Ordnungshomomorphismen werden auch streng monotone Abbildungen genannt

Homomorphismen

Beispiel



- Die Abbildung $h : A \to B$ ist ein bijektiver Ordnungshomomorphismus (also eine monotone Bijektion) zwischen $(A, \leq)$ und $(B, \sqsubseteq)$
- Die Umkehrabbildung h^{-1} ist jedoch nicht monoton, da zwar $2 \sqsubseteq 3$, aber $h^{-1}(2) = b \not\leq c = h^{-1}(3)$ gilt
- Dagegen ist für jede monotone Bijektion f zwischen linearen Ordnungen auch ihre Umkehrabbildung f^{-1} monoton. $\triangleleft$

Isomorphismen

Definition

- Seien (A_1, R_1) und (A_2, R_2) Relationalstrukturen
- Ein bijektiver Homomorphismus $h : A_1 \to A_2$, bei dem auch h^{-1} ein Homomorphismus ist, d.h. es gilt für alle $a, b \in A_1$,

  $$aR_1 b \Leftrightarrow h(a) R_2 h(b)$$

  heißt Isomorphismus
- In diesem Fall heißen die Strukturen (A_1, R_1) und (A_2, R_2) isomorph (kurz: $(A_1, R_1) \cong (A_2, R_2)$)

Sind (A_1, R_1) und (A_2, R_2) isomorph, so bedeutet dies, dass sich die beiden Strukturen nur in der Benennung ihrer Elemente unterscheiden

Isomorphismen

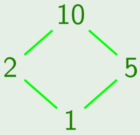
Beispiel

- Die Abbildung $h : x \mapsto e^x$ ist ein Isomorphismus zwischen den linearen Ordnungen $(\mathbb{R}, \leq)$ und $(\mathbb{R}^+, \leq)$
- Für $n \in \mathbb{N}$ sei

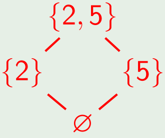
$$T_n \;=\; \{k \in \mathbb{N} \mid k \text{ teilt } n\}$$

und

$$P_n \;=\; \{p \in T_n \mid p \text{ ist prim}\}$$



$(T_{10}, |)$



$(\mathcal{P}(P_{10}), \subseteq)$

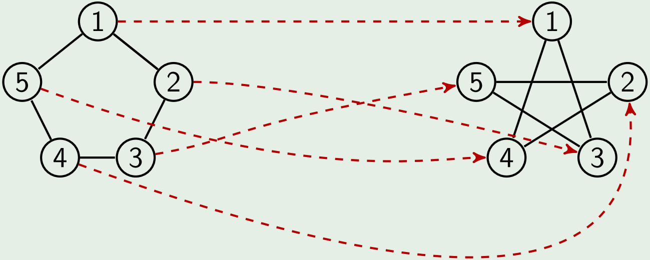
- Dann ist die Abbildung

$$h : k \mapsto P_k$$

ein (surjektiver) Ordnungshomomorphismus von $(T_n, |)$ auf $(\mathcal{P}(P_n), \subseteq)$

- h ist sogar ein Isomorphismus, falls n quadratfrei ist (d.h. es gibt keine Primzahl p, so dass p^2 die Zahl n teilt) ◁

Beispiel



$G = (V, E)$

v	1 2 3 4 5
$h_1(v)$	1 3 5 2 4
$h_2(v)$	1 4 2 5 3

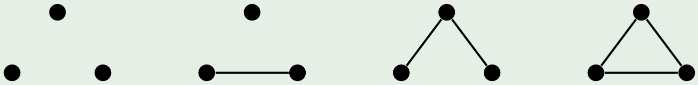
$G' = (V, E')$

- Die beiden Graphen G und G' sind isomorph
- Zwei Isomorphismen sind beispielsweise h_1 und h_2. ◁

Isomorphismen

Beispiel

- Während auf der Knotenmenge $V = \{1, 2, 3\}$ insgesamt $2^{\binom{3}{2}} = 2^3 = 8$ verschiedene Graphen existieren, gibt es auf dieser Menge nur 4 verschiedene nichtisomorphe Graphen:

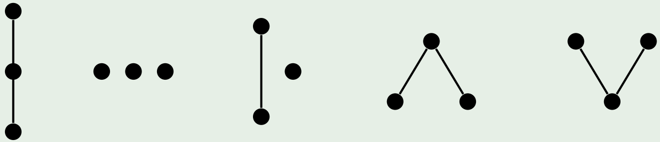


$\triangleleft$

Isomorphismen

Beispiel

- Es existieren genau 5 nichtisomorphe Ordnungen mit 3 Elementen:



- Anders ausgedrückt: Die Klasse aller dreielementigen Ordnungen zerfällt unter der Isomorphierelation $\cong$ in fünf Äquivalenzklassen, die durch obige fünf Hasse-Diagramme repräsentiert werden. $\triangleleft$

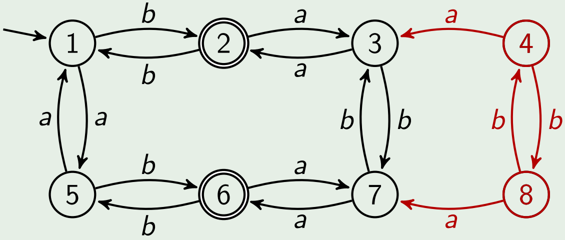
Minimierung von DFAs

Frage

Wie können wir feststellen, ob ein DFA $M = (Z, \Sigma, \delta, q_0, E)$ eine minimale Anzahl von Zuständen besitzt (und Z evtl. verkleinern)?

Beispiel

- Betrachte den DFA M



- Zunächst können alle Zustände entfernt werden, die vom Startzustand aus unerreichbar sind. ◁

Minimierung von DFAs

Frage

Wie können wir feststellen, ob ein DFA $M = (Z, \Sigma, \delta, q_0, E)$ eine minimale Anzahl von Zuständen besitzt (und Z evtl. verkleinern)?

Antwort

- Zunächst können alle unerreichbaren Zustände entfernt werden
- Zudem lassen sich zwei Zustände p und q verschmelzen, wenn M von p und q aus jeweils dieselben Wörter akzeptiert
- Für $z \in Z$ sei

$$M_z = (Z, \Sigma, \delta, z, E).$$

- Dann können wir p und q verschmelzen (in Zeichen: $p \sim_M q$), wenn $L(M_p) = L(M_q)$ ist
- Offensichtlich ist $\sim_M$ eine Äquivalenzrelation auf Z

Minimierung von DFAs

Idee

Verschmelze jeden Zustand q mit allen äquivalenten Zuständen $p \sim_M q$ zu einem neuen Zustand

Notation

- Für die durch q repräsentierte Äquivalenzklasse

  $$[q]_{\sim_M} = \{p \in Z \mid p \sim_M q\} = \{p \in Z \mid L(M_p) = L(M_q)\}$$

  schreiben wir auch einfach $[q]$ oder $\tilde{q}$
- Für eine Teilmenge $Q \subseteq Z$ bezeichne

  $$\tilde{Q} = \{\tilde{q} \mid q \in Q\}$$

  die Menge aller Äquivalenzklassen, die einen Zustand in Q enthalten
- Dann führt obige Idee auf den DFA

  $$M' = (\tilde{Z}, \Sigma, \delta', \tilde{q}_0, \tilde{E}) \quad \text{mit} \quad \delta'(\tilde{q}, a) = \widetilde{\delta(q, a)},$$

  wobei zu zeigen ist, dass $\delta'(\tilde{q}, a)$ nicht von der Wahl des Repräsentanten q für die Äquivalenzklasse $\tilde{q}$ abhängt

Wie können wir M' aus M berechnen?

- Es genügt, die Äquivalenzrelation $\sim_M$ auf der Zustandsmenge Z zu berechnen
- Hierzu genügt es, die Menge $D = \left\{ \{p, q\} \subseteq Z \;\middle|\; p \nsim_M q \right\}$ zu berechnen
- Sei $A \triangle B = (A \setminus B) \cup (B \setminus A)$ die symmetrische Differenz zweier Mengen A und B. Dann gilt

$$p \nsim_M q \;\Leftrightarrow\; L(M_p) \neq L(M_q) \;\Leftrightarrow\; L(M_p) \triangle L(M_q) \neq \emptyset$$

- Wörter $x \in L(M_p) \triangle L(M_q)$ heißen Unterscheider zwischen p und q
- Für $i \geq 0$ sei $D^{=i}$ die Menge aller Paare $\{p, q\} \in D$, die einen Unterscheider x der Länge $|x| = i$ haben, und D_i sei die Menge aller Paare $\{p, q\} \in D$, die einen Unterscheider x der Länge $|x| \leq i$ haben
- Dann gilt
  - $D_i = D^{=0} \cup D^{=1} \cup \cdots \cup D^{=i}$ und
  - $D = \bigcup_{j \geq 0} D^{=j} = \bigcup_{i \geq 0} D_i$

Algorithmische Konstruktion von M'

- Offenbar unterscheidet ε Endzustände und Nichtendzustände, d.h.

$$D_0 = D^{=0} = \left\{ \{p, q\} \subseteq Z \;\middle|\; p \in E, q \notin E \right\}$$

- Zudem gilt

$$\{p, q\} \in D^{=i+1} \Leftrightarrow \exists a \in \Sigma : \{\delta(p, a), \delta(q, a)\} \in D^{=i},$$

  da 2 Zustände p und q genau dann einen Unterscheider $x = x_1 \ldots x_{i+1}$ der Länge $i + 1$ haben, wenn die beiden Zustände $\delta(p, x_1)$ und $\delta(q, x_1)$ einen Unterscheider $x = x_2 \ldots x_{i+1}$ der Länge i haben

- Folglich ist

$$D_{i+1} = \underbrace{D_i}_{D^{=0} \cup \cdots \cup D^{=i}} \cup \underbrace{\left\{ \{p, q\} \subseteq Z \;\middle|\; \exists a \in \Sigma : \{\delta(p, a), \delta(q, a)\} \in D_i \right\}}_{D^{=1} \cup \cdots \cup D^{=i+1}}$$

- Da es nur endlich viele Zustandspaare gibt, gibt es ein $i \geq 0$ mit $D = D_i$
- Offensichtlich gilt: $D = D_i \Leftrightarrow D_{i+1} = D_i$

Algorithmus zur Minimierung eines DFA

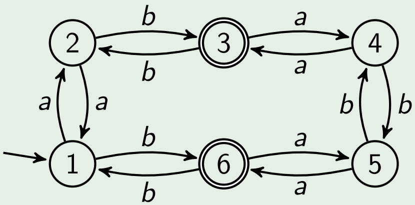
Algorithmus `min-DFA`(M)

Input: DFA $M = (Z, \Sigma, \delta, q_0, E)$
  entferne alle unerreichbaren Zustände aus Z
  $D' := \{\{p, q\} \subseteq Z \mid p \in E, q \notin E\}$
  repeat
    $D := D'$
    $D' := D \cup \{\{p, q\} \mid \exists a \in \Sigma : \{\delta(p, a), \delta(q, a)\} \in D\}$
  until $D' = D$
Output: $M' = (\tilde{Z}, \Sigma, \delta', \tilde{q}_0, \tilde{E})$, wobei $\delta'(\tilde{q}, a) = \widetilde{\delta(q, a)}$ ist
und für jeden Zustand $q \in Z$ gilt: $\tilde{q} = \{p \in Z \mid \{p, q\} \notin D\}$

Algorithmus zur Minimierung eines DFA

Beispiel

Betrachte den DFA M



2					
3	ε	ε			
4			ε		
5			ε		
6	ε	ε		ε	ε
	1	2	3	4	5

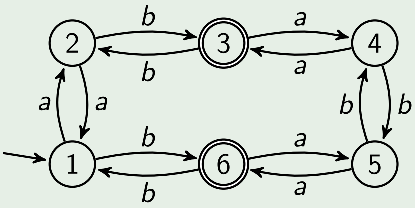
Dann enthält D_0 die Paare

$$\{1,3\}, \{1,6\}, \{2,3\}, \{2,6\}, \{3,4\}, \{3,5\}, \{4,6\}, \{5,6\}.$$

Algorithmus zur Minimierung eines DFA

Beispiel

Betrachte den DFA M



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

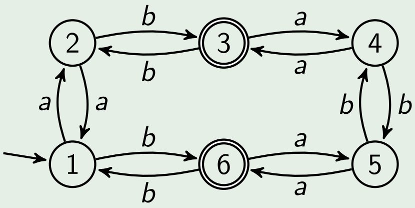
Wegen

$\{p, q\}$	$\{1, 4\}$	$\{1, 5\}$	$\{2, 4\}$	$\{2, 5\}$
$\{\delta(q, a), \delta(p, a)\}$	$\{2, 3\}$	$\{2, 6\}$	$\{1, 3\}$	$\{1, 6\}$

enthält D_1 zusätzlich die Paare $\{1, 4\}$, $\{1, 5\}$, $\{2, 4\}$, $\{2, 5\}$.

Beispiel

Betrachte den DFA M



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

Da nun jedoch keines der verbliebenen Paare $\{1,2\}$, $\{3,6\}$, $\{4,5\}$ wegen

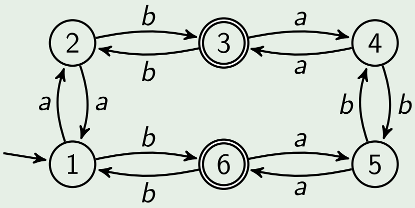
$\{p,q\}$	$\{1,2\}$	$\{3,6\}$	$\{4,5\}$
$\{\delta(p,a),\delta(q,a)\}$	$\{1,2\}$	$\{4,5\}$	$\{3,6\}$
$\{\delta(p,b),\delta(q,b)\}$	$\{3,6\}$	$\{1,2\}$	$\{4,5\}$

zu D_2 gehört, ist $D_2 = D_1$ und somit $D = D_1$.

Algorithmus zur Minimierung eines DFA

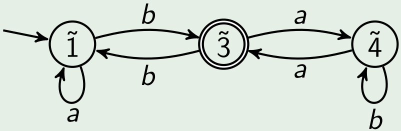
Beispiel

Betrachte den DFA M



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

Da die Paare $\{1,2\}$, $\{3,6\}$ und $\{4,5\}$ nicht in D enthalten sind, können die Zustände 1 und 2, 3 und 6, sowie 4 und 5 verschmolzen werden. Demnach hat M' die Zustände $\tilde{1} = \{1,2\}$, $\tilde{3} = \{3,6\}$ und $\tilde{4} = \{4,5\}$:



◁

Korrektheit des Minimierungsalgorithmus

Satz

Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA ohne unerreichbare Zustände. Dann ist

$$M' = (\tilde{Z}, \Sigma, \delta', \tilde{q}_0, \tilde{E}) \quad \text{mit} \quad \delta'(\tilde{q}, a) = \widetilde{\delta(q, a)}$$

ein DFA für $L(M)$ mit einer minimalen Anzahl von Zuständen

Beweis

Wir beweisen den Satz durch folgende 3 Behauptungen:

- Beh. 1: δ' ist wohldefiniert, da $\delta'(\tilde{q}, a) = \widetilde{\delta(q, a)}$ nicht von der Wahl des Repräsentanten q für die Äquivalenzklasse $\tilde{q}$ abhängt.
- Beh. 2: $L(M') = L(M)$
- Beh. 3: M' hat eine minimale Anzahl von Zuständen

Korrektheit des Minimierungsalgorithmus

Behauptung 1

δ' ist wohldefiniert, da $\delta'(\tilde{q}, a) = \widetilde{\delta(q, a)}$ nicht von der Wahl des Repräsentanten q für die Äquivalenzklasse $\tilde{q}$ abhängt

Beweis von Behauptung 1

- Hierzu ist die Implikation $p \sim_M q \Rightarrow \delta(p, a) \sim_M \delta(q, a)$ zu zeigen
- Diese ist äquivalent zur Kontraposition $\delta(p, a) \not\sim_M \delta(q, a) \Rightarrow p \not\sim_M q$, die wir bereits gezeigt haben:

$$
\begin{array}{rcl}
\delta(p, a) \not\sim_M \delta(q, a) & \Rightarrow & \text{es gibt einen Unterscheider } x \\
 & & \text{zwischen } \delta(p, a) \text{ und } \delta(q, a) \\
 & \Rightarrow & \text{es gibt einen Unterscheider } ax \\
 & & \text{zwischen } p \text{ und } q \\
 & \Rightarrow & p \not\sim_M q
\end{array}
$$

$\square$

Korrektheit des Minimierungsalgorithmus

Behauptung 2

$L(M') = L(M)$

Beweis von Behauptung 2

- Sei $x = x_1 \dots x_n \in \Sigma^*$ eine Eingabe und seien $q_0, q_1, \dots, q_n$ die von $M(x)$ besuchten Zustände, d.h. es gilt $\delta(q_{i-1}, x_i) = q_i$ für $i = 1, \dots, n$
- Nach Definition von δ' folgt daher $\delta'(\tilde{q}_{i-1}, x_i) = \tilde{q}_i$ für $i = 1, \dots, n$, d.h. M' besucht bei Eingabe x die Zustände $\tilde{q}_0, \tilde{q}_1, \dots, \tilde{q}_n$
- Da aber $\tilde{q}_n$ entweder nur End- oder nur Nicht-Endzustände enthält, gehört q_n genau dann zu E, wenn $\tilde{q}_n$ zu $\tilde{E}$ gehört, d.h. es gilt

$$x \in L(M) \Leftrightarrow x \in L(M').$$

□

Korrektheit des Minimierungsalgorithmus

Behauptung 3

M' hat eine minimale Anzahl von Zuständen

Beweis von Behauptung 3

- Wegen $\|\tilde{Z}\| \le \|Z\|$ ist M' sicher dann minimal, wenn M minimal ist
- Es reicht also zu zeigen, dass die Anzahl $\|\tilde{Z}\| = \mathit{index}(\sim_M)$ der Zustände von M' nur von der erkannten Sprache $L = L(M)$ abhängt
- Wegen $p \sim_M q \Leftrightarrow L(M_p) = L(M_q)$ gilt $\mathit{index}(\sim_M) = \|\{L(M_z) \mid z \in Z\}\|$
- Für jedes Wort $x \in \Sigma^*$ sei

  $$L_x = \{y \in \Sigma^* \mid xy \in L\} \text{ die Restsprache von } L \text{ für das Präfix } x.$$

- Dann gilt $\{L_x \mid x \in \Sigma^*\} = \{L(M_z) \mid z \in Z\}$:
  - $\subseteq$: Klar, da $L_x = L(M_z)$ für $z = \hat{\delta}(q_0, x)$ ist
  - $\supseteq$: Auch klar, da jedes $z \in Z$ über ein $x \in \Sigma^*$ erreichbar ist
- Also hängt $\mathit{index}(\sim_M) = \|\{L_x \mid x \in \Sigma^*\}\|$ nur von L ab $\square$

Ermittlung der Restsprachen von L

Beispiel

- Die Sprache

$$L = \{x_1 \dots x_n \in \{0,1\}^* \mid n \geq 2 \text{ und } x_{n-1} = 0\}$$

  hat die vier Restsprachen

$$L_x = \begin{cases} L, & x \in \{\varepsilon, 1\} \text{ oder } x \text{ endet mit } 11, \\ L \cup \{0,1\}, & x = 0 \text{ oder } x \text{ endet mit } 10, \\ L \cup \{\varepsilon, 0, 1\}, & x \text{ endet mit } 00, \\ L \cup \{\varepsilon\}, & x \text{ endet mit } 01 \end{cases}$$

- Entsprechend gibt es für L einen DFA mit 4 Zuständen, aber keinen mit 3 Zuständen. ◁

Die Nerode-Relation einer Sprache L

- Eine interessante Folgerung aus obigem Beweis ist, dass eine reguläre Sprache $L \subseteq \Sigma^*$ nur endlich viele Restsprachen hat
- Daraus folgt, dass die durch

  $$x \sim_L y \;:\Leftrightarrow\; L_x = L_y$$

  auf Σ^* definierte Äquivalenzrelation $\sim_L$ für jede reguläre Sprache $L \subseteq \Sigma^*$ einen endlichen Index hat
- Die Relation $\sim_L$ wird als Nerode-Relation von L bezeichnet

Direkte Konstruktion eines Minimal-DFA M_L aus L

- Sei $L = L(M)$ für einen DFA $M = (Z, \Sigma, \delta, q_0, E)$ und für $x \in \Sigma^*$ sei

  $$L_x = \{z \in \Sigma^* \mid xz \in L\} \text{ die Restsprache von } L \text{ für } x.$$

- Zudem sei $M' = (\tilde{Z}, \Sigma, \delta', \tilde{q}_0, \tilde{E})$ der zu M gehörige Minimal-DFA
- Dieser erreicht nach Lesen eines Wortes x den Zustand $\widetilde{\hat{\delta}(q_0, x)}$
- Wegen

  $$\begin{aligned} \widetilde{\hat{\delta}(q_0, x)} = \widetilde{\hat{\delta}(q_0, y)} &\Leftrightarrow \hat{\delta}(q_0, x) \sim_M \hat{\delta}(q_0, y) \\ &\Leftrightarrow L(M_{\hat{\delta}(q_0,x)}) = L(M_{\hat{\delta}(q_0,y)}) \Leftrightarrow L_x = L_y \end{aligned}$$

  können wir den Zustand $\widetilde{\hat{\delta}(q_0, x)}$ von M' auch mit L_x bezeichnen
- Dies führt auf den zu M' isomorphen DFA $M_L = (Z_L, \Sigma, \delta_L, L_\varepsilon, E_L)$ mit

  $$Z_L = \{L_x \mid x \in \Sigma^*\}, \; E_L = \{L_x \mid x \in L\} \text{ und } \delta_L(L_x, a) = L_{xa},$$

  der sich direkt aus der Sprache L gewinnen lässt
- M_L wird auch als Restsprachen-DFA für L bezeichnet

Direkte Konstruktion des Restsprachen-DFA M_L für L

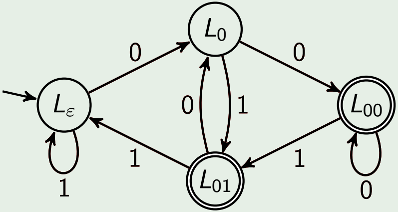
Beispiel

- Die Sprache $L = \{x_1 \dots x_n \in \{0,1\}^* \mid n \ge 2 \text{ und } x_{n-1} = 0\}$ hat die Restsprachen

$$L_x = \begin{cases} L, & x \in \{\varepsilon, 1\} \text{ oder } x \text{ endet mit } 11, \\ L \cup \{0,1\}, & x = 0 \text{ oder } x \text{ endet mit } 10, \\ L \cup \{\varepsilon, 0, 1\}, & x \text{ endet mit } 00, \\ L \cup \{\varepsilon\}, & x \text{ endet mit } 01 \end{cases}$$

- Konstruktion von M_L:

L_x	L_ε	L_0	L_{00}	L_{01}
L_{x0}	L_0	L_{00}	L_{00}	L_0
L_{x1}	L_ε	L_{01}	L_{01}	L_ε



- Für die Konstruktion von M_L spielt es keine Rolle, wie die Restsprachen L_x konkret aussehen, d.h. ihre Bestimmung ist nicht erforderlich. ◁

Der Satz von Myhill und Nerode

- Notwendig und hinreichend für die Existenz von M_L ist, dass die Menge $Z_L = \{L_x \mid x \in \Sigma^*\}$ aller Restsprachen von L endlich ist
- Dies ist genau dann der Fall, wenn die durch L auf Σ^* definierte Nerode-Relation $\sim_L$ mit

  $$x \sim_L y \;:\Leftrightarrow\; L_x = L_y$$

  einen endlichen Index hat
- Ist M bereits minimal, so haben die Zustände des zugehörigen Minimal-DFA M' die Form $\tilde{q} = \{q\}$, d.h. M' und M sind isomorph
- Da M' wiederum isomorph zu M_L ist, ist jeder minimale DFA M mit $L(M) = L$ isomorph zu M_L
- Folglich gibt es für jede reguläre Sprache L bis auf Isomorphie nur einen Minimal-DFA

Der Satz von Myhill und Nerode

Satz (Myhill und Nerode)

1. Für jede reguläre Sprache L gibt es bis auf Isomorphie nur einen Minimal-DFA. Dieser hat $index(\sim_L)$ Zustände
2. $\text{REG} = \{L \mid index(\sim_L) < \infty\}$

Bemerkung

- Sei R ein Repräsentantensystem für die Nerode-Relation $\sim_L$ von L, d.h. $\{L_x \mid x \in \Sigma^*\} = \{L_r \mid r \in R\}$ und $L_r \neq L_{r'}$ für alle $r, r' \in R$ mit $r \neq r'$
- Dann können wir die Zustände des Minimal-DFA anstelle von L_x auch mit den Repräsentanten $r \in R$ bezeichnen
- Dies führt auf den Minimal-DFA $M_R = (R, \Sigma, \delta, \varepsilon, E)$, wobei wir $\varepsilon \in R$ annehmen und $\delta(r, a) \in R$ der Repräsentant der Äquivalenzklasse $\widetilde{ra} = \{x \in \Sigma^* \mid x \sim_L ra\}$ sowie $E = R \cap L$ ist
- Wir bezeichnen M_R als den zu R gehörigen Repräsentanten-DFA für L

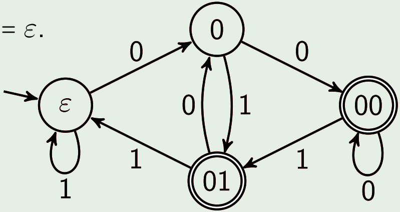
Direkte Konstruktion von M_R aus L

Beispiel

Für die Sprache $L = \{x_1 \dots x_n \in \{0,1\}^* \mid n \geq 2 \text{ und } x_{n-1} = 0\}$ lässt sich ein Repräsentanten-DFA M_R wie folgt konstruieren:

1. Sei $r_1 = \varepsilon$. Wegen $r_1 0 = 0 \not\sim_L r_1$ ist $r_2 = 0$ und $\delta(\varepsilon, 0) = 0$.
2. Wegen $r_1 1 = 1 \sim_L \varepsilon$ ist $\delta(\varepsilon, 1) = \varepsilon$.
3. Wegen $r_2 0 = 00 \not\sim_L r_i$ für $i = 1, 2$ ist $r_3 = 00$ und $\delta(0, 0) = 00$.
4. Wegen $r_2 1 = 01 \not\sim_L r_i$ für $i = 1, 2, 3$ ist $r_4 = 01$ und $\delta(0, 1) = 01$.
5. Wegen $r_3 0 = 000 \sim_L 00$ ist $\delta(00, 0) = 00$.
6. Wegen $r_3 1 = 001 \sim_L 01$ ist $\delta(00, 1) = 01$.
7. Wegen $r_4 0 = 010 \sim_L 0$ ist $\delta(01, 0) = 0$.
8. Wegen $r_4 1 = 011 \sim_L \varepsilon$ ist $\delta(01, 1) = \varepsilon$.

r	ε	0	00	01
$\delta(r,0)$	0	00	00	0
$\delta(r,1)$	ε	01	01	ε



◁

Charakterisierungen der Klasse REG

Korollar

Sei $L \subseteq \Sigma^*$ eine Sprache. Dann sind folgende Aussagen äquivalent:

- L ist regulär (d.h. es gibt einen DFA M mit $L = L(M)$),
- es gibt einen NFA N mit $L = L(N)$,
- es gibt einen regulären Ausdruck γ mit $L = L(\gamma)$,
- die Nerode-Relation $\sim_L$ von L auf Σ^* hat endlichen Index

Wir können also beweisen, dass eine Sprache L **nicht** regulär ist, indem wir

- unendlich viele verschiedene Restsprachen finden, bzw.
- unendlich viele Wörter finden, die paarweise inäquivalent bzgl. $\sim_L$ sind

Nachweis von $L \notin \mathsf{REG}$ mittels Myhill und Nerode

Satz

Die Sprache $L = \{a^n b^n \mid n \geq 0\}$ ist nicht regulär

Beweis

- Wegen

  $$b^i \in L_{a^i} \triangle L_{a^j} \quad (\text{für } 0 \leq i < j)$$

  sind die Restsprachen L_{a^i}, $i \geq 0$, paarweise verschieden.

- Wegen

  $$a^i \sim_L a^j \Leftrightarrow L_{a^i} = L_{a^j}$$

  folgt auch, dass $a^i \not\sim_L a^j$ für $i < j$ gilt und somit $\mathit{index}(\sim_L) = \infty$ ist. □

Das Pumping-Lemma

Frage

Gibt es noch andere Methoden, um zu zeigen, dass eine Sprache nicht regulär ist?

Antwort

Oft führt die Kontraposition folgender Aussage zum Ziel

Satz (Pumping-Lemma für reguläre Sprachen)

Zu jeder regulären Sprache L gibt es eine Zahl $l \geq 0$, so dass sich alle Wörter $x \in L$ mit $|x| \geq l$ in $x = uvw$ zerlegen lassen mit

1. $v \neq \varepsilon$,
2. $|uv| \leq l$ und
3. $uv^i w \in L$ für alle $i \geq 0$

Das kleinste solche l wird auch die Pumpingzahl von L genannt

Das Pumping-Lemma

Beispiel

- Die Sprache

$$L = \{x \in \{a,b\}^* \mid \#_a(x) - \#_b(x) \equiv_3 1\}$$

  lässt sich „pumpen“ (mit Pumpingzahl $l = 3$)

- Sei $x \in L$ beliebig mit $|x| \geq 3$
  - 1. Fall: x hat das Präfix $\underline{ab}$.
    Zerlege $x = uvw$ mit $u = \varepsilon$ und $v = ab$.
  - 2. Fall: x hat das Präfix $a\underline{ab}$.
    Zerlege $x = uvw$ mit $u = a$ und $v = ab$.
  - 3. Fall: x hat das Präfix $\underline{aaa}$.
    Zerlege $x = uvw$ mit $u = \varepsilon$ und $v = aaa$.
  - Restliche Fälle (Präfixe $\underline{ba}$, $b\underline{ba}$ und $\underline{bbb}$): analog

Das Pumping-Lemma

Beispiel

- Sei L eine endliche Sprache
- Offenbar lässt sich kein Wort $x \in L$ „pumpen"
- Dennoch hat L eine endliche Pumpingzahl
- Sei nämlich

$$l_{max} = \begin{cases} \max_{x \in L} |x|, & L \neq \emptyset, \\ -1, & \text{sonst.} \end{cases}$$

- Dann lässt sich jedes Wort $x \in L$ der Länge $|x| > l_{max}$ „pumpen", da solche Wörter gar nicht existieren
- Also ist die Pumpingzahl von L höchstens $l_{max} + 1$
- Zudem ist die Pumpingzahl von L größer als l_{max}, da es im Fall $l_{max} \geq 0$ ein Wort $x \in L$ der Länge $|x| = l_{max}$ gibt, das nicht „pumpbar" ist
- Also hat L die Pumpingzahl $l_{max} + 1$

Das Pumping-Lemma

Satz (Pumping-Lemma für reguläre Sprachen)

Zu jeder regulären Sprache L gibt es eine Zahl $l \geq 0$, so dass sich alle Wörter $x \in L$ mit $|x| \geq l$ in $x = uvw$ zerlegen lassen mit

1. $v \neq \varepsilon$,
2. $|uv| \leq l$ und
3. $uv^iw \in L$ für alle $i \geq 0$

Das kleinste solche l wird auch die Pumpingzahl von L genannt

Beweis

- Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA für L mit l Zuständen und sei $x = x_1 \ldots x_n \in L$ mit $n = |x| \geq l$
- Dann muss $M(x)$ nach spätestens l Schritten einen Zustand zum zweiten Mal annehmen, d.h. es ex $0 \leq j < k \leq l$ und $z \in Z$ mit

$$\hat{\delta}(q_0, x_1 \ldots x_j) = z \text{ und}$$

$$\hat{\delta}(q_0, x_1 \ldots x_j x_{j+1} \ldots x_k) = z$$

Das Pumping-Lemma

Beweis

- Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA für L mit l Zuständen und sei $x = x_1 \ldots x_n \in L$ mit $n = |x| \geq l$
- Dann muss $M(x)$ nach spätestens l Schritten einen Zustand zum zweiten Mal annehmen, d.h. es ex $0 \leq j < k \leq l$ und $z \in Z$ mit

$$\hat{\delta}(q_0, x_1 \ldots x_j) = z \text{ und}$$

$$\hat{\delta}(q_0, x_1 \ldots x_j x_{j+1} \ldots x_k) = z$$

- Setze $u = x_1 \ldots x_j$, $v = x_{j+1} \ldots x_k$ und $w = x_{k+1} \ldots x_n$.
- Dann gilt $|v| = k - j \geq 1$ (d.h. $v \neq \varepsilon$) und $|uv| = k \leq l$.
- Zudem gehört für alle $i \geq 0$ das Wort uv^iw zu L, da wegen $\hat{\delta}(z, v^i) = z$

$$\hat{\delta}(q_0, uv^iw) = \hat{\delta}(\underbrace{\hat{\delta}(\underbrace{\hat{\delta}(q_0, u)}_{z}, v^i)}_{z}, w) = \hat{\delta}(\hat{\delta}(\underbrace{\hat{\delta}(\hat{\delta}(q_0, u), v)}_{z}, w) = \hat{\delta}(q_0, x)$$

in E ist. □

Größe der Pumping-Zahl im Vergleich zum Index von $\sim_L$

Bemerkung

- Sei $min_{DFA}(L)$ ($min_{NFA}(L)$) die minimale Anzahl von Zuständen eines DFA (bzw. NFA) einer regulären Sprache L und sei $l_{reg}(L)$ die Pumping-Zahl für L
- Da wir im Beweis des Pumping-Lemmas auch einen NFA für L mit $l = min_{NFA}(L)$ Zuständen wählen können, folgt
$$l_{reg}(L) \leq min_{NFA}(L) \leq min_{DFA}(L) = index(\sim_L)$$
- Tatsächlich gibt es für jedes $i \geq 1$ eine Sprache L mit
$$l_{reg}(L) = index(\sim_L) = i$$
- Andererseits gibt es für jedes $i \geq 1$ auch eine Sprache L mit
$$l_{reg}(L) = 1 \text{ und } index(\sim_L) = i$$
- Dagegen ist $L = \varnothing$ die einzige Sprache mit der Pumping-Zahl 0
- Für diese gilt $index(\sim_\varnothing) = 1$

Kontraposition des Pumping-Lemmas

Um also $L \notin \mathrm{REG}$ zu zeigen, genügt es,

- für jede Zahl $l \geq 0$ ein Wort $x \in L$ der Länge $|x| \geq l$ zu finden, so dass
- für jede Zerlegung $x = uvw$ mindestens eine der folgenden drei Bedingungen verletzt ist:
  1. $v \neq \varepsilon$,
  2. $|uv| \leq l$ oder
  3. $uv^i w \in L$ für alle $i \geq 0$

Beispiel: $L = \{a^n b^n \mid n \geq 0\} \notin \mathrm{REG}$

- Für jede Zahl $l \geq 0$ enthält L das Wort $x = a^l b^l$ mit $|x| = 2l \geq l$
- Für jede Zerlegung $x = uvw$ von $x = a^l b^l$ mit
  1. $v \neq \varepsilon$

  ist die Bedingung

  3. $uv^i w \in L$

  für alle $i \geq 2$ verletzt

Kontraposition des Pumping-Lemmas

Um also $L \notin \mathrm{REG}$ zu zeigen, genügt es,

- für jede Zahl $l \geq 0$ ein Wort $x \in L$ der Länge $|x| \geq l$ zu finden, so dass
- für jede Zerlegung $x = uvw$ mindestens eine der folgenden drei Bedingungen verletzt ist:
  1. $v \neq \varepsilon$,
  2. $|uv| \leq l$ oder
  3. $uv^i w \in L$ für alle $i \geq 0$

Beispiel: $L = \{a^{n^2} \mid n \geq 0\} \notin \mathrm{REG}$

- Für jede Zahl $l \geq 0$ enthält L das Wort $x = a^{l^2}$ mit $|x| = l^2 \geq l$
- Für jede Zerlegung $x = uvw$ mit $|u| = r$, $|v| = s$, $|w| = t$ und
  1. $v \neq \varepsilon$ (d.h. $s \geq 1$) sowie
  2. $|uv| \leq l$ (d.h. $r + s \leq l$)

  ist die Bedingung

  3. $uv^2 w \in L$

  verletzt:

$$l^2 < \underbrace{l^2 + s}_{|uv^2w|} \leq l^2 + l < (l+1)^2$$

Kontraposition des Pumping-Lemmas

Um also $L \notin \mathrm{REG}$ zu zeigen, genügt es,

- für jede Zahl $l \geq 0$ ein Wort $x \in L$ der Länge $|x| \geq l$ zu finden, so dass
- für jede Zerlegung $x = uvw$ mindestens eine der folgenden drei Bedingungen verletzt ist:

  1. $v \neq \varepsilon$,
  2. $|uv| \leq l$ oder
  3. $uv^iw \in L$ für alle $i \geq 0$

Beispiel: $L = \{a^p \mid p \text{ prim}\} \notin \mathrm{REG}$

- Für jede Zahl $l \geq 0$ enthält L ein Wort x mit $|x| = p \geq l$
- Für jede Zerlegung $x = uvw$ mit $|v| = s$ und

  1. $v \neq \varepsilon$ (d.h. $s \geq 1$)

  ist die Bedingung

  3. $uv^iw \in L$

  wegen

  $$|uv^iw| = p + (i-1)s$$

  für $i = p + 1$ verletzt:

  $$|uv^{p+1}w| = p + ps = p(s+1)$$

Bemerkung

- Mit dem Pumping-Lemma können nicht alle Sprachen $L \notin \mathrm{REG}$ als nicht regulär nachgewiesen werden, da seine Umkehrung falsch ist
- Betrachte die Sprache
$$L = \{a^i b^j c^k \mid i = 0 \text{ oder } j = k\}$$
- L hat die Pumpingzahl $l = 1$, da jedes Wort $x \in L$ mit Ausnahme von ε „gepumpt“ werden kann
- Allerdings ist L nicht regulär (siehe Übungen)

Erzeugung regulärer Ausdrücke mit einer Grammatik

Mit Grammatiken lassen sich Sprachen oft sehr kompakt und elegant beschreiben. Implizit haben wir diese Methode bei der Definition der regulären Ausdrücke bereits benutzt

Beispiel

- Sei $\Sigma = \{a_1, \ldots, a_k\}$ ein Alphabet
- Dann lässt sich jeder reguläre Ausdruck über Σ aus dem Symbol R unter (eventuell mehrfacher) Anwendung folgender Ersetzungsregeln erzeugen:

$$
\begin{array}{lllllll}
R & \to & \emptyset & \qquad & R & \to & RR \\
R & \to & \epsilon & & R & \to & (R|R) \\
R & \to & a_i,\ i = 1, \ldots, k & & R & \to & (R)^*
\end{array}
$$

Definition einer Grammatik

Definition

Eine Grammatik ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

- V eine endliche Menge von Variablen (auch Nichtterminalsymbole genannt),
- Σ das Terminalalphabet,
- $P \subseteq (V \cup \Sigma)^* V (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ eine endliche Menge von Regeln (oder Produktionen), dabei links stets mindestens eine Variable, und
- $S \in V$ die Startvariable ist

Bemerkung

- P ist also eine binäre Relation auf $(V \cup \Sigma)^*$
- Für $(u, v) \in P$ schreiben wir auch kurz $u \to_G v$ bzw. $u \to v$, wenn die benutzte Grammatik aus dem Kontext ersichtlich ist
- Regeln der Form $\varepsilon \to v$ sind nicht erlaubt

Die von einer Grammatik erzeugte Sprache

- Ein Wort $\beta \in (V \cup \Sigma)^*$ ist aus einem Wort $\alpha \in (V \cup \Sigma)^+$ in einem Schritt ableitbar (kurz: $\alpha \Rightarrow_G \beta$), falls eine Regel $u \to_G v$ und Wörter $l, r \in (V \cup \Sigma)^*$ existieren mit

  $$\alpha = lur \text{ und } \beta = lvr$$

- Um aus einem Ableitungsschritt $\alpha = lur \Rightarrow_G lvr = \beta$ eindeutig die benutzte Regel $u \to_G v$ und das ersetzte Teilwort u von α rekonstruieren zu können, notieren wir ihn meist in der Form $l\underline{u}r \Rightarrow_G lvr$.
- Falls G aus dem Kontext ersichtlich ist, schreiben wir für $\Rightarrow_G$ auch einfach $\Rightarrow$.
- Da $\Rightarrow$ eine binäre Relation auf $(V \cup \Sigma)^*$ ist, bezeichnet
  - $\Rightarrow^m$ das m-fache Relationenprodukt von $\Rightarrow$ und
  - $\Rightarrow^*$ die reflexive, transitive Hülle von $\Rightarrow$
- Die durch G erzeugte Sprache ist $L(G) = \{x \in \Sigma^* \mid S \Rightarrow_G^* x\}$
- Ein Wort $\alpha \in (V \cup \Sigma)^*$ mit $S \Rightarrow_G^* \alpha$ heißt Satzform von G

Ableitung eines Wortes

Beispiel

- Sei $\Sigma = \{0,1\}$ und sei

$$G = (\{R\}, \Sigma', P, R)$$

die Grammatik für die Sprache RA_Σ aller regulären Ausdrücke über dem Alphabet Σ mit dem Terminalalphabet

$$\Sigma' = \Sigma \cup \{\varnothing, \epsilon, (,), |, ^*\} = \{0, 1, \varnothing, \epsilon, (,), |, ^*\}$$

und den Regeln

$$\begin{aligned} P\colon\ & R \to 0, 1, \varnothing, \epsilon \\ & R \to RR,\ (R|R),\ (R)^* \end{aligned}$$

- In G lässt sich der reguläre Ausdruck $(01)^*(\epsilon|\varnothing)$ in 8 Schritten ableiten:

$$\begin{aligned} \underline{R} \ &\Rightarrow \ \underline{R}R \ \Rightarrow \ (\underline{R})^*R \ \Rightarrow \ (RR)^*\underline{R} \ \Rightarrow \ (\underline{R}R)^*(R|R) \\ &\Rightarrow \ (0\underline{R})^*(R|R) \ \Rightarrow \ (01)^*(\underline{R}|R) \ \Rightarrow \ (01)^*(\epsilon|\underline{R}) \ \Rightarrow \ (01)^*(\epsilon|\varnothing) \end{aligned}$$

◁

Die Chomsky-Hierarchie

Man unterscheidet vier Typen von Grammatiken $G = (V, \Sigma, P, S)$

Definition

1. G heißt vom Typ 3 oder regulär, falls für alle Regeln $u \to v$ gilt:

   $u \in V$ und $v \in \Sigma V \cup \Sigma \cup \{\varepsilon\}$

   (d.h. alle Regeln haben die Form $A \to aB$, $A \to a$ oder $A \to \varepsilon$)

2. G heißt vom Typ 2 oder kontextfrei, falls für alle Regeln $u \to v$ gilt:

   $u \in V$ (d.h. alle Regeln haben die Form $A \to v$)

3. G heißt vom Typ 1 oder kontextsensitiv, falls für alle Regeln $u \to v$ gilt:

   $|v| \geq |u|$ (mit Ausnahme der ε-Sonderregel, s. unten)

4. Jede Grammatik ist automatisch vom Typ 0

Die ε-Sonderregel

In einer kontextsensitiven Grammatik ist auch die Regel $S \to \varepsilon$ zulässig, falls das Startsymbol S nicht auf der rechten Seite einer Regel vorkommt

Die Chomsky-Hierarchie

Beispiel

- Sei $G = (\{R\}, \{0, 1, \varnothing, \epsilon, (,), |, ^*\}, P, R)$ wieder die Grammatik für die Sprache aller regulären Ausdrücke über $\Sigma = \{0, 1\}$ mit den Regeln

$$\begin{aligned} P: \quad & R \to 0, 1, \varnothing, \epsilon \\ & R \to RR,\ (R|R),\ (R)^* \end{aligned}$$

- Da G eine korrekte Grammatik ist, ist G vom Typ 0
- G ist auch kontextsensitiv, da die linke Seite jeder Regel die Länge 1 und jede rechte Seite mindestens die Länge 1 hat (man beachte, dass ϵ ein Terminal ist)
- Da auf der linken Seite jeder Regel eine einzelne Variable steht, ist G sogar kontextfrei
- Offenbar ist G aber keine reguläre Grammatik, da zwar die vier Regeln $R \to 0, 1, \varnothing, \epsilon$ die geforderte Form haben, nicht jedoch die drei Regeln $R \to RR,\ (R|R),\ (R)^*$

Die Chomsky-Hierarchie

- Eine Sprache heißt vom Typ i bzw. kontextfrei oder kontextsensitiv, falls sie von einer entsprechenden Grammatik erzeugt wird
- Damit erhalten wir die neuen Sprachklassen

  $$\mathsf{CFL} = \{L(G) \mid G \text{ ist eine kontextfreie Grammatik}\}$$

  (context free languages)

  und

  $$\mathsf{CSL} = \{L(G) \mid G \text{ ist eine kontextsensitive Grammatik}\}$$

  (context sensitive languages).
- Da die Klasse der Typ 0 Sprachen mit der Klasse der rekursiv aufzählbaren Sprachen übereinstimmt, bezeichnen wir diese Sprachklasse mit

  $$\mathsf{RE} = \{L(G) \mid G \text{ ist eine Grammatik}\}$$

  (recursively enumerable languages).

Die Chomsky-Hierarchie

- Wir werden bald beweisen, dass die Sprachklassen

  $$\mathsf{REG} \subset \mathsf{CFL} \subset \mathsf{CSL} \subset \mathsf{RE}$$

  eine Hierarchie bilden (d.h. die Inklusionen sind echt), die so genannte Chomsky-Hierarchie
- Zunächst rechtfertigen wir jedoch die Bezeichnung regulär für die regulären Grammatiken und für die von ihnen erzeugten Sprachen

Satz

REG $= \{L(G) \mid G$ ist eine reguläre Grammatik$\}$

Beweis von REG $\subseteq \{L(G) \mid G$ ist eine reguläre Grammatik$\}$

- Sei $M = (Z, \Sigma, \delta, q_0, E)$ ein DFA
- Wir konstruieren eine reguläre Grammatik G mit $L(G) = L(M)$
- Betrachte die Grammatik $G = (V, \Sigma, P, S)$ mit $V = Z$, $S = q_0$ und

$$P \quad = \quad \{q \to ap \mid \delta(q,a) = p\} \cup \{q \to \varepsilon \mid q \in E\}$$

Beweis von $\mathsf{REG} \subseteq \{L(G) \mid G \text{ ist eine reguläre Grammatik}\}$

- Betrachte die Grammatik $G = (V, \Sigma, P, S)$ mit $V = Z$, $S = q_0$ und

$$P = \{q \to ap \mid \delta(q,a) = p\} \cup \{q \to \varepsilon \mid q \in E\}$$

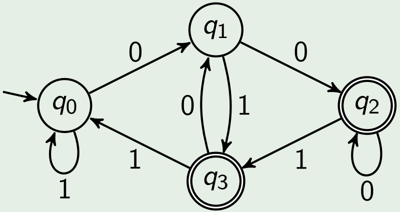
- Dann gilt für alle Wörter $x = x_1 \dots x_n \in \Sigma^*$:

$$\begin{aligned}
x \in L(M) &\Leftrightarrow \exists\, q_1, \dots, q_{n-1} \in Z \;\exists\, q_n \in E : \\
&\ \delta(q_{i-1}, x_i) = q_i \text{ für } i = 1, \dots, n \\
&\Leftrightarrow \exists\, q_1, \dots, q_n \in V : \\
&\ q_{i-1} \to x_i q_i \text{ für } i = 1, \dots, n \text{ und } q_n \to \varepsilon \\
&\Leftrightarrow \exists\, q_1, \dots, q_n \in V : \\
&\ q_0 \Rightarrow^i x_1 \dots x_i q_i \text{ für } i = 1, \dots, n \text{ und } q_n \to \varepsilon \\
&\Leftrightarrow x \in L(G)
\end{aligned}$$

□

Beispiel

- Für den DFA



erhalten wir die Grammatik $G = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, P, q_0)$ mit

$$P: \quad q_0 \to 0q_1,\ 1q_0 \qquad q_1 \to 0q_2,\ 1q_3$$
$$q_2 \to 0q_2,\ 1q_3,\ \varepsilon \qquad q_3 \to 0q_1,\ 1q_0,\ \varepsilon$$

- Der akzeptierenden Rechnung

$$q_0, q_1, q_2, q_3$$

bei Eingabe $x = 001$ entspricht dann folgende Ableitung:

$$\underline{q_0} \Rightarrow 0\underline{q_1} \Rightarrow 00\underline{q_2} \Rightarrow 001\underline{q_3} \Rightarrow 001$$

◁

- Offensichtlich lässt sich obige Konstruktion einer Grammatik G aus einem DFA M umdrehen, falls G keine Regeln der Form $A \to a$ enthält
- Zu jeder solchen regulären Grammatik G erhalten wir dann einen äquivalenten NFA
- Für den Beweis der Rückrichtung genügt es daher, alle Regeln der Form $A \to a$ zu eliminieren

Lemma

Zu jeder regulären Grammatik $G = (V, \Sigma, P, S)$ gibt es eine äquivalente reguläre Grammatik G', die keine Regeln der Form $A \to a$ hat

Beweis

Betrachte die Grammatik $G' = (V', \Sigma, P', S)$ mit

$$V' = V \cup \{X_{neu}\} \text{ und}$$

$$P' = \{A \to aX_{neu} \mid A \to_G a\} \cup \{X_{neu} \to \varepsilon\} \cup P \setminus (V \times \Sigma)$$

□

Beispiel

- Betrachte die Grammatik $G = (\{A, B, C\}, \{a, b\}, P, A)$ mit

$$
\begin{aligned}
P: \quad & A \to aB,\ bC,\ \varepsilon \\
& B \to aC,\ bA,\ b \\
& C \to aA,\ bB,\ a
\end{aligned}
$$

- Wir ersetzen die Regeln $B \to b$ und $C \to a$ durch die Regeln $B \to bD$ und $C \to aD$ und fügen die Regel $D \to \varepsilon$ hinzu
- Damit erhalten wir die Grammatik $G' = (\{A, B, C, D\}, \{a, b\}, P', A)$ mit

$$
\begin{aligned}
P': \quad & A \to aB,\ bC,\ \varepsilon \\
& B \to aC,\ bA,\ bD \\
& C \to aA,\ bB,\ aD \\
& D \to \varepsilon
\end{aligned}
$$

◁

Beweis von $\{L(G) \mid G \text{ ist eine reguläre Grammatik}\} \subseteq \mathrm{REG}$

- Sei $G = (V, \Sigma, P, S)$ eine reguläre Grammatik, die keine Regeln der Form $A \to a$ enthält
- Drehen wir obige Konstruktion einer Grammatik aus einem DFA um, so erhalten wir den NFA

  $$M = (Z, \Sigma, \Delta, \{S\}, E)$$

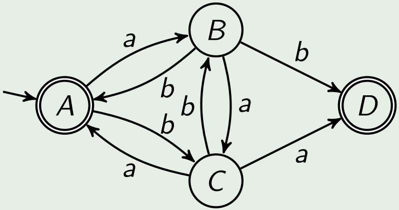
  mit $Z = V$, $\Delta(A, a) = \{B \mid A \to_G aB\}$ und $E = \{A \mid A \to_G \varepsilon\}$
- Genau wie oben folgt dann $L(M) = L(G)$ $\square$

Beispiel (Fortsetzung)

Die Grammatik $G' = (\{A, B, C, D\}, \{a, b\}, P', A)$ mit

$$\begin{aligned} P': \quad & A \to aB,\ bC,\ \varepsilon \\ & B \to aC,\ bA,\ bD \\ & C \to aA,\ bB,\ aD \\ & D \to \varepsilon \end{aligned}$$

führt auf den NFA



◁

Charakterisierungen der Klasse REG

Korollar

Sei L eine Sprache. Dann sind folgende Aussagen äquivalent:

- L ist regulär (d.h. es gibt einen DFA M mit $L = L(M)$)
- es gibt einen NFA N mit $L = L(N)$
- es gibt einen regulären Ausdruck γ mit $L = L(\gamma)$
- die Nerode-Relation $\sim_L$ von L hat einen endlichen Index
- es gibt eine reguläre Grammatik G mit $L = L(G)$

Bemerkung

- Es ist klar, dass jede reguläre Grammatik auch kontextfrei ist
- Zudem ist die Sprache $L = \{a^n b^n \mid n \geq 0\}$ nicht regulär
- Es ist aber leicht, eine kontextfreie Grammatik für L anzugeben:

$$G = (\{S\}, \{a, b\}, P, S) \text{ mit } P = \{S \to aSb, \varepsilon\}$$

- Also gilt REG $\subsetneq$ CFL
- Allerdings sind nicht alle kontextfreien Grammatiken kontextsensitiv
- Z.B. ist obige Grammatik G nicht kontextsensitiv, da sie die Regel $S \to \varepsilon$ enthält und S auf der rechten Seite der Regel $S \to aSb$ vorkommt
- Wir können G jedoch wie folgt in eine Grammatik G' umwandeln:
  - ersetze die Regel $S \to \varepsilon$ durch die Regel $S \to ab$ und
  - füge ein neues Startsymbol S' sowie die Regeln $S' \to S, \varepsilon$ hinzu
- Tatsächlich lässt sich jede kontextfreie Grammatik G in eine äquivalente kontextfreie Grammatik G' umwandeln, die auch kontextsensitiv ist

Chomsky-Normalform

Definition

Eine Grammatik $G = (V, \Sigma, P, S)$ ist in Chomsky-Normalform (CNF), falls $P \subseteq V \times (V^2 \cup \Sigma)$ ist, d.h. alle Regeln haben die Form $A \to BC$ oder $A \to a$

Satz

Zu jeder kontextfreien Grammatik G lässt sich eine CNF-Grammatik G' mit $L(G') = L(G) \setminus \{\varepsilon\}$ konstruieren

Anwendungen der Chomsky-Normalform

Korollar

CFL $\subseteq$ CSL

Beweis

- Sei $L \in$ CFL und sei $G = (V, \Sigma, P, S)$ eine CNF-Grammatik mit $L(G) = L \setminus \{\varepsilon\}$
- Im Fall $\varepsilon \notin L$ folgt sofort $L = L(G) \in$ CSL, da G kontextsensitiv ist
- Ist $\varepsilon \in L$, so erzeugt folgende kontextsensitive (und kontextfreie) Grammatik G' die Sprache $L = L(G) \cup \{\varepsilon\}$:

$$G' = (V \cup \{S_{neu}\}, \Sigma, P \cup \{S_{neu} \to S, \varepsilon\}, S_{neu})$$

□

Weitere Anwendungen der Chomsky-Normalform

- Der Beweis des Pumping-Lemmas für kontextfreie Sprachen basiert auf CNF-Grammatiken
- Zudem ermöglichen sie einen effizienten Algorithmus zur Lösung des Wortproblems für kontextfreie Sprachen

Das Pumping-Lemma für kontextfreie Sprachen

Zu jeder kontextfreien Sprache $L \in \mathrm{CFL}$ gibt es eine Zahl l, so dass sich alle Wörter $z \in L$ mit $|z| \geq l$ in $z = uvwxy$ zerlegen lassen mit

1. $vx \neq \varepsilon$,
2. $|vwx| \leq l$ und
3. $uv^iwx^iy \in L$ für alle $i \geq 0$

Das Wortproblem für kontextfreie Grammatiken

Gegeben: Eine kontextfreie Grammatik G und ein Wort x

Gefragt: Ist $x \in L(G)$?

Das Pumping-Lemma für kontextfreie Sprachen

Beispiel

- Betrachte die Sprache $L = \{a^n b^n | n \geq 0\}$
- Dann lässt sich jedes Wort $z = a^n b^n = a^{n-1} ab b^{n-1}$ in L mit $|z| \geq l = 2$ pumpen
- Zerlegen wir nämlich z in

$$z = uvwxy \text{ mit } u = a^{n-1},\ v = a,\ w = \varepsilon,\ x = b \text{ und } y = b^{n-1},$$

dann gilt

1. $vx = ab \neq \varepsilon$
2. $|vwx| = |ab| \leq 2$ und
3. $uv^i wx^i y = a^{n-1} a^i b^i b^{n-1} \in L$ für alle $i \geq 0$ ◁

Anwendung des Pumping-Lemmas

Beispiel

- Die Sprache $L = \{a^n b^n c^n \mid n \geq 0\}$ ist nicht kontextfrei
- Für eine vorgegebene Zahl $l \geq 0$ hat nämlich das Wort $z = a^l b^l c^l \in L$ die Länge $|z| = 3l \geq l$
- Dieses Wort lässt sich aber nicht pumpen:

  Für jede Zerlegung $z = uvwxy$ mit $vx \neq \varepsilon$ und $|vwx| \leq l$ gehört $z' = uv^0wx^0y$ nicht zu L:

  - Wegen $vx \neq \varepsilon$ ist $|z'| < |z|$
  - Wegen $|vwx| \leq l$ kommen in vx nicht alle drei Zeichen a, b, c vor
  - Kommt aber in vx beispielsweise kein a vor, so ist $\#_a(z) = \#_a(z')$ und somit gilt

    $$|z'| < |z| = 3\,\#_a(z) = 3\,\#_a(z')$$

  - Also gehört z' nicht zu L $\triangleleft$

Abschlusseigenschaften von CFL

Satz

CFL ist abgeschlossen unter Vereinigung, Produkt und Sternhülle

Beweis

- Seien $G_1 = (V_1, \Sigma, P_1, S_1)$ und $G_2 = (V_2, \Sigma, P_2, S_2)$ kontextfreie Grammatiken mit $V_1 \cap V_2 = \varnothing$ und sei S eine neue Variable
- Dann gilt
  - $L(G_1) \cup L(G_2) = L(G_3)$ für die kontextfreie Grammatik
    $$G_3 = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1, S_2\}, S)$$
  - $L(G_1)L(G_2) = L(G_4)$ für die kontextfreie Grammatik
    $$G_4 = (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S) \text{ und}$$
  - $L(G_1)^* = L(G_5)$ für die kontextfreie Grammatik
    $$G_5 = (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S \to S_1 S, \varepsilon\}, S)$$

□

Abschlusseigenschaften von CFL

Satz

CFL ist nicht abgeschlossen unter Schnitt und Komplement

Beweis von $L_1, L_2 \in \text{CFL} \not\Rightarrow L_1 \cap L_2 \in \text{CFL}$

- Folgende Sprachen sind kontextfrei (siehe Übungen):

$$L_1 = \{a^n b^m c^m \mid n, m \geq 0\} \quad \text{und} \quad L_2 = \{a^n b^n c^m \mid n, m \geq 0\}$$

- Nicht jedoch ihr Schnitt $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ □

Beweis von $L \in \text{CFL} \not\Rightarrow \overline{L} \in \text{CFL}$

- Wäre CFL unter Komplement abgeschlossen, so wäre CFL wegen de Morgan auch unter Schnitt abgeschlossen
- Mit $A, B \in \text{CFL}$ wären dann nämlich auch $\overline{A}, \overline{B} \in \text{CFL}$, woraus wegen

$$\overline{A}, \overline{B} \in \text{CFL} \Rightarrow \overline{A} \cup \overline{B} = \overline{A \cap B} \in \text{CFL}$$

wiederum $A \cap B \in \text{CFL}$ folgen würde □

Umwandlung in Chomsky-Normalform

Satz

Zu jeder kontextfreien Grammatik G lässt sich eine CNF-Grammatik G' mit $L(G') = L(G) \setminus \{\varepsilon\}$ konstruieren

Beweis

Wir wandeln $G = (V, \Sigma, P, S)$ wie folgt in eine CNF-Grammatik G' um:

- Wir beseitigen zunächst alle Regeln der Form $A \to \varepsilon$ und danach alle Regeln der Form $A \to B$ (siehe folgende Folien)
- Dann fügen wir für jedes Terminal $a \in \Sigma$ eine neue Variable X_a und eine neue Regel $X_a \to a$ hinzu und ersetzen jedes Vorkommen von a, bei dem a nicht alleine auf der rechten Seite einer Regel steht, durch X_a
- Anschließend führen wir für jede Regel $A \to B_1 \dots B_k$, $k \geq 3$, neue Variablen $A_1, \dots, A_{k-2}$ ein und ersetzen sie durch die $k-1$ Regeln

$$A \to B_1 A_1,\ A_1 \to B_2 A_2,\ \dots\ A_{k-3} \to B_{k-2} A_{k-2},\ A_{k-2} \to B_{k-1} B_k$$

□

Falls G Regeln mit vielen Variablen auf der rechten Seite hat, empfiehlt es sich, Regeln der Form $A \to \varepsilon$ und $A \to B$ zuletzt zu beseitigen (s. Übungen)

Beseitigung von ε-Regeln

Satz

Zu jeder kontextfreien Grammatik $G = (V, \Sigma, P, S)$ gibt es eine kontextfreie Grammatik $G' = (V, \Sigma, P', S)$ ohne ε-Regeln mit $L(G') = L(G) \setminus \{\varepsilon\}$

Beweis

- Zuerst berechnen wir die Menge $E = \{A \in V \mid A \Rightarrow^* \varepsilon\}$ aller Variablen, die nach ε ableitbar sind:

$E' := \{A \in V \mid A \to \varepsilon\}$
repeat
  $E := E'$
  $E' := E \cup \{A \in V \mid \exists\, B_1, \ldots, B_k \in E : A \to B_1 \ldots B_k\}$
until $E = E'$

- Nun bilden wir P' wie folgt:

$$\left\{ A \to v' \;\middle|\; \begin{array}{l} \text{es ex. eine Regel } A \to_G v \text{, so dass } v' \neq \varepsilon \text{ aus } v \text{ durch} \\ \text{Entfernen von beliebig vielen Variablen } A \in E \text{ entsteht} \end{array} \right\}$$

Beseitigung von ε-Regeln

Beispiel

Betrachte die Grammatik $G = (\{S, T, U, X, Y, Z\}, \{a, b, c\}, P, S)$ mit

$$
\begin{array}{llll}
P: & S \to aY, bX, Z & Y \to bS, aYY & T \to U \\
 & X \to aS, bXX & Z \to \varepsilon, S, T, cZ & U \to abc
\end{array}
$$

- Berechnung von E:

E'	$\{Z\}$	$\{Z, S\}$
E	$\{Z, S\}$	$\{Z, S\}$

- Entferne $Z \to \varepsilon$ und füge die Regeln $Y \to b$ (wegen $Y \to bS$), $X \to a$ (wegen $X \to aS$) und $Z \to c$ (wegen $Z \to cZ$) hinzu:

$$
\begin{array}{llll}
P': & S \to aY, bX, Z & Y \to b, bS, aYY & T \to U \\
 & X \to a, aS, bXX & Z \to c, S, T, cZ & U \to abc
\end{array}
$$

Beseitigung von Variablenumbenennungen

Satz

Zu jeder kontextfreien Grammatik $G = (V, \Sigma, P, S)$ gibt es eine kontextfreie Grammatik $G' = (V, \Sigma, P', S)$ ohne Regeln der Form $A \to B$ mit $L(G') = L(G)$

Beweis

- Zuerst entfernen wir sukzessive alle Zyklen $A_1 \to A_2 \to \cdots \to A_k \to A_1$
- Hierzu entfernen wir diese Regeln aus P und ersetzen alle Vorkommen der Variablen $A_2, \ldots, A_k$ in den übrigen Regeln durch A_1
- Befindet sich die Startvariable unter $A_1, \ldots, A_k$, so sei dies o.B.d.A. A_1
- Nun eliminieren wir sukzessive die restlichen Variablenumbenennungen, indem wir
    - eine Regel $A \to B$ wählen, so dass in P keine Variablenumbenennung $B \to C$ mit B auf der linken Seite existiert,
    - diese Regel $A \to B$ aus P entfernen und
    - für jede Regel $B \to v$ in P die Regel $A \to v$ zu P hinzunehmen $\square$

Beseitigung von Variablenumbenennungen

Beispiel (Fortsetzung)

$$P\colon\quad S \to aY, bX, Z \qquad Y \to b, bS, aYY \qquad T \to U$$
$$X \to a, aS, bXX \qquad Z \to c, S, T, cZ \qquad U \to abc$$

- Entferne den Zyklus $S \to Z \to S$ und ersetze Z durch S:

$$S \to aY, bX, c, T, cS \qquad Y \to b, bS, aYY \qquad T \to U$$
$$X \to a, aS, bXX \qquad U \to abc$$

- Ersetze die Regel $T \to U$ durch $T \to abc$ (wegen $U \to abc$):

$$S \to aY, bX, c, T, cS \qquad Y \to b, bS, aYY \qquad T \to abc$$
$$X \to a, aS, bXX \qquad U \to abc$$

- Ersetze dann auch die Regel $S \to T$ durch $S \to abc$ (wegen $T \to abc$):

$$S \to abc, aY, bX, c, cS \qquad Y \to b, bS, aYY \qquad T \to abc$$
$$X \to a, aS, bXX \qquad U \to abc$$

- Da T und U nirgends mehr auf der rechten Seite vorkommen, können wir die Regeln $T \to abc$ und $U \to abc$ weglassen:

$$S \to abc, aY, bX, c, cS \qquad Y \to b, bS, aYY \qquad X \to a, aS, bXX$$

Bringe alle Regeln in die Form $A \to a$ und $A \to BC$

Beispiel (Schluss)

Betrachte die Grammatik $G = (\{S, X, Y, Z\}, \{a, b, c\}, P, S)$ mit

$$P: S \to abc, aY, bX, c, cS \qquad Y \to b, bS, aYY \qquad X \to a, aS, bXX$$

- Ersetze a, b und c durch A, B und C (außer wenn sie alleine auf der rechten Seite einer Regel stehen) und füge die Regeln $A \to a$, $B \to b$, $C \to c$ hinzu:

$$S \to ABC, AY, BX, c, CS \qquad Y \to b, BS, AYY \qquad X \to a, AS, BXX$$
$$A \to a \qquad B \to b \qquad C \to c$$

- Ersetze die Regeln $S \to ABC$, $Y \to AYY$ und $X \to BXX$ durch die Regeln $S \to AS'$, $S' \to BC$, $Y \to AY'$, $Y' \to YY$ und $X \to BX'$, $X' \to XX$:

$$S \to AS', AY, BX, c, CS \qquad S' \to BC \qquad Y \to b, BS, AY' \qquad Y' \to YY$$
$$X \to a, AS, BX' \qquad X' \to XX \qquad A \to a \qquad B \to b \qquad C \to c$$

◁

Links- und Rechtsableitungen

Definition

Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik

- Eine Ableitung

$$\underline{S} \Rightarrow l_1 \underline{A_1} r_1 \Rightarrow \cdots \Rightarrow l_{m-1} \underline{A_{m-1}} r_{m-1} \Rightarrow \alpha_m$$

  heißt Linksableitung von α_m (kurz $S \Rightarrow_L^* \alpha_m$), falls in jedem Ableitungsschritt die am weitesten links stehende Variable ersetzt wird, d.h. es gilt $l_i \in \Sigma^*$ für $i = 1, \ldots, m-1$

- Rechtsableitungen $S_0 \Rightarrow_R^* \alpha_m$ sind analog definiert
- G heißt mehrdeutig, wenn es ein Wort $x \in L(G)$ gibt, das zwei verschiedene Linksableitungen hat
- Andernfalls heißt G eindeutig

Für alle $x \in \Sigma^*$ gilt: $x \in L(G) \Leftrightarrow S \Rightarrow^* x \Leftrightarrow S \Rightarrow_L^* x \Leftrightarrow S \Rightarrow_R^* x$

Ein- und mehrdeutige Grammatiken

Beispiel

- In $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$ gibt es 8 Ableitungen für $aabb$:

$$\boxed{\underline{S} \Rightarrow_L a\underline{S}bS \Rightarrow_L aa\underline{S}bSbS \Rightarrow_L aab\underline{S}bS \Rightarrow_L aabb\underline{S} \Rightarrow_L aabb}$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}bSbS \Rightarrow aabSb\underline{S} \Rightarrow aab\underline{S}b \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}bSbS \Rightarrow aa\underline{S}bbS \Rightarrow aabb\underline{S} \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}bSbS \Rightarrow aaSbb\underline{S} \Rightarrow aa\underline{S}bb \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSbSb\underline{S} \Rightarrow aa\underline{S}bSb \Rightarrow aab\underline{S}b \Rightarrow aabb$$
$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSbSb\underline{S} \Rightarrow aaSb\underline{S}b \Rightarrow aa\underline{S}bb \Rightarrow aabb$$
$$\underline{S} \Rightarrow aSb\underline{S} \Rightarrow a\underline{S}b \Rightarrow aa\underline{S}bSb \Rightarrow aab\underline{S}b \Rightarrow aabb$$
$$\boxed{\underline{S} \Rightarrow_R aSb\underline{S} \Rightarrow_R a\underline{S}b \Rightarrow_R aaSb\underline{S}b \Rightarrow_R aa\underline{S}bb \Rightarrow_R aabb}$$

- Darunter sind genau eine Links- und genau eine Rechtsableitung
- In $G' = (\{S\}, \{a, b\}, \{S \to aSbS, ab, \varepsilon\}, S)$ gibt es 3 Ableitungen für ab:

$$\boxed{\boxed{\underline{S} \Rightarrow ab}} \qquad \boxed{\underline{S} \Rightarrow a\underline{S}bS \Rightarrow ab\underline{S} \Rightarrow ab} \qquad \boxed{\underline{S} \Rightarrow aSb\underline{S} \Rightarrow a\underline{S}b \Rightarrow ab}$$

- Darunter sind zwei Links- und zwei Rechtsableitungen ◁

Ein- und mehrdeutige Grammatiken

Beispiel

- Die Grammatik $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$ ist eindeutig
- Dies liegt daran, dass keine Satzform von G das Teilwort Sa enthält
- Daher muss auf die aktuelle Satzform $y\underline{S}\beta$ einer Linksableitung

  $$S \quad \Rightarrow_L^* \quad y\underline{S}\beta \quad \Rightarrow_L^* \quad yz = x$$

  genau dann die Regel $S \to aSbS$ angewandt werden, wenn in x auf das Präfix y ein a folgt
- Dagegen ist die Grammatik $G' = (\{S\}, \{a, b\}, \{S \to aSbS, ab, \varepsilon\}, S)$ mehrdeutig, da das Wort $x = ab$ zwei Linksableitungen hat:

  $$\underline{S} \Rightarrow ab \text{ und } \underline{S} \Rightarrow a\underline{S}bS \Rightarrow ab\underline{S} \Rightarrow ab$$

$\triangleleft$

Gerichtete Bäume und Wälder

Sei $G = (V, E)$ ein Digraph.

- Ein (gerichteter) v_0-v_k-Weg in G ist eine Folge von Knoten $v_0, \ldots, v_k$ mit $(v_i, v_{i+1}) \in E$ für $i = 0, \ldots, k-1$. Seine Länge ist k
- Ein Weg heißt Pfad, falls alle Knoten paarweise verschieden sind
- Ein u-v-Weg der Länge ≥ 1 mit $u = v$ heißt Zyklus
- G heißt azyklisch, wenn es in G keinen Zyklus gibt
- G heißt gerichteter Wald, wenn G azyklisch ist und jeder Knoten $v \in V$ Eingangsgrad $\deg^-(v) \leq 1$ hat
- Ein Knoten $u \in V$ vom Ausgangsgrad $\deg^+(u) = 0$ heißt Blatt
- Ein Knoten $w \in V$ heißt Wurzel von G, falls alle Knoten $v \in V$ von w aus erreichbar sind (d.h. es gibt einen w-v-Weg in G)
- Ein gerichteter Wald, der eine Wurzel hat, heißt gerichteter Baum
- Da die Kantenrichtungen durch die Wahl der Wurzel eindeutig bestimmt sind, kann auf ihre Angabe verzichtet werden. Man spricht dann auch von einem Wurzelbaum

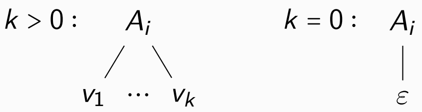
Syntaxbäume

Wir ordnen einer Ableitung

$$\underline{A_0} \Rightarrow l_1 \underline{A_1} r_1 \Rightarrow \cdots \Rightarrow l_{m-1} \underline{A_{m-1}} r_{m-1} \Rightarrow \alpha_m$$

den Syntaxbaum (oder Ableitungsbaum, engl. *parse tree*) T_m zu, wobei die Bäume $T_0, \ldots, T_m$ induktiv wie folgt definiert sind:

- T_0 besteht aus einem einzigen Knoten, der mit A_0 markiert ist
- Wird im $(i+1)$-ten Ableitungsschritt die Regel $A_i \to v_1 \ldots v_k$ mit $v_1, \ldots, v_k \in \Sigma \cup V$ angewandt, so ensteht T_{i+1} aus T_i, indem wir das Blatt A_i durch folgenden Unterbaum ersetzen:



- Hierbei stellen wir uns die Kanten von oben nach unten gerichtet und die Kinder $v_1 \ldots v_k$ von links nach rechts geordnet vor
- Syntaxbäume sind also geordnete Wurzelbäume

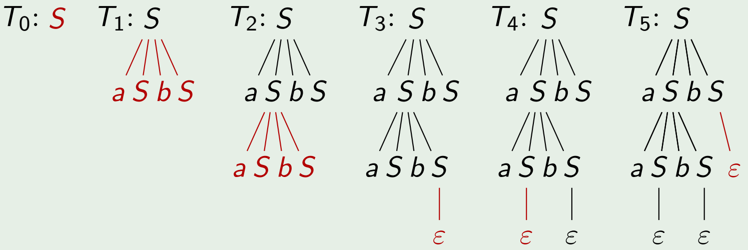
Syntaxbäume

Beispiel

- Betrachte die Grammatik $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$ und die Ableitung

$$\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aaSb\underline{S}bS \Rightarrow aa\underline{S}bbS \Rightarrow aabb\underline{S} \Rightarrow aabb$$

- Die zugehörigen Syntaxbäume sind dann

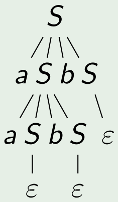


◁

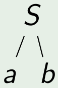
Syntaxbäume

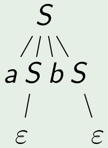
Beispiel

- In $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$ führen alle acht Ableitungen des Wortes *aabb* auf denselben Syntaxbaum:



- Dagegen führen in $G' = (\{S\}, \{a, b\}, \{S \to aSbS, ab, \varepsilon\}, S)$ die drei Ableitungen des Wortes *ab* auf zwei unterschiedliche Syntaxbäume:





Syntaxbäume und Linksableitungen

- Seien $T_0, \ldots, T_m$ die zu einer Ableitung $S = \alpha_0 \Rightarrow \cdots \Rightarrow \alpha_m$ gehörigen Syntaxbäume
- Dann haben alle Syntaxbäume $T_0, \ldots, T_m$ die Wurzel S
- Die Satzform α_i ergibt sich aus T_i, indem wir die Blätter von T_i von links nach rechts zu einem Wort zusammensetzen
- Auf den Syntaxbaum T_m führen neben $\alpha_0 \Rightarrow \cdots \Rightarrow \alpha_m$ alle Ableitungen, die sich von dieser nur in der Reihenfolge der Regelanwendungen unterscheiden
- Dazu gehört genau eine Linksableitung
- Linksableitungen und Syntaxbäume entsprechen sich also eineindeutig
- Dasselbe gilt für Rechtsableitungen
- Ist T Syntaxbaum einer CNF-Grammatik, so hat jeder Knoten in T höchstens zwei Kinder (d.h. T ist ein Binärbaum)

Abschätzung der Blätterzahl bei Binärbäumen

Definition

Die Tiefe eines Baumes mit Wurzel w ist die maximale Länge eines Weges von w zu einem Blatt

Lemma

Ein Binärbaum B der Tiefe $\leq k$ hat $\leq 2^k$ Blätter

Beweis durch Induktion über k:

$k = 0$: Ein Baum der Tiefe 0 kann nur einen Knoten haben

$k \rightsquigarrow k + 1$: Sei B ein Binärbaum der Tiefe $\leq k + 1$
Dann hängen an B's Wurzel maximal zwei Unterbäume
Da deren Tiefe $\leq k$ ist, haben sie nach IV $\leq 2^k$ Blätter
Also hat $B \leq 2^{k+1}$ Blätter $\square$

Mindesttiefe von Binärbäumen

Lemma

Ein Binärbaum B der Tiefe $\leq k$ hat $\leq 2^k$ Blätter

Korollar

Ein Binärbaum B mit $> 2^{k-1}$ Blättern hat eine Tiefe $\geq k$

Beweis

Wäre die Tiefe von B kleiner als k (also $\leq k-1$), so hätte B nach obigem Lemma $\leq 2^{k-1}$ Blätter (Widerspruch). $\square$

Beweis des Pumping-Lemmas für CFL

Satz (Pumping-Lemma für kontextfreie Sprachen)

Zu jeder kontextfreien Sprache $L \in \mathrm{CFL}$ gibt es eine Zahl l, so dass sich alle Wörter $z \in L$ mit $|z| \geq l$ in $z = uvwxy$ zerlegen lassen mit

1. $vx \neq \varepsilon$,
2. $|vwx| \leq l$ und
3. $uv^iwx^iy \in L$ für alle $i \geq 0$

Beweis

- Sei $G = (V, \Sigma, P, S)$ eine CNF-Grammatik für $L \setminus \{\varepsilon\}$
- Ist nun $z = z_1 \ldots z_n \in L$ mit $n \geq 1$, so ex. in G eine Ableitung

$$S = \alpha_0 \Rightarrow \alpha_1 \cdots \Rightarrow \alpha_m = z$$

- Da G in CNF ist, werden hierbei genau $n - 1$ Regeln der Form $A \to BC$ und genau n Regeln der Form $A \to a$ angewandt

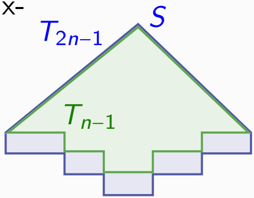
Beweis des Pumping-Lemmas für CFL

Beweis

- Sei $G = (V, \Sigma, P, S)$ eine CNF-Grammatik für $L \setminus \{\varepsilon\}$
- Ist nun $z = z_1 \dots z_n \in L$ mit $n \geq 1$, so ex. in G eine Ableitung

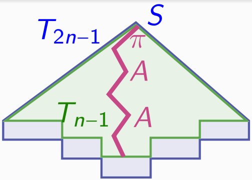
  $$S = \alpha_0 \Rightarrow \cdots \Rightarrow \alpha_m = z \text{ mit zugehörigen Syntaxbäumen } T_0, \dots, T_m$$

- Da G in CNF ist, werden hierbei genau $n - 1$ Regeln der Form $A \to BC$ und genau n Regeln der Form $A \to a$ angewandt
- Folglich ist $m = 2n - 1$ und wir können annehmen, dass die Regeln der Form $A \to BC$ vor den Regeln der Form $A \to a$ zur Anwendung kommen
- Dann besteht α_{n-1} aus n Variablen und die Syntaxbäume T_{2n-1} und T_{n-1} haben genau n Blätter
- Setzen wir $l = 2^k$, wobei $k = \|V\|$ ist, so hat T_{n-1} im Fall $n \geq l$ mindestens die Tiefe k, da T_{n-1} mindestens $l = 2^k > 2^{k-1}$ Blätter hat

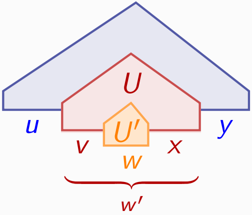


Beweis des Pumping-Lemmas für CFL

Beweis (Fortsetzung)

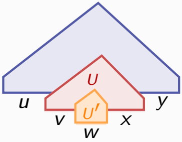


- Setzen wir $l = 2^k$, wobei $k = \|V\|$ ist, so hat T_{n-1} im Fall $n \geq l$ mindestens die Tiefe k, da T_{n-1} mindestens $l = 2^k > 2^{k-1}$ Blätter hat
- Sei π ein von der Wurzel ausgehender Pfad maximaler Länge in T_{n-1}
- Dann hat π mindestens die Länge k und unter den letzten $k+1$ Knoten von π müssen zwei mit derselben Variablen A markiert sein
- Seien U und U' die Unterbäume von T_{2n-1} mit diesen Knoten als Wurzel
- Dann hat U höchstens $l = 2^k$ Blätter und U' hat weniger Blätter als U
- Nun zerlegen wir z wie folgt:
  - w' ist das Teilwort von $z = uw'y$, das von U erzeugt wird und
  - w ist das Teilwort von $w' = vwx$, das von U' erzeugt wird.

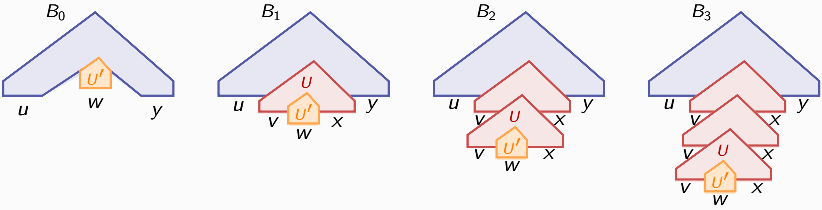


Beweis des Pumping-Lemmas für CFL

Beweis (Schluss)



- Dann ist $vx \neq \varepsilon$ (Bed. 1), da U mehr Blätter als U' hat
- Zudem gilt $|vwx| \leq l$ (Bed. 2), da U höchstens $2^k = l$ Blätter hat (sonst hätte der Baum $U^* = U \cap T_{n-1}$ eine Tiefe größer k und π wäre nicht maximal)
- Schließlich lassen sich Syntaxbäume B_i für die Wörter uv^iwx^iy, $i \geq 0$, wie folgt konstruieren (Bed. 3):
  - B_0 entsteht aus $B_1 = T_{2n-1}$, indem wir U durch U' ersetzen.
  - B_{i+1} entsteht aus B_i, indem wir U' durch U ersetzen:



□

Das Wortproblem für CFL

Das Wortproblem für kontextfreie Grammatiken

Gegeben: Eine kontextfreie Grammatik G und ein Wort x

Gefragt: Ist $x \in L(G)$?

Frage

Wie lässt sich das Wortproblem für kontextfreie Grammatiken entscheiden?

Der CYK-Algorithmus

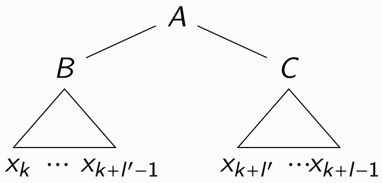
- Sei eine Grammatik $G = (V, \Sigma, P, S)$ und ein Wort $x = x_1 \dots x_n$ gegeben
- Falls $x = \varepsilon$ ist, können wir effizient prüfen, ob $S \Rightarrow^* \varepsilon$ gilt
- Hierzu genügt es, die Menge $E = \{A \in V \mid A \Rightarrow^* \varepsilon\}$ aller ε-ableitbaren Variablen zu berechnen und zu prüfen, ob $S \in E$ ist
- Andernfalls bringen wir G in CNF und starten den nach seinen Autoren Cocke, Younger und Kasami benannten CYK-Algorithmus
- Dieser bestimmt mittels dynamischer Programmierung für $l = 1, \dots, n$ und $k = 1, \dots, n - l + 1$ die Menge $V_{l,k}$ aller Variablen, aus denen das Teilwort $x_k \dots x_{k+l-1}$ ableitbar ist
- Dann gilt $x \in L(G) \Leftrightarrow S \in V_{n,1}$

Berechnung der Mengen $V_{l,k}$

- Sei $G = (V, \Sigma, P, S)$ eine CNF-Grammatik und sei $x \in \Sigma^+$
- Dann lassen sich die Mengen $V_{l,k} = \{A \in V \mid A \Rightarrow^* x_k \dots x_{k+l-1}\}$ wie folgt bestimmen
- Für $l = 1$ gehört A zu $V_{1,k}$, falls die Regel $A \to x_k$ existiert:

$$V_{1,k} = \{A \in V \mid A \to x_k\}$$

- Für $l > 1$ gehört A zu $V_{l,k}$, falls eine Regel $A \to BC$ und eine Zahl $l' \in \{1, \dots, l-1\}$ ex. mit $B \in V_{l',k}$ und $C \in V_{l-l',k+l'}$:



$$V_{l,k} = \{A \in V \mid \exists\, l' < l,\ B \in V_{l',k},\ C \in V_{l-l',k+l'} : A \to BC \in P\}$$

Der CYK-Algorithmus

Algorithmus $\texttt{CYK}(G, x)$

Input: CNF−Grammatik $G = (V, \Sigma, P, S)$ und Wort $x = x_1 \ldots x_n$

  for $k := 1$ **to** n **do**

    $V_{1,k} := \{A \in V \mid A \to x_k \in P\}$

  for $l := 2$ **to** n **do**

    for $k := 1$ **to** $n - l + 1$ **do**

      $V_{l,k} := \varnothing$

      for $l' := 1$ **to** $l - 1$ **do**

        for all $A \to BC \in P$ **do**

          if $B \in V_{l',k}$ **and** $C \in V_{l-l',k+l'}$ **then**

            $V_{l,k} := V_{l,k} \cup \{A\}$

  if $S \in V_{n,1}$ **then accept else reject**

Der CYK-Algorithmus lässt sich dahingehend erweitern, dass er im Fall $x \in L(G)$ auch einen Syntaxbaum T von x bestimmt

Beispiel

- Betrachte die CNF-Grammatik mit den Regeln

$$P\colon\ S \to AS', AY, BX, CS, c, \quad S' \to BC, \quad X \to AS, BX', a, \quad X' \to XX,$$
$$Y \to BS, AY', b, \quad Y' \to YY, \quad A \to a, \quad B \to b, \quad C \to c$$

- Dann erhalten wir für das Wort $x = abb$ folgende Mengen $V_{l,k}$:

k:	1	2	3
	a	b	b
l: 1	$\{X, A\}$	$\{Y, B\}$	$\{Y, B\}$
2	$\{S\}$	$\{Y'\}$	
3	$\{Y\}$		

- Wegen $S \notin V_{3,1}$ ist $x \notin L(G)$

Der CYK-Algorithmus

Beispiel (Fortsetzung)

- Betrachte die CNF-Grammatik mit den Regeln

$$P\colon\ S \to AS', AY, BX, CS, c,\quad S' \to BC,\quad X \to AS, BX', a,\quad X' \to XX,$$
$$Y \to BS, AY', b,\quad Y' \to YY,\quad A \to a,\quad B \to b,\quad C \to c$$

- Dagegen gehört das Wort $y = aababb$ zu $L(G)$:

a	a	b	a	b	b
$\{X, A\}$	$\{X, A\}$	$\{Y, B\}$	$\{X, A\}$	$\{Y, B\}$	$\{Y, B\}$
$\{X'\}$	$\{S\}$	$\{S\}$	$\{S\}$	$\{Y'\}$	
$\{X\}$	$\{X\}$	$\{Y\}$	$\{Y\}$		
$\{X'\}$	$\{S\}$	$\{Y'\}$			
$\{X\}$	$\{Y\}$				
$\{S\}$					

◁

Ein Maschinenmodell für die kontextfreien Sprachen

Frage

Wie lässt sich das Maschinenmodell des DFA erweitern, um die Sprache

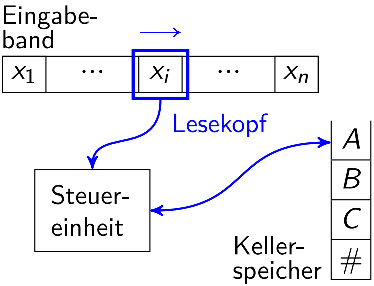
$$L = \{a^n b^n \mid n \geq 0\}$$

und alle anderen kontextfreien Sprachen erkennen zu können?

Antwort

- Ein DFA kann Sprachen wie L nicht erkennen, da er nur seinen Zustand als Speicher benutzen kann und die Anzahl der Zustände zwar von L aber nicht von der Eingabe abhängen darf
- Um kontextfreie Sprachen erkennen zu können, genügt bereits ein Kellerspeicher (auch Stapel, engl. *stack* oder *pushdown memory*)
- Dieser erlaubt nur den Zugriff auf die höchste belegte Speicheradresse

Der Kellerautomat



- verfügt zusätzlich über einen Kellerspeicher
- kann auch ε-Übergänge machen
- hat Lesezugriff auf das aktuelle Eingabezeichen und auf das oberste Kellersymbol
- kann in jedem Rechenschritt das oberste Kellersymbol löschen und durch beliebig viele Symbole ersetzen

Kellerautomaten mit Endzuständen

Notation

Für eine Menge M bezeichne $\mathcal{P}_e(M)$ die Menge aller endlichen Teilmengen von M, d.h. $\mathcal{P}_e(M) = \{A \subseteq M \mid A \text{ ist endlich}\}$

Definition

Ein Kellerautomat mit Endzuständen (auch FS-PDA für engl. *final state pushdown automaton*) ist ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$. Dabei ist

- $Z \neq \emptyset$ eine endliche Menge von Zuständen
- Σ das Eingabealphabet
- Γ das Kelleralphabet
- $\delta : Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to \mathcal{P}_e(Z \times \Gamma^*)$ die Überführungsfunktion
- $q_0 \in Z$ der Startzustand
- $\# \in \Gamma$ das Kelleranfangszeichen
- $E \subseteq Z$ die Menge der Endzustände

Arbeitsweise eines Kellerautomaten

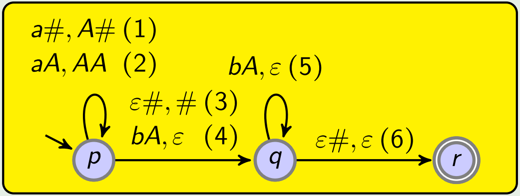
- Wenn p der momentane Zustand, A das oberste Kellerzeichen und $u \in \Sigma$ das nächste Eingabezeichen (bzw. $u = \varepsilon$) ist, so kann M im Fall $(q, B_1 \dots B_k) \in \delta(p, u, A)$
  - in den Zustand q wechseln,
  - den Lesekopf auf dem Eingabeband um $|u| \in \{0, 1\}$ Positionen vorrücken und
  - das Zeichen A aus- sowie die Zeichenfolge $B_1 \dots B_k$ einkellern (danach ist B_1 das oberste Kellerzeichen)
- Hierfür sagen wir auch, M führt die Anweisung $puA \to qB_1 \dots B_k$ aus, und sprechen im Fall
  - $k = 0$ von einer pop-Operation,
  - $k = 2$ und $B_2 = A$ von einer push-Operation, sowie
  - im Fall $u = \varepsilon$ von einem ε-Übergang
- Man beachte, dass bei leerem Keller kein Übergang mehr möglich ist

Ein Kellerautomat mit Endzuständen

Beispiel

- Sei $M = (Z, \Sigma, \Gamma, \delta, p, \#, E)$ mit $Z = \{p, q, r\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \#\}$, $E = \{r\}$ und der Überführungsfunktion

$$
\begin{aligned}
\delta : pa\# &\to pA\# && (1)\\
paA &\to pAA && (2)\\
p\varepsilon\# &\to q\# && (3)\\
pbA &\to q && (4)\\
qbA &\to q && (5)\\
q\varepsilon\# &\to r && (6)
\end{aligned}
$$



◁

Konfiguration eines Kellerautomaten

- Eine Konfiguration von M wird durch ein Tripel

  $$K = (p, x_i \dots x_n, A_1 \dots A_l) \in Z \times \Sigma^* \times \Gamma^*$$

  beschrieben und besagt, dass
  - p der momentane Zustand,
  - $x_i \dots x_n$ der ungelesene Rest der Eingabe und
  - $A_1 \dots A_l$ der aktuelle Kellerinhalt ist (A_1 ist oberstes Symbol)
- In einer solchen Konfiguration K kann M eine beliebige Anweisung $puA_1 \to qB_1 \dots B_k$ mit $u \in \{\varepsilon, x_i\}$ ausführen
- Diese überführt M in die Folgekonfiguration

  $$K' = (q, x_j \dots x_n, B_1 \dots B_k A_2 \dots A_l) \text{ mit } j = i + |u|$$

- Hierfür schreiben wir auch kurz $K \vdash K'$
- Eine Rechnung von M bei Eingabe x ist eine Folge von Konfigurationen

  $$K_0, K_1, K_2 \dots \text{ mit } K_0 \vdash K_1 \vdash K_2 \dots ,$$

  wobei $K_0 = (q_0, x, \#)$ die Startkonfiguration von M bei Eingabe x ist

Akzeptanz durch Erreichen eines Endzustands

Notation

Die reflexive, transitive Hülle von $\vdash$ bezeichnen wir wie üblich mit $\vdash^*$

Definition

Die von einem Kellerautomaten $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ akzeptierte (oder erkannte) Sprache ist

$$L(M) = \{x \in \Sigma^* \mid \exists\, q \in E, \alpha \in \Gamma^* : (q_0, x, \#) \vdash^* (q, \varepsilon, \alpha)\}$$

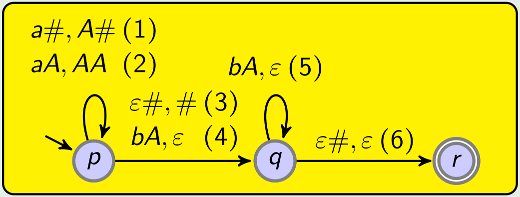
- Ein Kellerautomat M mit Endzuständen akzeptiert also genau dann ein Wort x, wenn es bei dieser Eingabe eine Rechnung gibt, bei der M
  - alle Eingabezeichen liest und
  - einen Endzustand $q \in E$ erreicht

Ein Kellerautomat mit Endzuständen

Beispiel (Fortsetzung)

- Sei $M = (Z, \Sigma, \Gamma, \delta, p, \#, E)$ mit $Z = \{p, q, r\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \#\}$, $E = \{r\}$ und der Überführungsfunktion

$$
\begin{array}{rll}
\delta: pa\# & \to pA\# & (1) \\
paA & \to pAA & (2) \\
p\varepsilon\# & \to q\# & (3) \\
pbA & \to q & (4) \\
qbA & \to q & (5) \\
q\varepsilon\# & \to r & (6)
\end{array}
$$



- Dann akzeptiert M die Eingabe $x = aabb$:

$$
(p, aabb, \#) \underset{(1)}{\vdash} (p, abb, A\#) \underset{(2)}{\vdash} (p, bb, AA\#) \underset{(4)}{\vdash} (q, b, A\#) \underset{(5)}{\vdash} (q, \varepsilon, \#) \underset{(6)}{\vdash} (r, \varepsilon, \varepsilon)
$$

◁

Akzeptanz durch Leeren des Kellers

Es gibt auch ein Akzeptanzkriterium, das keine Endzustände erfordert.

Definition

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ ein Kellerautomat ohne Endzustandsmenge
- Die von M durch Leeren des Kellers akzeptierte (oder erkannte) Sprache ist

$$L(M) = \{x \in \Sigma^* \mid \exists\, p \in Z \colon (q_0, x, \#) \vdash^* (p, \varepsilon, \varepsilon)\}$$

- Wir nennen M auch einen ES-PDA (für engl. *empty stack pushdown automaton*) oder einfach PDA

- Ein PDA M akzeptiert also genau dann eine Eingabe, wenn es eine Rechnung gibt, bei der M alle Eingabezeichen liest und den Keller leert
- Es gilt (siehe Übungen)

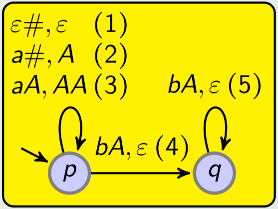
$$\{L(M) \mid M \text{ ist ein FS-PDA}\} = \{L(M) \mid M \text{ ist ein ES-PDA}\}$$

Ein PDA für die Sprache $\{a^n b^n \mid n \geq 0\}$

Beispiel

- Sei $M = (Z, \Sigma, \Gamma, \delta, p, \#)$ mit $Z = \{p, q\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \#\}$ und

$$\delta: p\varepsilon\# \to p\ (1) \quad pa\# \to pA\ (2) \quad paA \to pAA\ (3)$$
$$pbA \to q\ (4) \quad qbA \to q \quad (5)$$



- Dann akzeptiert M die Eingabe $x = aabb$:

$$(p, aabb, \#) \underset{(2)}{\vdash} (p, abb, A) \underset{(3)}{\vdash} (p, bb, AA) \underset{(4)}{\vdash} (q, b, A) \underset{(5)}{\vdash} (q, \varepsilon, \varepsilon)$$

- Allgemeiner akzeptiert M das Wort $x = a^n b^n$ mit folgender Rechnung:

$n = 0$: $(p, \varepsilon, \#) \underset{(1)}{\vdash} (p, \varepsilon, \varepsilon)$

$n \geq 1$: $(p, a^n b^n, \#) \underset{(2)}{\vdash} (p, a^{n-1} b^n, A) \underset{(3)}{\vdash}^{n-1} (p, b^n, A^n)$

$$\underset{(4)}{\vdash} (q, b^{n-1}, A^{n-1}) \underset{(5)}{\vdash}^{n-1} (q, \varepsilon, \varepsilon)$$

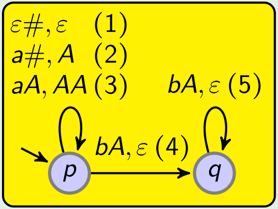
- Dies zeigt, dass M alle Wörter der Form $a^n b^n$, $n \geq 0$, akzeptiert

Ein PDA für die Sprache $\{a^n b^n \mid n \geq 0\}$

Beispiel (Fortsetzung)

- Sei $M = (Z, \Sigma, \Gamma, \delta, p, \#)$ mit $Z = \{p, q\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \#\}$ und

$$\delta : p\varepsilon\# \to p \ (1) \quad pa\# \to pA \ (2) \quad paA \to pAA \ (3)$$
$$pbA \to q \ (4) \quad qbA \to q \quad (5)$$



- Es gilt auch die umgekehrte Inklusion:

  alle Wörter $x = x_1 \ldots x_m \in L(M)$ haben die Form $x = a^n b^n$

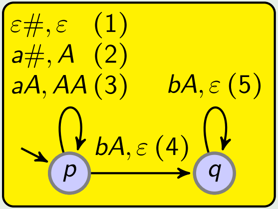
- Ausgehend von der Startkonfiguration $(p, x, \#)$ sind nämlich nur die Anweisungen (1) oder (2) ausführbar
- Führt M zuerst Anweisung (1) aus, so wird der Keller geleert
- Daher kann M in diesem Fall nur das leere Wort $x = \varepsilon = a^0 b^0$ akzeptieren
- Falls M mit Anweisung (2) beginnt, muss M später mittels Anweisung (4) in den Zustand q gelangen, da sonst der Keller nicht geleert wird

Ein PDA für die Sprache $\{a^n b^n \mid n \geq 0\}$

Beispiel (Schluss)

- Sei $M = (Z, \Sigma, \Gamma, \delta, p, \#)$ mit $Z = \{p, q\}$, $\Sigma = \{a, b\}$, $\Gamma = \{A, \#\}$ und

$$\begin{array}{llll} \delta : p\varepsilon\# \to p\ (1) & pa\# \to pA\ (2) & paA \to pAA\ (3) \\ \ pbA \to q\ (4) & qbA \to q \quad (5) & \end{array}$$



- Falls M mit Anweisung (2) beginnt, muss M später mittels Anweisung (4) in den Zustand q gelangen, da sonst der Keller nicht geleert wird
- Dies geschieht, sobald M nach Lesen von $n \geq 1$ a's das erste b liest:

$$\begin{aligned} (p, x_1 x_2 \ldots x_m, \#) &\underset{(2)}{\vdash} (p, x_2 \ldots x_n x_{n+1} \ldots x_m, A) \\ &\underset{(3)}{\vdash}^{n-1} (p, x_{n+1} x_{n+2} \ldots x_m, A^n) \underset{(4)}{\vdash} (q, x_{n+2} \ldots x_m, A^{n-1}) \end{aligned}$$

  mit $x_1 = x_2 = \cdots = x_n = a$ und $x_{n+1} = b$.

- Damit der Keller nach dem Lesen von x_m leer ist, muss M nun noch genau $n - 1$ b's lesen, weshalb $m = 2n$ und $x = a^n b^n$ folgt. $\triangleleft$

Ein Maschinenmodell für die Klasse CFL

Ziel

Als nächstes wollen wir zeigen, dass PDAs genau die kontextfreien Sprachen erkennen

Satz

CFL = $\{L(M) \mid M$ ist ein PDA$\}$

Beweis von $\mathrm{CFL} \subseteq \{L(M) \mid M \text{ ist ein PDA}\}$

Idee:

Konstruiere zu einer kontextfreien Grammatik $G = (V, \Sigma, P, S)$ einen PDA $M = (\{z\}, \Sigma, \Gamma, \delta, z, S)$ mit $\Gamma = V \cup \Sigma$ und folgenden Anweisungen:

- für jede Regel $A \rightarrow_G \alpha$ die Anweisung $z\varepsilon A \rightarrow z\alpha$
- für jedes Zeichen $a \in \Sigma$ die Anweisung $zaa \rightarrow z\varepsilon$

Beweis von CFL $\subseteq \{L(M) \mid M \text{ ist ein PDA}\}$

Beispiel

- Betrachte die Grammatik $G = (\{S\}, \{a, b\}, P, S)$ mit den Regeln

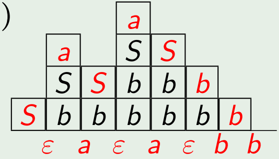
$$P\colon S \to aSb \ \ (1) \quad S \to \varepsilon \ \ (2)$$

- Der zugehörige PDA besitzt dann die Anweisungen

$$\delta: \ zaa \to z \ \ (0) \quad zbb \to z \ \ (0') \quad z\varepsilon S \to zaSb \ \ (1') \quad z\varepsilon S \to z \ \ (2')$$

- Der Linksableitung $\underline{S} \underset{(1)}{\Rightarrow} a\underline{S}b \underset{(1)}{\Rightarrow} aa\underline{S}bb \underset{(2)}{\Rightarrow} aabb$ in G entspricht dann die Rechnung

$$(z, aabb, S) \underset{(1')}{\vdash} (z, aabb, aSb) \underset{(0)}{\vdash} (z, abb, Sb)$$
$$\underset{(1')}{\vdash} (z, abb, aSbb) \underset{(0)}{\vdash} (z, bb, Sbb)$$
$$\underset{(2')}{\vdash} (z, bb, bb) \underset{(0')}{\vdash} (z, b, b) \underset{(0')}{\vdash} (z, \varepsilon, \varepsilon)$$



von M und umgekehrt ◁

Beweis von CFL $\subseteq \{L(M) \mid M$ ist ein PDA$\}$

Idee:

Konstruiere zu einer kontextfreien Grammatik $G = (V, \Sigma, P, S)$ einen PDA $M = (\{z\}, \Sigma, \Gamma, \delta, z, S)$ mit $\Gamma = V \cup \Sigma$ und folgenden Anweisungen:

- für jede Regel $A \rightarrow_G \alpha$ die Anweisung $z\varepsilon A \rightarrow z\alpha$
- für jedes Zeichen $a \in \Sigma$ die Anweisung $zaa \rightarrow z\varepsilon$

- M versucht also, eine Linksableitung für die Eingabe x zu finden. Da M hierbei den Syntaxbaum von oben nach unten aufbaut, wird M als *Top-Down Parser* bezeichnet
- Zudem gilt $S \Rightarrow_L^m x_1 \dots x_n$ gdw. $(z, x_1 \dots x_n, S) \vdash^{m+n} (z, \varepsilon, \varepsilon)$
- Daher folgt

$$x \in L(G) \Leftrightarrow S \Rightarrow_L^* x \Leftrightarrow (z, x, S) \vdash^* (z, \varepsilon, \varepsilon) \Leftrightarrow x \in L(M)$$

$\square$

Beweis von $\{L(M) \mid M \text{ ist ein PDA}\} \subseteq \mathsf{CFL}$

Vorbetrachtung:

- Obige Konstruktion eines PDA M aus einer kontextfreien Grammatik lässt sich leicht umdrehen, falls M nur einen Zustand hat
- Zu einem solchen PDA $M = (\{z\}, \Sigma, \Gamma, \delta, z, \#)$ lässt sich wie folgt eine kontextfreie Grammatik $G = (V, \Sigma, P, X_\#)$ mit $L(G) = L(M)$ konstruieren:
  - Die Variablenmenge von G ist $V = \{X_A \mid A \in \Gamma\}$ (wir können auch o.B.d.A. $\Sigma \cap \Gamma = \varnothing$ annehmen und $V = \Gamma$ setzen)
  - die Startvariable von G ist $X_\#$ und
  - P enthält für jede Anweisung $zuA \to zA_1 \ldots A_k$ von M die Regel

$$X_A \to uX_{A_1} \ldots X_{A_k}$$

- Dann lässt sich jede akzeptierende Rechnung $(z, x, \#) \vdash^m (z, \varepsilon, \varepsilon)$ von $M(x)$ der Länge m direkt in eine Linksableitung $X_\# \Rightarrow_L^m x$ in G der Länge m transformieren und umgekehrt

Beweis von $\{L(M) \mid M \text{ ist ein PDA}\} \subseteq \mathrm{CFL}$

Beispiel

- Betrachte den PDA $M = (\{z\}, \{a,b\}, \{S,a,b\}, \delta, z, S)$ mit
  $$\delta: \; zaa \to z \;\; (1) \quad zbb \to z \;\; (2) \quad z\varepsilon S \to zaSb \;\; (3) \quad z\varepsilon S \to z \;\; (4)$$
  den wir aus der Grammatik $G = (\{S\}, \{a,b\}, P, S)$ mit den beiden Regeln $S \to aSb, \varepsilon$ konstruiert haben
- Dann führt M auf die Grammatik $G' = (\{X_S, X_a, X_b\}, \{a,b\}, P', X_S)$ mit
  $$P': X_a \to a \;\; (1') \quad X_b \to b \;\; (2') \quad X_S \to X_a X_S X_b \;\; (3') \quad X_S \to \varepsilon \;\; (4')$$
- Der Rechnung
  $$(z, ab, S) \underset{(3)}{\vdash} (z, ab, aSb) \underset{(1)}{\vdash} (z, b, Sb) \underset{(4)}{\vdash} (z, b, b) \underset{(2)}{\vdash} (z, \varepsilon, \varepsilon)$$
  von M entspricht dann folgende Linksableitung in G (und umgekehrt):
  $$\underline{X_S} \underset{(3')}{\Rightarrow} \underline{X_a} X_S X_b \underset{(1')}{\Rightarrow} a \underline{X_S} X_b \underset{(4')}{\Rightarrow} a \underline{X_b} \underset{(2')}{\Rightarrow} ab$$
- Man beachte, dass G' eine aufgeblähte Variante von G ist. ◁

Simulation eines PDA durch einen single-state PDA

Idee:

Transformiere einen PDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ wie folgt in einen äquivalenten PDA $M' = (\{z\}, \Sigma, \Gamma', \delta', z, S)$ mit nur einem Zustand:

- Das Kelleralphabet ist $\Gamma' = \{S\} \cup \{X_{pAq} \mid A \in \Gamma,\ p, q \in Z\}$ und
- die Überführungsfunktion δ' enthält folgende Anweisungen:
  - für jeden Zustand $q \in Z$ die Anweisung $z\varepsilon S \to zX_{q_0\#q}$ und
  - für jede Anweisung $p_0 u A_0 \to p_1 A_1 \dots A_k$, $k \geq 0$, von M und für jede Folge von k Zuständen $p_2, \dots, p_{k+1} \in Z$ die Anweisung

    $$zuX_{p_0 A_0 p_{k+1}} \to zX_{p_1 A_1 p_2} \dots X_{p_k A_k p_{k+1}}$$

- Dabei rät M' durch die Wahl der Anweisung
  - $z\varepsilon S \to zX_{q_0\#q}$ den Zustand q, den M mit leerem Keller (also im letzten Rechenschritt) erreicht, und
  - $zuX_{p_0 A_0 p_{k+1}} \to zX_{p_1 A_1 p_2} \dots X_{p_k A_k p_{k+1}}$ für $i = 1, \dots, k-1$ die Zustände p_{i+1}, die M bei Freigabe der mit A_i belegten Speicherzelle erreicht
- Zudem verifiziert M' bei jeder pop-Operation $zuX_{p_0 A_0 p_1} \to z$, dass M den (geratenen) Zustand p_1 nach Entfernen von A_0 auch erreichen kann

Simulation eines PDA durch einen single-state PDA

Beispiel

- Betrachte den PDA $M = (\{p,q\},\{a,b\},\{A,\#\},\delta,p,\#)$ mit den Anweisungen

$$\begin{aligned} \delta: p\varepsilon\# &\to q \quad (1) & pa\# &\to pA \quad (2) & paA &\to pAA \quad (3) \\ pbA &\to q \quad (4) & qbA &\to q \quad (5) \end{aligned}$$

- Der zugehörige PDA $M' = (\{z\},\{a,b\},\Gamma',\delta',z,S)$ mit nur einem Zustand hat dann das Kelleralphabet

$$\Gamma' = \{S, X_{p\#p}, X_{p\#q}, X_{q\#p}, X_{q\#q}, X_{pAp}, X_{pAq}, X_{qAp}, X_{qAq}\}$$

Simulation eines PDA durch einen single-state PDA

Beispiel (Fortsetzung)

- Zudem enthält M' neben den beiden Anweisungen $\cancel{z\varepsilon S \to zX_{p\#p}\ (0)}$ und $z\varepsilon S \to zX_{p\#q}$ $(0')$ die folgenden Anweisungen:

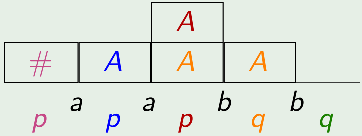
Anweisung von M		k	$p_2, \ldots, p_{k+1}$	Anweisungen von M'	
$p\varepsilon\# \to q$	(1)	0	-	$z\varepsilon X_{p\#q} \to z$	$(1')$
$pa\# \to pA$	(2)	1	p	$\cancel{zaX_{p\#p} \to zX_{pAp}}$	$\cancel{(2')}$
			q	$zaX_{p\#q} \to zX_{pAq}$	$(2'')$
$paA \to pAA$	(3)	2	p, p	$zaX_{pAp} \to zX_{pAp}X_{pAp}$	$(3')$
			p, q	$zaX_{pAq} \to zX_{pAp}X_{pAq}$	$(3'')$
			q, p	$\cancel{zaX_{pAp} \to zX_{pAq}X_{qAp}}$	$\cancel{(3''')}$
			q, q	$zaX_{pAq} \to zX_{pAq}X_{qAq}$	$(3'''')$
$pbA \to q$	(4)	0	-	$zbX_{pAq} \to z$	$(4')$
$qbA \to q$	(5)	0	-	$zbX_{qAq} \to z$	$(5')$

Simulation eines PDA durch einen single-state PDA

Beispiel (Schluss)

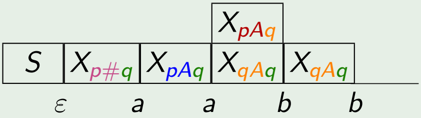
- Der (akzeptierenden) Rechnung

$$(p, aabb, \#) \underset{(2)}{\vdash} (p, abb, A) \underset{(3)}{\vdash} (p, bb, AA) \underset{(4)}{\vdash} (q, b, A) \underset{(5)}{\vdash} (q, \varepsilon, \varepsilon)$$



von M entspricht dann folgende Rechnung von M':

$$(z, aabb, S) \underset{(0')}{\vdash} (z, aabb, X_{p\#q}) \underset{(2'')}{\vdash} (z, abb, X_{pAq})$$
$$\underset{(3'''')}{\vdash} (z, bb, X_{pAq}X_{qAq}) \underset{(4')}{\vdash} (z, b, X_{qAq}) \underset{(5')}{\vdash} (z, \varepsilon, \varepsilon)$$



◁

Simulation eines PDA durch einen single-state PDA

- Es bleibt noch zu zeigen, dass der zu einem PDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ konstruierte PDA $M' = (\{z\}, \Sigma, \Gamma', \delta', z, S)$ mit dem Kelleralphabet $\Gamma' = \{S\} \cup \{X_{pAq} \mid p, q \in Z, A \in \Gamma\}$, der
  - für jeden Zustand $q \in Z$ die Anweisung $z\varepsilon S \to zX_{q_0\#q}$ sowie
  - für jede Anweisung $puA \to p_1 A_1 \dots A_k$, $k \geq 0$, von M und jede Folge $p_2, \dots, p_{k+1}$ die Anweisung $zuX_{pAp_{k+1}} \to zX_{p_1A_1p_2} \dots X_{p_kA_kp_{k+1}}$

  enthält, die gleiche Sprache wie M akzeptiert.
- Hierzu zeigen wir, dass jede Rechnung $(p, x, A) \vdash^m (q, \varepsilon, \varepsilon)$ von M einer Rechnung $(z, x, X_{pAq}) \vdash^m (z, \varepsilon, \varepsilon)$ von M' entspricht und umgekehrt
- Aus dieser Äquivalenz folgt dann sofort $L(M) = L(M')$:

$$
\begin{aligned}
x \in L(M) \quad &\Leftrightarrow \quad M \text{ hat für ein } q \in Z \text{ eine akzeptierende Rechnung } \\
&\qquad (q_0, x, \#) \vdash^m (q, \varepsilon, \varepsilon) \text{ der Länge } m \geq 1 \\
&\Leftrightarrow \quad M' \text{ hat für ein } q \in Z \text{ eine akzeptierende Rechnung } \\
&\qquad (z, x, S) \vdash (z, x, X_{q_0\#q}) \vdash^m (z, \varepsilon, \varepsilon) \text{ mit } m \geq 1 \\
&\Leftrightarrow \quad x \in L(M')
\end{aligned}
$$

Es bleibt noch zu zeigen, dass für alle $p, q \in Z$, $A \in \Gamma$, $x \in \Sigma^*$ und $m \geq 0$ gilt:

$$(p, x, A) \vdash_M^m (q, \varepsilon, \varepsilon) \text{ gdw. } (z, x, X_{pAq}) \vdash_{M'}^m (z, \varepsilon, \varepsilon) \qquad (*)$$

Induktionsanfang ($m = 0$):

Da weder M noch M' in $m = 0$ Rechenschritten den Keller leeren können, gilt die Äquivalenz $(*)$ für $m = 0$

Induktionsschritt ($m \rightsquigarrow m+1$):

- Sei eine Rechnung $(p, x, A) \vdash^{m+1} (q, \varepsilon, \varepsilon)$ der Länge $m+1$ von M gegeben und sei $puA \to p_1 A_1 \dots A_k$ die im ersten Rechenschritt ausgeführte Anweisung:

$$(p, x, A) \vdash (p_1, x', A_1 \dots A_k) \vdash^{m} (q, \varepsilon, \varepsilon)$$

- Im Fall $k \geq 2$ sei p_i für $i = 2, \dots, k$ der Zustand, in den M mit dem Kellerinhalt $A_i \dots A_k$ gelangt
- Zudem sei u_i für $i = 1, \dots, k$ das zwischen den Besuchen von p_i und p_{i+1} gelesene Teilwort von x, wobei $p_{k+1} = q$ ist
- Dann gilt $x = ux'$ und $x' = u_1 \dots u_k$ sowie

$$(p_1, x', A_1 \dots A_k) \vdash^{*} (p_i, u_i \dots u_k, A_i \dots A_k) \vdash^{*} (q, \varepsilon, \varepsilon)$$

- Für $i = 1, \dots, k$ ex. daher Zahlen $m_i \geq 1$ mit

$$(p_i, u_i, A_i) \vdash^{m_i} (p_{i+1}, \varepsilon, \varepsilon) \text{ und } m_1 + \dots + m_k = m$$

Simulation eines PDA durch einen single-state PDA

Induktionsschritt ($m \rightsquigarrow m+1$):

- Für $i = 1, \dots, k$ ex. daher Zahlen $m_i \geq 1$ mit

$$(p_i, u_i, A_i) \vdash^{m_i} (p_{i+1}, \varepsilon, \varepsilon) \text{ und } m_1 + \cdots + m_k = m$$

- Daher hat M' nach IV die Rechnungen $(z, u_i, X_{p_i A_i p_{i+1}}) \vdash^{m_i} (z, \varepsilon, \varepsilon)$
- Zudem hat M' wegen $puA \to_M p_1 A_1 \dots A_k$ die Anweisung $zuX_{pAq} \to zX_{p_1 A_1 p_2} \dots X_{p_{k-1} A_{k-1} p_k} X_{p_k A_k q}$, so dass wir die gesuchte Rechnung der Länge $m+1$ von M' wie folgt erhalten:

$$\begin{array}{lll}
(z, x, X_{pAq}) & = & (z, uu_1 \dots u_k, X_{pAq}) \\
& \vdash & (z, u_1 \dots u_k, X_{p_1 A_1 p_2} \dots X_{p_{k-1} A_{k-1} p_k} X_{p_k A_k q}) \\
& \vdash^{m_1} & (z, u_2 \dots u_k, X_{p_2 A_2 p_3} \dots X_{p_{k-1} A_{k-1} p_k} X_{p_k A_k q}) \\
& \vdots & \\
& \vdash^{m_{k-1}} & (z, u_k, X_{p_k A_k q}) \\
& \vdash^{m_k} & (z, \varepsilon, \varepsilon)
\end{array}$$

- Entsprechend lässt sich umgekehrt aus jeder solchen Rechnung von M' eine Rechnung $(p, x, A) \vdash^{m+1} (q, \varepsilon, \varepsilon)$ von M gewinnen. $\square$

Direkte Konstruktion einer Grammatik aus einem PDA

- Wir können die beiden Schritte
  - PDA M → PDA M' mit nur einem Zustand und
  - PDA M' mit nur einem Zustand → kontextfreie Grammatik G

  zu einem Schritt zusammenfassen
- Dazu konstruieren wir wie folgt zu einem PDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#)$ eine äquivalente kontextfreie Grammatik $G = (V, \Sigma, P, S)$
- Die Variablenmenge von G ist $V = \{S\} \cup \{X_{pAq} \mid A \in \Gamma, p, q \in Z\}$
- Zudem fügen wir für jeden Zustand $q \in Z$ die Startregel

  $$S \to X_{q_0 \# q}$$

  und für jede Anweisung $puA \to p_1 A_1 \ldots A_k$, $k \geq 0$, von M und jede Zustandsfolge $p_2, \ldots, p_{k+1}$ die Regel

  $$X_{pAp_{k+1}} \to uX_{p_1 A_1 p_2} \ldots X_{p_k A_k p_{k+1}}$$

  zu P hinzu

Beweis von $\{L(M) \mid M \text{ ist ein PDA}\} \subseteq \mathrm{CFL}$

Beispiel

- Betrachte den PDA $M = (\{p, q\}, \{a, b\}, \{A, \#\}, \delta, p, \#)$ mit den Anweisungen

$$\begin{aligned} \delta : p\varepsilon\# &\to q \quad (1) & pa\# &\to pA \quad (2) & paA &\to pAA \quad (3) \\ pbA &\to q \quad (4) & qbA &\to q \quad (5) \end{aligned}$$

- Dann erhalten wir die Grammatik $G = (V, \Sigma, P, S)$ mit der Variablenmenge

$$V = \{S, X_{p\#p}, X_{p\#q}, X_{q\#p}, X_{q\#q}, X_{pAp}, X_{pAq}, X_{qAp}, X_{qAq}\}$$

- Die Regelmenge P enthält die beiden Startregeln

$$S \to \cancel{X_{p\#p}}, X_{p\#q} \quad (0, 0')$$

Beweis von $\{L(M) \mid M \text{ ist ein PDA}\} \subseteq \mathrm{CFL}$

Beispiel (Fortsetzung)

- Zudem enthält P die folgenden Produktionen:

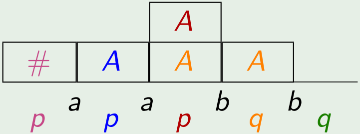
Anweisung		k	$p_2, \ldots, p_{k+1}$	zugehörige Regeln	
$p\varepsilon\# \to q$	(1)	0	-	$X_{p\#q} \to \varepsilon$	(1′)
$pa\# \to pA$	(2)	1	p	$\cancel{X_{p\#p} \to aX_{pAp}}$	~~(2′)~~
			q	$X_{p\#q} \to aX_{pAq}$	(2″)
$paA \to pAA$	(3)	2	p, p	$X_{pAp} \to aX_{pAp}X_{pAp}$	(3′)
			p, q	$X_{pAq} \to aX_{pAp}X_{pAq}$	(3″)
			q, p	$\cancel{X_{pAp} \to aX_{pAq}X_{qAp}}$	~~(3‴)~~
			q, q	$X_{pAq} \to aX_{pAq}X_{qAq}$	(3⁗)
$pbA \to q$	(4)	0	-	$X_{pAq} \to b$	(4′)
$qbA \to q$	(5)	0	-	$X_{qAq} \to b$	(5′)

Beweis von $\{L(M) \mid M \text{ ist ein PDA}\} \subseteq \mathrm{CFL}$

Beispiel (Schluss)

- Der (akzeptierenden) Rechnung

$$(p, aabb, \#) \underset{(2)}{\vdash} (p, abb, A) \underset{(3)}{\vdash} (p, bb, AA) \underset{(4)}{\vdash} (q, b, A) \underset{(5)}{\vdash} (q, \varepsilon, \varepsilon)$$



von M entspricht dann in G die Linksableitung

$$\underline{S} \underset{(0')}{\Rightarrow} \underline{X_{p\#q}} \underset{(2'')}{\Rightarrow} a\underline{X_{pAq}} \underset{(3'''')}{\Rightarrow} aa\underline{X_{pAq}}X_{qAq} \underset{(4')}{\Rightarrow} aab\underline{X_{qAq}} \underset{(5')}{\Rightarrow} aabb$$

$\triangleleft$

Deterministische Kellerautomaten

In der Praxis spielen det. Kellerautomaten eine wichtige Rolle.

Definition

Ein Kellerautomat M heißt deterministisch, falls die Relation $\vdash_M$ rechtseindeutig ist:

$$K \vdash_M K_1 \wedge K \vdash_M K_2 \Rightarrow K_1 = K_2$$

- Die Anzahl $N(K)$ der Folgekonfigurationen einer Konfiguration $K = (q, x_i \dots x_n, A_1 \dots A_k)$, $1 \le i \le n+1$, ist

$$N(K) = \begin{cases} 0, & k = 0 \\ \|\delta(q, \varepsilon, A_1)\|, & i = n+1 \text{ und } k \ge 1 \\ \|\delta(q, x_i, A_1)\| + \|\delta(q, \varepsilon, A_1)\|, & i \le n \text{ und } k \ge 1 \end{cases}$$

- Daher ist ein Kellerautomat $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ genau dann deterministisch, wenn die Überführungsfunktion δ für alle $(q, a, A) \in Z \times \Sigma \times \Gamma$ folgende Bedingung erfüllt:

$$\|\delta(q, a, A)\| + \|\delta(q, \varepsilon, A)\| \le 1$$

Deterministische Kellerautomaten

Beispiel

- Betrachte den PDA $M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{A, B, \#\}, \delta, q_0, \#)$ mit

$$\begin{array}{lllll} \delta: & q_0 a\# \to q_0 A\# & q_0 b\# \to q_0 B\# & q_0 aA \to q_0 AA & q_0 bA \to q_0 BA \\ & q_0 aB \to q_0 AB & q_0 bB \to q_0 BB & q_0 cA \to q_1 A & q_0 cB \to q_1 B \\ & q_1 aA \to q_1 & q_1 bB \to q_1 & q_1 \varepsilon\# \to q_2 & \end{array}$$

  Darstellung von δ in Tabellenform

δ	$q_0, \#$	q_0, A	q_0, B	$q_1, \#$	q_1, A	q_1, B	$q_2, \#$	q_2, A	q_2, B
ε				q_2					
a	$q_0 A\#$	$q_0 AA$	$q_0 AB$		q_1				
b	$q_0 B\#$	$q_0 BA$	$q_0 BB$			q_1			
c		$q_1 A$	$q_1 B$						

- Man beachte, dass jedes Tabellenfeld höchstens eine Anweisung enthält und jede Spalte mit einem ε-Eintrag keine weiteren Einträge enthält.
- Daher ist die Bedingung $\|\delta(q, a, A)\| + \|\delta(q, \varepsilon, A)\| \le 1$ für alle $q \in Z$, $a \in \Sigma$ und $A \in \Gamma$ erfüllt. $\triangleleft$

Wie mächtig sind deterministische ES-PDAs?

Frage

- Können deterministische ES-PDAs (also empty stack PDAs) zumindest alle regulären Sprachen (durch Leeren des Kellers) akzeptieren?
- Kann z.B. die Sprache $L = \{a, aa\}$ von einem deterministischen ES-PDA M akzeptiert werden?

Antwort: Nein

- Um $x = a$ zu akzeptieren, muss M den Keller nach Lesen von a leeren und kann somit keine anderen Wörter mit dem Präfix a akzeptieren
- Deterministische ES-PDAs können also nur präfixfreie Sprachen L akzeptieren (d.h. kein Wort $x \in L$ darf Präfix eines anderen Wortes $y \in L$ sein)

Deterministisch kontextfreie Sprachen

Definition

- Die Klasse der deterministisch kontextfreien Sprachen ist

  $$\mathrm{DCFL} = \{L(M) \mid M \text{ ist ein deterministischer FS-PDA}\}$$

  (für *engl. Deterministic Context Free Languages*)
- Ein deterministischer FS-PDA wird auch als FS-DPDA (für *engl. Finite State Deterministic Push Down Automaton*) oder einfach als DPDA bezeichnet

Abschlusseigenschaften von DCFL

Frage

Ist DCFL unter Komplementbildung abgeschlossen?

Antwort

Ja. Allerdings ergeben sich beim Versuch, einfach die End- und Nichtendzustände eines DPDA M zu vertauschen, um einen DPDA $\overline{M}$ für $\overline{L(M)}$ zu erhalten, folgende Schwierigkeiten:

1. Falls M eine Eingabe x nicht zu Ende liest, wird x weder von M noch von $\overline{M}$ akzeptiert.
2. Falls M nach dem Lesen von x noch ε-Übergänge ausführt und dabei End- und Nichtendzustände besucht, wird x von M und von $\overline{M}$ akzeptiert

DPDAs, die ihre Eingabe zu Ende lesen

Satz

Jede Sprache $L \in \mathrm{DCFL}$ wird von einem DPDA M' erkannt, der alle Eingaben zu Ende liest (und bei allen Eingaben hält)

Beweisskizze

Falls ein DPDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ eine Eingabe $x = x_1 \dots x_n$ nicht zu Ende liest, muss einer der folgenden drei Gründe vorliegen:

1. M gerät in eine Konfiguration $(q, x_i \dots x_n, \varepsilon)$, $i \le n$, mit leerem Keller
2. M gerät in eine Konfiguration $(q, x_i \dots x_n, A\gamma)$, $i \le n$, in der wegen $\delta(q, x_i, A) = \delta(q, \varepsilon, A) = \varnothing$ keine ausführbare Anweisung existiert
3. M gerät in eine Konfiguration $(q, x_i \dots x_n, A\gamma)$, $i \le n$, so dass M ausgehend von (q, ε, A) eine unendliche Folge von ε-Anweisungen ausführt

Dies lässt sich vermeiden, indem zu Beginn der Rechnung ein neues Kellerzeichen auf dem Kellerboden platziert wird und ein neuer Fehlerzustand hinzugefügt wird, in dem der Rest der Eingabe gelesen wird. □

Komplementabschluss von DCFL

Satz

Die Klasse DCFL ist unter Komplement abgeschlossen

Beweisskizze

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ ein DPDA, der alle Eingaben zu Ende liest, und sei $L(M) = L$
- Wir konstruieren einen DPDA $\overline{M}$ für $\overline{L}$, der M simuliert
- $\overline{M}$ hat $Z \times \{1, 2, 3\}$ als Zustands- und $Z \times \{3\}$ als Endzustandsmenge
- $\overline{M}$ merkt sich in seinem Zustand (q, i) neben dem aktuellen Zustand q von M, ob M nach Lesen des letzten Zeichens (bzw. seit Beginn der Rechnung) einen Endzustand besucht hat $(i = 1)$ oder nicht $(i = 2)$
- Möchte M das nächste Zeichen lesen und befindet sich $\overline{M}$ im Zustand $(q, 2)$, so macht $\overline{M}$ noch einen Umweg über den Endzustand $(q, 3)$, bevor die Simulation von M fortgesetzt wird $\square$

Man beachte, dass $\overline{M}$ in einem Endzustand keine ε-Übergänge macht.

Komplementabschluss von DCFL

Definition

Für eine Sprachklasse $\mathcal{C}$ bezeichne $\mathrm{co}\text{-}\mathcal{C}$ die Klasse $\{\bar{L} \mid L \in \mathcal{C}\}$ aller Komplemente von Sprachen in $\mathcal{C}$

Korollar

- REG $=$ co-REG,
- DCFL $=$ co-DCFL,
- CFL $\neq$ co-CFL

Weitere Abschlusseigenschaften von DCFL

Satz

Die Klasse DCFL ist nicht abgeschlossen unter Schnitt, Vereinigung, Produkt und Sternhülle

$A, B \in \mathrm{DCFL} \not\Rightarrow A \cap B \in \mathrm{DCFL}$

- Die beiden Sprachen

  $$L_1 = \{a^n b^m c^m \mid n, m \geq 0\} \quad \text{und} \quad L_2 = \{a^n b^n c^m \mid n, m \geq 0\}$$

  sind sogar deterministisch kontextfrei (siehe Übungen)
- Da $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\}$ nicht kontextfrei ist, liegt der Schnitt dieser Sprachen natürlich auch nicht in DCFL

$A, B \in \mathrm{DCFL} \not\Rightarrow A \cup B \in \mathrm{DCFL}$

- Da DCFL unter Komplementbildung abgeschlossen ist, kann DCFL wegen de Morgan dann auch nicht unter Vereinigung abgeschlossen sein
- Beispielsweise sind die Sprachen

  $$L_3 = \{a^i b^j c^k \mid i \neq j \wedge i, j, k \geq 1\} \text{ und } L_4 = \{a^i b^j c^k \mid j \neq k \wedge i, j, k \geq 1\}$$

  deterministisch kontextfrei (siehe Übungen)
- Ihre Vereinigung gehört aber nicht zu DCFL, d.h.

  $$L_3 \cup L_4 = \{a^i b^j c^k \mid (i \neq j \vee j \neq k) \wedge i, j, k \geq 1\} \in \mathrm{CFL} \setminus \mathrm{DCFL}$$

- DCFL ist nämlich unter Schnitt mit regulären Sprachen abgeschlossen (siehe Übungen)
- Daher wäre mit $L_3 \cup L_4$ auch die Sprache

  $$(\overline{L_3 \cup L_4}) \cap L(a^+ b^+ c^+) = \{a^n b^n c^n \mid n \geq 1\}$$

  (deterministisch) kontextfrei

$A, B \in \mathrm{DCFL} \not\Rightarrow AB \in \mathrm{DCFL}$

- Betrachte die DCFL Sprachen

$$L_3 = \{a^i b^j c^k \mid i \neq j \wedge i,j,k \geq 1\} \text{ und } L_4 = \{a^i b^j c^k \mid j \neq k \wedge i,j,k \geq 1\}$$

- Wir wissen bereits, dass $L = L_3 \cup L_4 \notin \mathrm{DCFL}$ ist
- Dann ist aber auch die Sprache

$$0L = 0L_3 \cup 0L_4 \notin \mathrm{DCFL},$$

da sich ein DPDA $M = (Z, \Sigma, \Gamma, \delta, q_0, \#, E)$ für $0L$ leicht zu einem DPDA M' für L umbauen ließe:

  - Sei (p, ε, γ) die Konfiguration, die M nach Lesen der Eingabe 0 erreicht
  - Dann erkennt der DPDA $M' = (Z \cup \{s\}, \Sigma, \Gamma, \delta', s, \#, E)$ die Sprache L, wobei δ' wie folgt definiert ist:

$$\delta'(q,u,A) = \begin{cases} (p,\gamma), & (q,u,A) = (s,\varepsilon,\#) \\ \delta(q,u,A), & (q,u,A) \in Z \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \end{cases}$$

$A, B \in \mathsf{DCFL} \not\Rightarrow AB \in \mathsf{DCFL}$

- Betrachte die DCFL Sprachen

$$L_3 = \{a^i b^j c^k \mid i \neq j \wedge i,j,k \geq 1\} \text{ und } L_4 = \{a^i b^j c^k \mid j \neq k \wedge i,j,k \geq 1\}$$

- In den Übungen wird gezeigt, dass auch die beiden Sprachen $\{\varepsilon, 0\}$ und $L_5 = L_3 \cup 0L_4$ in DCFL sind
- Ihr Produkt $\{\varepsilon, 0\}L_5 = L_5 \cup 0L_5 = L_3 \cup 0L_4 \cup 0L_3 \cup 00L_4$ gehört aber nicht zu DCFL
- Da DCFL unter Schnitt mit regulären Sprachen abgeschlossen ist, wäre andernfalls auch die Sprache

$$\{\varepsilon, 0\}L_5 \cap 0\{a,b,c\}^* = 0L_3 \cup 0L_4$$

in DCFL, was wir bereits ausgeschlossen haben

Bemerkung

Dass DCFL auch nicht unter Sternhüllenbildung abgeschlossen ist, lässt sich ganz ähnlich zeigen (siehe Übungen)

Abschlusseigenschaften

Abschlusseigenschaften der Klassen REG, DCFL und CFL

	Vereinigung	Schnitt	Komplement	Produkt	Sternhülle
REG	*ja*	*ja*	*ja*	*ja*	*ja*
DCFL	*nein*	*nein*	*ja*	*nein*	*nein*
CFL	*ja*	*nein*	*nein*	*ja*	*ja*

Charakterisierung von DCFL mittels Grammatiken

- Die Klasse DCFL lässt sich auch mit Hilfe von speziellen kontextfreien Grammatiken charakterisieren, den so genannten $LR(k)$-Grammatiken
- Der erste Buchstabe L steht für die Leserichtung bei der Syntaxanalyse, d.h. das Eingabewort x wird von links (nach rechts) gelesen
- Der zweite Buchstabe R bedeutet, dass bei der Syntaxanalyse eine Rechtsableitung entsteht
- Schließlich gibt der Parameter k an, wieviele Zeichen man über das aktuelle Eingabezeichen hinauslesen muss, damit der nächste Schritt eindeutig feststeht (k wird auch als *Lookahead* bezeichnet)
- Durch $LR(0)$-Grammatiken lassen sich nur die präfixfreien Sprachen in DCFL erzeugen
- Dagegen erzeugen die $LR(k)$-Grammatiken für jedes $k \geq 1$ genau die Sprachen in DCFL
- Daneben gibt es noch $LL(k)$-Grammatiken, die für wachsendes k immer mehr deterministisch kontextfreie Sprachen erzeugen

Kontextsensitive Sprachen

Definition

Sei $G = (V, \Sigma, P, S)$ eine Grammatik

1. G heißt vom Typ 3 oder regulär, falls für alle Regeln $u \to v$ gilt:

   $u \in V$ und $v \in \Sigma V \cup \Sigma \cup \{\varepsilon\}$

   (d.h. alle Regeln haben die Form $A \to aB$, $A \to a$ oder $A \to \varepsilon$)

2. G heißt vom Typ 2 oder kontextfrei, falls für alle Regeln $u \to v$ gilt:

   $u \in V$ (d.h. alle Regeln haben die Form $A \to \alpha$)

3. G heißt vom Typ 1 oder kontextsensitiv, falls für alle Regeln $u \to v$ gilt:

   $|v| \geq |u|$ (mit Ausnahme der ε-Sonderregel, s. unten)

4. Jede Grammatik ist automatisch vom Typ 0

Die ε-Sonderregel

In einer kontextsensitiven Grammatik ist auch die Regel $S \to \varepsilon$ zulässig, falls das Startsymbol S nicht auf der rechten Seite einer Regel vorkommt

CFL ist echt in CSL enthalten

Bemerkung

- Wie wir gesehen haben, ist CFL in CSL enthalten
- Zudem ist die Sprache $L = \{a^n b^n c^n \mid n \geq 1\}$ nicht kontextfrei
- L kann jedoch von einer kontextsensitiven Grammatik erzeugt werden (siehe nächste Folie)
- Daher ist CFL echt in CSL enthalten

Eine kontextsensitive Grammatik für $\{a^n b^n c^n \mid n \geq 1\}$

Beispiel (Schluss)

- Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit $V = \{S, B\}$, $\Sigma = \{a, b, c\}$ und den Regeln

$$P\colon\ S \to aSBc, abc\ (1,2) \quad cB \to Bc\ (3) \quad bB \to bb\ (4)$$

- In G lässt sich beispielsweise das Wort $w = aabbcc$ ableiten:

$$\underline{S} \underset{(1)}{\Rightarrow} a\underline{S}Bc \underset{(2)}{\Rightarrow} aab\underline{cB}c \underset{(3)}{\Rightarrow} aa\underline{bB}cc \underset{(4)}{\Rightarrow} aabbcc$$

- Allgemein gilt für alle $n \geq 1$:

$$\underline{S} \underset{(1)}{\Rightarrow}^{n-1} a^{n-1}\underline{S}(Bc)^{n-1} \underset{(2)}{\Rightarrow} a^{n-1}ab\underline{c(Bc)^{n-1}}$$

$$\underset{(3)}{\Rightarrow}^{\binom{n}{2}} a^n \underline{bB^{n-1}}c^n \underset{(4)}{\Rightarrow}^{n-1} a^n b^n c^n$$

- Also gilt $a^n b^n c^n \in L(G)$ für alle $n \geq 1$

Eine kontextsensitive Grammatik für $\{a^n b^n c^n \mid n \geq 1\}$

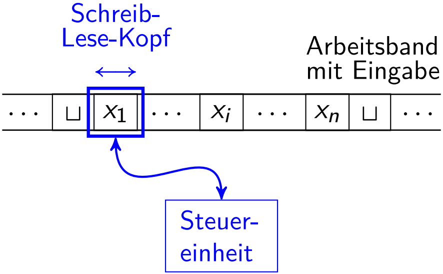
Beispiel (Schluss)

- Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit $V = \{S, B\}$, $\Sigma = \{a, b, c\}$ und den Regeln

$$P\colon S \to aSBc, abc \ (1,2) \quad cB \to Bc \ (3) \quad bB \to bb \ (4)$$

- Umgekehrt folgt durch Induktion über die Ableitungslänge m, dass jede Satzform $\alpha \in (V \cup \Sigma)^*$ mit $S \Rightarrow^m \alpha$ die folgenden Bedingungen erfüllt:
  - $\#_a(\alpha) = \#_b(\alpha) + \#_B(\alpha) = \#_c(\alpha)$
  - links von a und links von S kommen nur a's vor
  - links von b kommen nur a's oder b's vor
- Daraus ergibt sich, dass in G nur Wörter $w \in \Sigma^*$ der Form $w = a^n b^n c^n$ ableitbar sind, d.h. $L(G) = \{a^n b^n c^n \mid n \geq 1\} \in \mathrm{CSL}$ $\triangleleft$

Die Turingmaschine



- Um ein geeignetes Maschinenmodell für die kontextsensitiven Sprachen zu finden, führen wir zunächst das Rechenmodell der nichtdeterministischen Turingmaschine (NTM) ein
- Eine NTM erhält ihre Eingabe auf einem nach links und rechts unbegrenzten Band, das in Felder unterteilt ist; zudem kann sie weitere Bänder benutzen, die zu Beginn der Rechnung komplett leer sind
- In jedem Rechenschritt kann sie die aktuell besuchten Bandfelder lesen, die gelesenen Zeichen überschreiben und den Schreib-Lese-Kopf auf jedem Band um maximal ein Feld nach links oder rechts bewegen

Definition

- Sei $k \geq 1$. Eine nichtdeterministische k-Band-Turingmaschine (k-NTM oder einfach NTM) wird durch ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ beschrieben. Dabei ist
  - Z eine endliche Menge von Zuständen
  - Σ das Eingabealphabet (mit $\sqcup \notin \Sigma$; $\sqcup$ heißt Leerzeichen oder Blank)
  - Γ das Arbeitsalphabet (mit $\Sigma \cup \{\sqcup\} \subseteq \Gamma$)
  - δ: $Z \times \Gamma^k \to \mathcal{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ die Überführungsfunktion
  - q_0 der Startzustand und
  - $E \subseteq Z$ die Menge der Endzustände
- Eine k-NTM M heißt deterministisch (kurz: M ist eine k-DTM oder einfach DTM), falls für alle $(q, a_1, \ldots a_k) \in Z \times \Gamma^k$ gilt:

  $$\|\delta(q, a_1, \ldots a_k)\| \leq 1$$

Das Rechenmodell der Turingmaschine

- Für $(q, b_1, \ldots, b_k, D_1, \ldots, D_k) \in \delta(p, a_1, \ldots a_k)$ schreiben wir auch

$$(p, a_1, \ldots, a_k) \to (q, b_1, \ldots, b_k, D_1, \ldots, D_k)$$

- Eine solche Anweisung ist ausführbar, falls
  - p der aktuelle Zustand von M ist und
  - sich für $i = 1, \ldots, k$ der Kopf des i-ten Bandes auf einem mit a_i beschrifteten Feld befindet
- Bei ihrer Ausführung
  - geht M vom Zustand p in den Zustand q über
  - ersetzt auf Band $i = 1, \ldots, k$ das Symbol a_i durch b_i und
  - bewegt den Kopf auf Band $i = 1, \ldots, k$ gemäß D_i (L: ein Feld nach links, R: ein Feld nach rechts, N: keine Bewegung)

- Eine Konfiguration ist ein $(3k+1)$-Tupel

  $$K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \in Z \times (\Gamma^* \times \Gamma \times \Gamma^*)^k$$

  und besagt, dass

  - q der momentane Zustand ist und
  - das i-te Band mit $\ldots \sqcup u_i a_i v_i \sqcup \ldots$ beschriftet ist, wobei sich der Kopf auf dem Zeichen a_i befindet

- Im Fall $k = 1$ notieren wir eine Konfiguration $K = (q, u, a, v)$ auch in der Form $K = uqav$

Das Rechenmodell der Turingmaschine

- Seien $K = (p, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$ und $K' = (q, u'_1, a'_1, v'_1, \ldots, u'_k, a'_k, v'_k)$ Konfigurationen
- K' heißt Folgekonfiguration von K (kurz $K \vdash K'$), falls eine Anweisung $(p, a_1, \ldots, a_k) \to (q, b_1, \ldots, b_k, D_1, \ldots, D_k)$ existiert, so dass für $i = 1, \ldots, k$ gilt:

$D_i = \mathrm{N}$	$D_i = \mathrm{R}$	$D_i = \mathrm{L}$
K: $u_i \ \boxed{a_i} \ v_i$	K: $u_i \ \boxed{a_i} \ v_i$	K: $u_i \ \boxed{a_i} \ v_i$
K': $u_i \ \boxed{b_i} \ v_i$	K': $u_i \ b_i \ \boxed{a'_i} \ v'_i$	K': $u'_i \ \boxed{a'_i} \ b_i \ v_i$
$u'_i = u_i$ $a'_i = b_i$ $v'_i = v_i$	$u'_i = u_i b_i$ $a'_i v'_i = \begin{cases} v_i, & v_i \neq \varepsilon \\ \sqcup, & \text{sonst} \end{cases}$	$u'_i a'_i = \begin{cases} u_i, & u_i \neq \varepsilon \\ \sqcup, & \text{sonst} \end{cases}$ $v'_i = b_i v_i$

- Die Startkonfiguration von M bei Eingabe $x = x_1 \dots x_n \in \Sigma^*$ ist

$$K_x = \begin{cases} (q_0, \varepsilon, x_1, x_2 \dots x_n, \varepsilon, \sqcup, \varepsilon, \dots, \varepsilon, \sqcup, \varepsilon), & x \neq \varepsilon, \\ (q_0, \varepsilon, \sqcup, \varepsilon, \dots, \varepsilon \sqcup, \varepsilon), & x = \varepsilon \end{cases}$$

- Eine Rechnung von M bei Eingabe x ist eine (endliche oder unendliche) Folge von Konfigurationen $K_0, K_1, K_2 \dots$ mit $K_0 = K_x$ und $K_0 \vdash K_1 \vdash K_2 \dots$
- Die von M akzeptierte oder erkannte Sprache ist

$$L(M) = \{x \in \Sigma^* \mid \exists\, K \in E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k : K_x \vdash^* K\}$$

- Ein Wort x wird also genau dann von M akzeptiert (kurz: $M(x)$ akzeptiert), wenn es eine Rechnung von M bei Eingabe x gibt, bei der ein Endzustand erreicht wird

Beispiel

Betrachte die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit $Z = \{q_0, \ldots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{A, B, \sqcup\}$, $E = \{q_4\}$ und den Anweisungen

δ:	$q_0 a \to q_1 A R$	(1)	Beginn der Schleife: Falls ein a gelesen wird, ersetze es durch A und ...
	$q_1 a \to q_1 a R$	(2)	... lies a's und B's bis ein b kommt (falls kein b
	$q_1 B \to q_1 B R$	(3)	kommt, halte ohne zu akzeptieren), ersetze
	$q_1 b \to q_2 B L$	(4)	das b durch ein B und ...
	$q_2 a \to q_2 a L$	(5)	... bewege den Kopf wieder nach links bis
	$q_2 B \to q_2 B L$	(6)	auf das Feld hinter dem letzten A und
	$q_2 A \to q_0 A R$	(7)	gehe zum Beginn der Schleife
	$q_0 B \to q_3 B R$	(8)	Falls zu Beginn der Schleife ein B gelesen wird,
	$q_3 B \to q_3 B R$	(9)	teste, ob alle Eingabezeichen gelesen wurden
	$q_3 \sqcup \to q_4 \sqcup N$	(10)	(wenn ja, dann akzeptiere)

Beispiel (Fortsetzung)

Betrachte die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit $Z = \{q_0, \ldots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \Sigma \cup \{A, B, \sqcup\}$, $E = \{q_4\}$ und den Anweisungen

$$
\begin{array}{llllllll}
\delta\colon q_0 a \to q_1 AR & (1) & q_1 a \to q_1 aR & (2) & q_2 a \to q_2 aL & (5) & q_0 B \to q_3 BR & (8) \\
 & & q_1 B \to q_1 BR & (3) & q_2 B \to q_2 BL & (6) & q_3 B \to q_3 BR & (9) \\
 & & q_1 b \to q_2 BL & (4) & q_2 A \to q_0 AR & (7) & q_3 \sqcup \to q_4 \sqcup N & (10)
\end{array}
$$

- Dann akzeptiert M die Eingabe $aabb$ wie folgt:

$$
\begin{array}{lllll}
q_0 aabb & \underset{(1)}{\vdash} Aq_1 abb & \underset{(2)}{\vdash} Aaq_1 bb & \underset{(4)}{\vdash} Aq_2 aBb & \underset{(5)}{\vdash} q_2 AaBb \\
 & \underset{(7)}{\vdash} Aq_0 aBb & \underset{(1)}{\vdash} AAq_1 Bb & \underset{(3)}{\vdash} AABq_1 b & \underset{(4)}{\vdash} AAq_2 BB \\
 & \underset{(6)}{\vdash} Aq_2 ABB & \underset{(7)}{\vdash} AAq_0 BB & \underset{(8)}{\vdash} AABq_3 B & \underset{(9)}{\vdash} AABBq_3 \sqcup \\
 & \underset{(10)}{\vdash} AABBq_4 \sqcup & & &
\end{array}
$$

- Ähnlich lässt sich für ein beliebiges $n \geq 1$ zeigen, dass $a^n b^n \in L(M)$ ist

Beispiel (Schluss)

$$\delta\colon \begin{array}{llllllll} q_0 a \to q_1 AR & (1) & q_1 a \to q_1 aR & (2) & q_2 a \to q_2 aL & (5) & q_0 B \to q_3 BR & (8) \\ & & q_1 B \to q_1 BR & (3) & q_2 B \to q_2 BL & (6) & q_3 B \to q_3 BR & (9) \\ & & q_1 b \to q_2 BL & (4) & q_2 A \to q_0 AR & (7) & q_3 \sqcup \to q_4 \sqcup N & (10) \end{array}$$

- Andererseits führen die Eingaben *aba*, *abb* und *aab* auf die Rechnungen

$$q_0 aba \underset{(1)}{\vdash} Aq_1 ba \underset{(4)}{\vdash} q_2 ABa \underset{(7)}{\vdash} Aq_0 Ba \underset{(8)}{\vdash} ABq_3 a \text{ und}$$

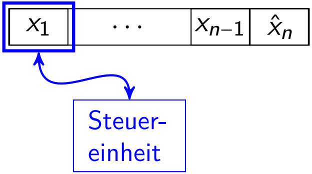
$$q_0 abb \underset{(1)}{\vdash} Aq_1 bb \underset{(4)}{\vdash} q_2 ABb \underset{(7)}{\vdash} Aq_0 Bb \underset{(8)}{\vdash} ABq_3 b \text{ und}$$

$$q_0 aab \underset{(1)}{\vdash} Aq_1 ab \underset{(2)}{\vdash} Aaq_1 b \underset{(4)}{\vdash} Aq_2 aB \underset{(5)}{\vdash} q_2 AaB$$
$$\underset{(7)}{\vdash} Aq_0 aB \underset{(1)}{\vdash} AAq_1 B \underset{(8)}{\vdash} AABq_1 \sqcup$$

- Da diese nicht fortsetzbar sind und M deterministisch ist, kann M nicht den Endzustand q_4 erreichen, d.h. $aba, abb, aab \notin L(M)$
- Tatsächlich lässt sich zeigen, dass $L(M) = \{a^n b^n \mid n \geq 1\}$ ist ◁

Ein Maschinenmodell für CSL

- In den Übungen werden wir eine 1-DTM M' für die Sprache $L' = \{a^n b^n c^n \mid n \geq 1\}$ konstruieren
- Wie M besucht auch M' außer den Eingabefeldern nur das erste Blank hinter der Eingabe
- Dies ist notwendig, damit M' das Ende der Eingabe erkennen kann
- Falls wir jedoch das letzte Zeichen der Eingabe x markieren, muss der Eingabebereich im Fall $|x| \geq 1$ für diesen Zweck nicht mehr verlassen werden:



- NTMs und DTMs mit dieser Eigenschaft werden auch als LBAs bzw. DLBAs bezeichnet

Linear beschränkte Automaten

Definition

- Für ein Alphabet Σ sei $\hat{\Sigma} = \Sigma \cup \{\hat{a} \mid a \in \Sigma\}$
- Für $x = x_1 \dots x_n \in \Sigma^*$ sei $\hat{x} = x_1 \dots x_{n-1}\hat{x}_n$
- Eine 1-NTM $M = (Z, \hat{\Sigma}, \Gamma, \delta, q_0, E)$ heißt LBA, falls gilt:

  $$\forall x \in \Sigma^+ : K_{\hat{x}} \vdash^* uqav \;\Rightarrow\; |uav| \le |x|$$

- Die von einem LBA M akzeptierte oder erkannte Sprache ist

  $$L(M) = \{x \in \Sigma^* \mid M(\hat{x}) \text{ akzeptiert}\}$$

- Ein deterministischer LBA wird auch als DLBA bezeichnet
- Die Klasse der deterministisch kontextsensitiven Sprachen ist DCSL $= \{L(M) \mid M \text{ ist ein DLBA}\}$

Bemerkung

Jede k-NTM, die bei Eingaben der Länge n höchstens linear viele (also $cn + c$ für eine Konstante c) Bandfelder benutzt, kann von einem LBA simuliert werden; LBA steht also für linear beschränkter Automat

Linear beschränkte Automaten

Beispiel

- Es ist nicht schwer, die 1-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ mit der Überführungsfunktion

  $$\delta\colon\ q_0 a \to q_1 AR \ (1) \quad \begin{array}{llllll} q_1 a \to q_1 aR & (2) & q_2 a \to q_2 aL & (5) & q_0 B \to q_3 BR & (8) \\ q_1 B \to q_1 BR & (3) & q_2 B \to q_2 BL & (6) & q_3 B \to q_3 BR & (9) \\ q_1 b \to q_2 BL & (4) & q_2 A \to q_0 AR & (7) & q_3 \sqcup \to q_4 \sqcup N & (10) \end{array}$$

  in einen DLBA $M' = (Z, \hat{\Sigma}, \Gamma', \delta', q_0, E)$ für die Sprache $\{a^n b^n \mid n \geq 1\}$ umzuwandeln
- Ersetze hierzu
  - $\Sigma = \{a, b\}$ durch $\hat{\Sigma} = \{a, b, \hat{a}, \hat{b}\}$ und
  - $\Gamma = \Sigma \cup \{A, B, \sqcup\}$ durch $\Gamma' = \hat{\Sigma} \cup \{A, B, \hat{B}, \sqcup\}$
- Füge zudem
  - die Anweisungen $q_1 \hat{b} \to q_2 \hat{B} L$ $(4a)$ und $q_0 \hat{B} \to q_4 \hat{B} N$ $(8a)$ hinzu und
  - ersetze die Anweisung $q_3 \sqcup \to q_4 \sqcup N$ (10) durch $q_3 \hat{B} \to q_4 \hat{B} N$ $(10')$

Beispiel (Fortsetzung)

- Damit erhalten wir folgende Überführungsfunktion für den DLBA M':

$$
\begin{array}{lllllllll}
\delta': & q_0 a \to q_1 AR & (1) & q_1 \hat{b} \to q_2 \hat{B} L & (4a) & q_0 B \to q_3 BR & (8) \\
& q_1 a \to q_1 aR & (2) & q_2 a \to q_2 aL & (5) & q_0 \hat{B} \to q_4 \hat{B} N & (8a) \\
& q_1 B \to q_1 BR & (3) & q_2 B \to q_2 BL & (6) & q_3 B \to q_3 BR & (9) \\
& q_1 b \to q_2 BL & (4) & q_2 A \to q_0 AR & (7) & q_3 \hat{B} \to q_4 \hat{B} N & (10')
\end{array}
$$

- Dieser akzeptiert die beiden Eingaben $a\hat{b}$ und $aab\hat{b}$ wie folgt:

$$
q_0 a\hat{b} \underset{(1)}{\vdash} Aq_1\hat{b} \underset{(4a)}{\vdash} q_2 A\hat{B} \underset{(7)}{\vdash} Aq_0\hat{B} \underset{(8a)}{\vdash} Aq_4\hat{B}
$$

$$
\begin{array}{lllllllll}
q_0 aab\hat{b} & \underset{(1)}{\vdash} & Aq_1 ab\hat{b} & \underset{(2)}{\vdash} & Aaq_1 b\hat{b} & \underset{(4)}{\vdash} & Aq_2 aB\hat{b} & \underset{(5)}{\vdash} & q_2 AaB\hat{b} \\
& \underset{(7)}{\vdash} & Aq_0 aB\hat{b} & \underset{(1)}{\vdash} & AAq_1 B\hat{b} & \underset{(3)}{\vdash} & AABq_1\hat{b} & \underset{(4a)}{\vdash} & AAq_2 B\hat{B} \\
& \underset{(6)}{\vdash} & Aq_2 AB\hat{B} & \underset{(7)}{\vdash} & AAq_0 B\hat{B} & \underset{(8)}{\vdash} & AABq_3\hat{B} & \underset{(10')}{\vdash} & AABq_4\hat{B}
\end{array}
$$

◁

Das LBA-Problem

Bemerkung

- Der DLBA M' für die Sprache $\{a^n b^n \mid n \geq 1\}$ aus obigem Beispiel lässt sich leicht in einen DLBA für die kontextsensitive Sprache $\{a^n b^n c^n \mid n \geq 1\}$ transformieren (siehe Übungen)
- Die Sprache $\{a^n b^n c^n \mid n \geq 1\}$ liegt also in DCSL $\setminus$ CFL
- Bis heute ungelöst ist die Frage, ob die Klasse DCSL eine echte Teilklasse von CSL ist oder nicht
- Diese Fragestellung ist als LBA-Problem bekannt

Charakterisierung von CSL mittels LBAs

Als nächstes zeigen wir, dass LBAs genau die kontextsensitiven Sprachen erkennen

Satz

CSL = $\{L(M) \mid M$ ist ein LBA$\}$

Beweis von CSL $\subseteq \{L(M) \mid M \text{ ist ein LBA}\}$

Sei $G = (V, \Sigma, P, S)$ eine kontextsensitive Grammatik. Dann wird $L(G)$ von folgendem LBA M akzeptiert (o.B.d.A. sei $\varepsilon \notin L(G)$):

Arbeitsweise von M bei Eingabe $\hat{x} = x_1 \dots x_{n-1}\hat{x}_n$ mit $n > 0$:

1 Markiere das erste Eingabezeichen x_1 mittels $\tilde{x}_1$ (bzw. $\hat{x}_1$ mittels $\tilde{\hat{x}}_1$)
2 Wähle (nichtdeterministisch) eine Regel $\alpha \to \beta$ aus P
3 Wähle ein beliebiges Vorkommen von β auf dem Band
  (falls β nicht vorkommt, halte ohne zu akzeptieren)
4 Ersetze die ersten $|\alpha|$ Zeichen von β durch α
5 Falls das erste (oder letzte) Zeichen von β markiert war,
  markiere auch das erste (letzte) Zeichen von α
6 Verschiebe die Zeichen rechts von β um $|\beta| - |\alpha|$ Positionen nach
  links und überschreibe die frei werdenden Felder mit Blanks
7 Falls auf dem Band das (doppelt markierte) Startsymbol erscheint,
  halte in einem Endzustand
8 Gehe zurück zu Schritt 2

Beweis von CSL $\subseteq \{L(M) \mid M$ ist ein LBA$\}$

- Da M sukzessive ein Teilwort β des aktuellen Bandinhalts durch ein Wort α mit $|\alpha| \leq |\beta|$ ersetzt, ist M tatsächlich ein LBA
- Zudem akzeptiert M eine Eingabe x genau dann, falls es gelingt, eine Ableitung $S \Rightarrow^* x$ in G zu finden (in umgekehrter Reihenfolge von rechts nach links)
- Da sich genau für die Wörter $x \in L(G)$ eine solche Ableitung finden lässt, folgt $L(M) = L(G)$ $\square$

Beweis von $\{L(M) \mid M \text{ ist ein LBA}\} \subseteq \mathrm{CSL}$

- Sei $M = (Z, \hat{\Sigma}, \Gamma, \delta, q_0, E)$ ein LBA (o.B.d.A. sei $\varepsilon \notin L(M)$)
- Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit

$$V = \{S, A\} \cup (Z\Gamma \cup \Gamma) \times \Sigma = \{S, A, (qd, a), (d, a) \mid q \in Z, d \in \Gamma, a \in \Sigma\},$$

die für alle $a, b \in \Sigma$ und $c, c', d \in \Gamma$ folgende Regeln enthält:

P:	$S \to A(\hat{a}, a), (q_0\hat{a}, a)$		(S)	„Startregeln“
	$A \to A(a, a), (q_0 a, a)$		(A)	„A-Regeln“
	$(c, a) \to a$		(F)	„Finale Regeln“
	$(qc, a) \to a,$	falls $q \in E$	(E)	„E-Regeln“
	$(qc, a) \to (q'c', a),$	falls $qc \to_M q'c'N$	(N)	„N-Regeln“
	$(qc, a)(d, b) \to (c', a)(q'd, b),$	falls $qc \to_M q'c'R$	(R)	„R-Regeln“
	$(d, a)(qc, b) \to (q'd, a)(c', b),$	falls $qc \to_M q'c'L$	(L)	„L-Regeln“

Beweis von $\{L(M) \mid M \text{ ist ein LBA}\} \subseteq \mathrm{CSL}$

Beispiel

- Betrachte den LBA $M = (Z, \hat{\Sigma}, \Gamma, \delta, q_0, E)$ mit $Z = \{q_0, \dots q_4\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \hat{a}, \hat{b}, A, B, \hat{B}, \sqcup\}$ und $E = \{q_4\}$, sowie

$$\delta\colon \begin{array}{llll} q_0 a \to q_1 AR & q_1 b \to q_2 BL & q_2 A \to q_0 AR & q_0 \hat{B} \to q_4 \hat{B} N \\ q_1 a \to q_1 aR & q_1 \hat{b} \to q_2 \hat{B} L & q_2 B \to q_2 BL & q_3 B \to q_3 BR \\ q_1 B \to q_1 BR & q_2 a \to q_2 aL & q_0 B \to q_3 BR & q_3 \hat{B} \to q_4 \hat{B} N \end{array}$$

- Die zugehörige kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ hat dann die Variablenmenge

$$\begin{aligned} V &= \{S, A\} \cup (Z\Gamma \cup \Gamma) \times \Sigma \\ &= \{S, A, (q_i c, a), (q_i c, b), (c, a), (c, b) \mid 0 \le i \le 4, c \in \Gamma\} \end{aligned}$$

- Die Regelmenge P von G enthält folgende Start- und A-Regeln:

$$S \to A(\hat{a}, a),\ A(\hat{b}, b),\ (q_0 \hat{a}, a),\ (q_0 \hat{b}, b) \qquad (S_1\text{-}S_4)$$
$$A \to A(a, a),\ A(b, b),\ (q_0 a, a),\ (q_0 b, b) \qquad (A_1\text{-}A_4)$$

Beweis von $\{L(M) \mid M \text{ ist ein LBA}\} \subseteq \mathrm{CSL}$

Beispiel (Fortsetzung)

- Zudem enthält P wegen $E = \{q_4\}$ für jedes $c \in \Gamma$ die F- und E-Regeln

$$(c, a) \to a,\ (c, b) \to b \qquad (F_1\text{-}F_{16})$$
$$(q_4c, a) \to a,\ (q_4c, b) \to b \qquad (E_1\text{-}E_{16})$$

- Schließlich enthält P noch 4 N-, 128 L- und 192 R-Regeln wie z.B.
  - für die Anweisung $q_3\hat{B} \to q_4\hat{B}N$ die beiden folgenden N-Regeln:

$$(q_3\hat{B}, a) \to (q_4\hat{B}, a) \text{ und } (q_3\hat{B}, b) \to (q_4\hat{B}, b)$$

  - für $q_1b \to q_2BL$ insgesamt 32 L-Regeln, nämlich für jedes $d \in \Gamma$:

$$(d, a)(q_1b, a) \to (q_2d, a)(B, a) \quad (d, a)(q_1b, b) \to (q_2d, a)(B, b)$$
$$(d, b)(q_1b, a) \to (q_2d, b)(B, a) \quad (d, b)(q_1b, b) \to (q_2d, b)(B, b)$$

  - für $q_0a \to q_1AR$ insgesamt 32 R-Regeln, nämlich für jedes $d \in \Gamma$:

$$(q_0a, a)(d, a) \to (A, a)(q_1d, a) \quad (q_0a, a)(d, b) \to (A, a)(q_1d, b)$$
$$(q_0a, b)(d, a) \to (A, b)(q_1d, a) \quad (q_0a, b)(d, b) \to (A, b)(q_1d, b) \quad \triangleleft$$

Beweis von $\{L(M) \mid M \text{ ist ein LBA}\} \subseteq \mathrm{CSL}$

- Sei $M = (Z, \hat{\Sigma}, \Gamma, \delta, q_0, E)$ ein LBA (o.B.d.A. sei $\varepsilon \notin L(M)$)
- Betrachte die kontextsensitive Grammatik $G = (V, \Sigma, P, S)$ mit

$$V = \{S, A\} \cup (Z\Gamma \cup \Gamma) \times \Sigma = \{S, A, (qd, a), (d, a) \mid q \in Z, d \in \Gamma, a \in \Sigma\},$$

die für alle $a, b \in \Sigma$ und $c, c', d \in \Gamma$ folgende Regeln enthält:

P:	$S \to A(\hat{a}, a), (q_0\hat{a}, a)$		(S)	„Startregeln“
	$A \to A(a, a), (q_0 a, a)$		(A)	„A-Regeln“
	$(c, a) \to a$		(F)	„Finale Regeln“
	$(qc, a) \to a,$	falls $q \in E$	(E)	„E-Regeln“
	$(qc, a) \to (q'c', a),$	falls $qc \to_M q'c'N$	(N)	„N-Regeln“
	$(qc, a)(d, b) \to (c', a)(q'd, b),$	falls $qc \to_M q'c'R$	(R)	„R-Regeln“
	$(d, a)(qc, b) \to (q'd, a)(c', b),$	falls $qc \to_M q'c'L$	(L)	„L-Regeln“

Beweis von $\{L(M) \mid M \text{ ist ein LBA}\} \subseteq \mathrm{CSL}$

- Durch Induktion über m lässt sich nun leicht für alle $a_1, \ldots, a_n \in \Gamma$ und $q \in Z$ die folgende Äquivalenz beweisen:

$$q_0 x_1 \ldots x_{n-1} \hat{x}_n \vdash^m a_1 \ldots a_{i-1} q a_i \ldots a_n \text{ gdw.}$$

$$(q_0 x_1, x_1) \ldots (\hat{x}_n, x_n) \underset{(N,R,L)}{\Rightarrow}^m (a_1, x_1) \ldots (q a_i, x_i) \ldots (a_n, x_n)$$

- Ist also $q_0 x_1 \ldots x_{n-1} \hat{x}_n \vdash^m a_1 \ldots a_{i-1} q a_i \ldots a_n$ eine akzeptierende Rechnung von $M(x_1 \ldots x_{n-1} \hat{x}_n)$ mit $q \in E$, so folgt

$$S \underset{(S)}{\Rightarrow} A(\hat{x}_n, x_n) \underset{(A)}{\Rightarrow}^{n-1} (q_0 x_1, x_1)(x_2, x_2) \ldots (x_{n-1}, x_{n-1})(\hat{x}_n, x_n)$$

$$\underset{(N,L,R)}{\Rightarrow}^m (a_1, x_1) \ldots (a_{i-1}, x_{i-1})(q a_i, x_i) \ldots (a_n, x_n) \underset{(F,E)}{\Rightarrow}^n x_1 \ldots x_n$$

- Die Inklusion $L(G) \subseteq L(M)$ folgt analog $\square$

Bemerkung

Eine einfache Modifikation des Beweises zeigt, dass 1-NTMs genau die Sprachen vom Typ 0 akzeptieren (siehe Übungen)

Zusammenfassung der Abschlusseigenschaften

	Vereinigung	Schnitt	Komplement	Produkt	Sternhülle
REG	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	ja	nein	nein	ja	ja
DCSL	ja	ja	ja	ja	ja
CSL	ja	ja	ja	ja	ja
RE	ja	ja	nein	ja	ja

- In der VL Komplexitätstheorie wird gezeigt, dass die Klasse CSL unter Komplementbildung abgeschlossen ist
- Im nächsten Kapitel werden wir sehen, dass die Klasse RE nicht unter Komplementbildung abgeschlossen ist
- Die übrigen Abschlusseigenschaften der Klassen DCSL, CSL und RE in obiger Tabelle werden in den Übungen bewiesen

Entscheidbare und semi-entscheidbare Sprachen

Definition

- Eine NTM M hält bei Eingabe x (kurz: $M(x) = \downarrow$ oder $M(x)\downarrow$), falls alle Rechnungen von $M(x)$ nach endlich vielen Schritten halten
- Falls $M(x)$ nicht hält, schreiben wir auch kurz $M(x) = \uparrow$ oder $M(x)\uparrow$
- Eine DTM M entscheidet eine Eingabe x, falls $M(x)$ hält oder eine Konfiguration mit einem Endzustand erreicht
- Eine Sprache heißt entscheidbar, falls sie von einer DTM M akzeptiert wird, die alle Eingaben entscheidet. Die zugehörige Sprachklasse ist

$$\mathsf{REC} = \{L(M) \mid M \text{ ist eine DTM, die alle Eingaben entscheidet}\}$$

- Jede von einer DTM akzeptierte Sprache heißt semi-entscheidbar

Bemerkung

- Eine DTM M entscheidet zwar immer alle Eingaben $x \in L(M)$, aber eventuell nicht alle $x \in \overline{L(M)}$. Daher heißt $L(M)$ semi-entscheidbar
- Später werden wir sehen, dass $\mathsf{RE} = \{L(M) \mid M \text{ ist eine DTM}\}$ ist

Berechnung von Funktionen

Definition

- Eine k-DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ berechnet eine Funktion $f : \Sigma^* \to \Gamma^*$, falls M bei jeder Eingabe $x \in \Sigma^*$ in einer Konfiguration
$$K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \text{ mit } u_k = f(x)$$
hält (d.h. $K_x \vdash^* K$ und K hat keine Folgekonfiguration)
- Hierfür sagen wir auch, M gibt bei Eingabe x das Wort $f(x)$ aus und schreiben $M(x) = f(x)$
- f heißt Turing-berechenbar (oder einfach berechenbar), falls es eine k-DTM M mit $M(x) = f(x)$ für alle $x \in \Sigma^*$ gibt
- Aus historischen Gründen werden berechenbare Funktionen auch rekursiv (engl. *recursive*) genannt

Berechenbarkeit von charakteristischen Funktionen

Definition

Für eine Sprache $A \subseteq \Sigma^*$ ist die charakteristische Funktion $\chi_A : \Sigma^* \to \{0,1\}$ wie folgt definiert:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases}$$

Bemerkung

- In den Übungen wird gezeigt, dass eine Sprache A genau dann entscheidbar ist, wenn χ_A berechenbar (also rekursiv) ist
- Dies erklärt die Bezeichnung REC für die Klasse der entscheidbaren Sprachen
- Dort wird auch gezeigt, dass CSL echt in REC enthalten ist
- Beispiele für interessante semi-entscheidbare Sprachen, die nicht entscheidbar sind, werden wir noch kennenlernen
- Somit gilt REG $\subsetneq$ DCFL $\subsetneq$ CFL $\subsetneq$ DCSL $\subseteq$ CSL $\subsetneq$ REC $\subsetneq$ RE

Berechenbarkeit von partiellen Funktionen

Definition

- Eine partielle Funktion hat die Form $f : A \to B \cup \{\uparrow\}$, wobei $\uparrow \notin B$ ist
- Für $f(x) = \uparrow$ sagen wir auch $f(x)$ ist undefiniert
- Der Definitionsbereich (engl. *domain*) von f ist

  $$dom(f) = \{x \in A \mid f(x) \neq \uparrow\}$$

- Das Bild (engl. *image*) von f ist

  $$img(f) = \{f(x) \mid x \in dom(f)\}$$

- Eine partielle Funktion $f : A \to B \cup \{\uparrow\}$ heißt total, falls $dom(f) = A$ ist
- Eine partielle Funktion $f : \Sigma^* \to \Gamma^* \cup \{\uparrow\}$ heißt berechenbar, falls es eine DTM M mit $M(x) = f(x)$ für alle $x \in \Sigma^*$ gibt (d.h. $M(x)$ gibt für alle $x \in dom(f)$ das Wort $f(x)$ aus und hält im Fall $x \notin dom(f)$ nicht)

- Jede DTM $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ ber. eine part. Fkt. $f : \Sigma^* \to \Gamma^* \cup \{\uparrow\}$
- Die Menge $dom(f) = \{x \in \Sigma^* \mid M(x)\downarrow\}$ bezeichnen wir mit $dom(M)$

Wir fassen die berechenbaren Funktionen und berechenbaren partiellen Funktionen in folgenden Klassen zusammen:

$$\mathrm{FREC} = \{f \mid f \text{ ist eine berechenbare (totale) Funktion}\}$$

$$\mathrm{FREC}_p = \{f \mid f \text{ ist eine berechenbare partielle Funktion}\}$$

Dann gilt $\mathrm{FREC} \subsetneq \mathrm{FREC}_p$

Beispiel

- Bezeichne x^+ den lexikografischen Nachfolger von $x \in \Sigma^*$
- Für $\Sigma = \{0, 1\}$ ergeben sich beispielsweise folgende Werte:

x	ε	0	1	00	01	10	11	000	$\cdots$
x^+	0	1	00	01	10	11	000	001	$\cdots$

- Betrachte die auf Σ^* definierten partiellen Funktionen f_1, f_2, f_3, f_4 mit

$$f_1(x) = 0 \qquad f_2(x) = x \qquad f_3(x) = x^+ \qquad \text{und} \qquad f_4(x) = \begin{cases} \uparrow, & x = \varepsilon \\ y, & x = y^+ \end{cases}$$

- Da f_1, f_2, f_3, f_4 berechenbar sind, gehören die totalen Funktionen f_1, f_2, f_3 zu FREC und die partielle Funktion f_4 zu FREC_p
- Da f_4 keine totale Funktion ist, gehört f_4 nicht zu FREC ◁

Partielle charakteristische Funktionen

Definition

Sei $A \subseteq \Sigma^*$ eine Sprache

- Die partielle charakteristische Funktion von A ist $\hat{\chi}_A : \Sigma^* \to \{1\} \cup \{\uparrow\}$ mit

$$\hat{\chi}_A(x) = \begin{cases} 1, & x \in A \\ \uparrow, & x \notin A \end{cases}$$

- A heißt rekursiv aufzählbar, falls $A = \emptyset$ oder das Bild $img(f)$ einer (totalen) berechenbaren Funktion $f : \Gamma^* \to \Sigma^*$ ist

Charakterisierung der rekursiv aufzählbaren Sprachen

Satz

Folgende Eigenschaften sind für eine Sprache $A \subseteq \Sigma^*$ äquivalent:

1. A ist semi-entscheidbar (d.h. A wird von einer DTM akzeptiert)
2. A wird von einer 1-DTM akzeptiert
3. A ist vom Typ 0
4. A wird von einer NTM akzeptiert
5. A ist rek. aufzählbar (d.h. $A = \varnothing$ oder $A = img(f)$ für eine Fkt. $f \in \mathrm{FREC}$)
6. $\hat{\chi}_A$ ist berechenbar (d.h. $\hat{\chi}_A \in \mathrm{FREC}_p$)
7. es gibt eine DTM M mit $A = dom(M)$

Beweis

Die Implikationen 2 $\Rightarrow$ 3 $\Rightarrow$ 4 werden in den Übungen gezeigt.
Hier zeigen wir 1 $\Rightarrow$ 2 und 4 $\Rightarrow$ 5 $\Rightarrow$ 6 $\Rightarrow$ 7 $\Rightarrow$ 1

Simulation einer k-DTM durch eine 1-DTM

Beweis von ❶ $\Rightarrow$ ❷: $\{L(M) \mid M \text{ ist eine DTM}\} \subseteq \{L(M) \mid M \text{ ist eine 1-DTM}\}$

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-DTM mit $L(M) = A$
- Wir konstruieren eine 1-DTM $M' = (Z', \Sigma, \Gamma', \delta', z_0, E)$ für A
- M' simuliert M, indem sie jede Konfiguration K von M der Form

$$
\begin{array}{c|c|c|c|c|c}
\hline
\cdots & a & b & c & d & \cdots \\
\hline
\multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\vdots} & \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} \\
\hline
\cdots & e & f & g & h & \cdots \\
\hline
\multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{}
\end{array}
$$

durch eine Konfiguration K' folgender Form nachbildet:

$$
\begin{array}{c|c|c|c|c|c}
\hline
\cdots & \begin{pmatrix} a \\ \vdots \\ \hat{e} \end{pmatrix} & \begin{pmatrix} b \\ \vdots \\ f \end{pmatrix} & \begin{pmatrix} \hat{c} \\ \vdots \\ g \end{pmatrix} & \begin{pmatrix} d \\ \vdots \\ h \end{pmatrix} & \cdots \\
\hline
\end{array}
$$

Simulation einer k-DTM durch eine 1-DTM

Beweis von ❶ ⇒ ❷: $\{L(M) \mid M \text{ ist eine DTM}\} \subseteq \{L(M) \mid M \text{ ist eine 1-DTM}\}$

- Hierzu arbeitet M' mit dem Alphabet

$$\Gamma' = \Gamma \cup (\Gamma \cup \hat{\Gamma})^k, \text{ wobei } \hat{\Gamma} = \{\hat{a} \mid a \in \Gamma\} \text{ ist}$$

- Bei Eingabe $x = x_1 \dots x_n$ erzeugt M' zuerst die der Startkonfiguration

$$K_x = (q_0, \varepsilon, x_1, x_2 \dots x_n, \varepsilon, \sqcup, \varepsilon, \dots, \varepsilon, \sqcup, \varepsilon)$$

von M bei Eingabe x entsprechende Konfiguration

$$K'_x = q'_0 \begin{pmatrix} \hat{x}_1 \\ \hat{\sqcup} \\ \vdots \\ \hat{\sqcup} \end{pmatrix} \begin{pmatrix} x_2 \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix} \cdots \begin{pmatrix} x_n \\ \sqcup \\ \vdots \\ \sqcup \end{pmatrix}$$

Simulation einer k-DTM durch eine 1-DTM

Beweis von ❶ ⇒ ❷: $\{L(M) \mid M \text{ ist eine DTM}\} \subseteq \{L(M) \mid M \text{ ist eine 1-DTM}\}$

- Dann simuliert M' jeweils einen Schritt von M durch folgende Sequenz von Rechenschritten:
  - Zuerst geht M' solange nach rechts, bis sie alle mit ˆ markierten Zeichen (z.B. $\hat{a}_1, \ldots, \hat{a}_k$) gefunden hat
  - Diese Zeichen speichert M' in ihrem Zustand
  - Anschließend geht M' wieder nach links und realisiert dabei die durch $\delta(q, a_1, \ldots, a_k)$ vorgegebene Anweisung von M
  - Dabei speichert M' den aktuellen Zustand q von M ebenfalls in ihrem Zustand
- Sobald M in einen Endzustand übergeht, wechselt M' ebenfalls in einen Endzustand und hält
- Somit gilt $L(M') = L(M)$ □

Beweis von ④ ⇒ ⑤: $\{L(M) \mid M \text{ ist eine NTM}\} \subseteq \{A \mid A \text{ ist rek. aufzählbar}\}$

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine k-NTM und sei $A = L(M) \neq \varnothing$
- Sei $\tilde{\Gamma}$ das Alphabet $Z \cup \Gamma \cup \{\#\}$
- Wir kodieren eine Konfiguration $K = (q, u_1, a_1, v_1, \dots, u_k, a_k, v_k)$ durch das Wort

  $$code(K) = \#q\#u_1\#a_1\#v_1\# \dots \#u_k\#a_k\#v_k\#$$

  und eine Rechnung $K_0 \vdash \dots \vdash K_t$ durch $code(K_0) \dots code(K_t)$
- Dann lassen sich die Wörter von A durch folgende Funktion $f : \tilde{\Gamma}^* \to \Sigma^*$ aufzählen (dabei ist x_0 ein beliebiges Wort in A):

  $$f(w) = \begin{cases} x, & w \text{ kodiert eine akz. Rechnung } K_0 \vdash \dots \vdash K_t \text{ von } \\ & M(x), \text{ d.h. } K_0 = K_x \text{ und } K_t \in E \times (\Gamma^* \times \Gamma \times \Gamma^*)^k \\ x_0, & \text{sonst} \end{cases}$$

- Da f berechenbar ist, ist $A = img(f)$ rekursiv aufzählbar □

Charakterisierung der rekursiv aufzählbaren Sprachen

Beweis von ❺ ⇒ ❻: $\{A \mid A \text{ ist rek. aufzählbar}\} \subseteq \{A \mid \hat{\chi}_A \in \mathrm{FREC}_p\}$

- Sei M eine DTM, die eine Fkt. $f : \Gamma^* \to \Sigma^*$ mit $A = img(f)$ berechnet
- Dann wird $\hat{\chi}_A$ von der DTM M' berechnet, die bei Eingabe x
  - der Reihe nach für alle $w \in \Gamma^*$ das Wort $f(w)$ berechnet und
  - den Wert 1 ausgibt, sobald $f(w) = x$ ist □

Beweis von ❻ ⇒ ❼: $\{A \mid \hat{\chi}_A \in \mathrm{FREC}_p\} \subseteq \{dom(M) \mid M \text{ ist eine DTM}\}$

- Sei M eine DTM, die $\hat{\chi}_A$ berechnet
- Da $dom(\hat{\chi}_A) = A$ ist, folgt $A = dom(M)$ □

Beweis von ❼ ⇒ ❶: $\{dom(M) \mid M \text{ ist eine DTM}\} \subseteq \{L(M) \mid M \text{ ist eine DTM}\}$

- Sei $A = dom(M)$ für eine DTM M
- Dann gilt $A = L(M')$ für die DTM M', die M simuliert und genau dann in einen Endzustand übergeht, wenn M hält □

Charakterisierung der entscheidbaren Sprachen

Satz

Folgende Eigenschaften sind äquivalent:

1. A ist entscheidbar (d.h. A wird von einer DTM akzeptiert, die alle Eingaben entscheidet)
2. die charakteristische Funktion χ_A von A ist berechenbar
3. A wird von einer 1-DTM akzeptiert, die bei allen Eingaben hält
4. A wird von einer NTM akzeptiert, die bei allen Eingaben hält
5. A und $\bar{A}$ sind semi-entscheidbar

Beweis

Die Äquivalenz der Bedingungen 1 bis 4 wird in den Übungen gezeigt.
Hier zeigen wir nur die Äquivalenz dieser vier Bedingungen zu 5

Charakterisierung der entscheidbaren Sprachen

Beweis von ❶ ⇒ ❺: $\mathrm{REC} \subseteq \mathrm{RE} \cap \mathrm{co}\text{-}\mathrm{RE}$

- Falls A entscheidbar ist, ist mit χ_A auch $\chi_{\bar{A}}$ berechenbar, d.h. A und $\bar{A}$ sind entscheidbar und damit auch semi-entscheidbar

Beweis von ❺ ⇒ ❶: $\mathrm{RE} \cap \mathrm{co}\text{-}\mathrm{RE} \subseteq \mathrm{REC}$

- Seien M_A und $M_{\bar{A}}$ DTMs, die $\hat{\chi}_A$ und $\hat{\chi}_{\bar{A}}$ berechnen
- Betrachte die DTM M, die abwechselnd M_A und $M_{\bar{A}}$ für jeweils einen weiteren Rechenschritt simuliert und
  - in einem Endzustand hält, sobald $M_A(x)$ hält, sowie
  - in einem Nichtendzustand hält, sobald $M_{\bar{A}}(x)$ hält
- Da jede Eingabe x entweder in $dom(\hat{\chi}_A) = A$ oder in $dom(\hat{\chi}_{\bar{A}}) = \bar{A}$ enthalten ist, hält M bei allen Eingaben
- Da zudem $L(M) = A$ ist, folgt $A \in \mathrm{REC}$ □

Kodierung (Gödelisierung) von Turingmaschinen

- Um Eigenschaften von TMs algorithmisch untersuchen zu können, müssen wir TMs als Teil der Eingabe kodieren
- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine 1-DTM mit
  - Zustandsmenge $Z = \{q_0, \ldots, q_m\}$ (o.B.d.A. sei $E = \{q_m\}$),
  - Eingabealphabet $\Sigma = \{0, 1\}$ und
  - Arbeitsalphabet $\Gamma = \{a_0, \ldots, a_l\}$,
    wobei wir o.B.d.A. $a_0 = 0$, $a_1 = 1$ und $a_2 = \sqcup$ annehmen
- Dann kodieren wir eine Anweisung $q_i a_j \to q_{i'} a_{j'} D$ durch das Wort

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#b_D\#$$

- Dabei ist $bin(n)$ die Binärdarstellung von n und

$$b_D = \begin{cases} 0, & D = N \\ 1 & D = L \\ 10, & D = R \end{cases}$$

Kodierung von Turingmaschinen

- M lässt sich nun als ein Wort über dem Alphabet $\{0, 1, \#\}$ kodieren, indem wir die Anweisungen von M in kodierter Form auflisten
- Kodieren wir die Zeichen $0, 1, \#$ binär (z.B. $0 \mapsto 00$, $1 \mapsto 01$, $\# \mapsto 10$), so gelangen wir zu einer Binärkodierung w_M von M
- Die durch die Binärzahl $w_M = b_n \dots b_0$ repräsentierte natürliche Zahl $(w_M)_2 = \sum_{i=0}^{n} b_i 2^i$ wird auch die Gödel-Nummer von M genannt
- M_w ist durch Angabe von w_M bzw. $(w_M)_2$ bis auf die Benennung ihrer Zustände und der Arbeitszeichen in $\Gamma \setminus \{\sqcup, 0, 1\}$ eindeutig bestimmt
- Ganz analog lassen sich auch k-DTMs mit $k > 1$ sowie NTMs binär kodieren
- Umgekehrt können wir jedem Binärstring $w \in \{0, 1\}^*$ eine DTM M_w wie folgt zuordnen (dabei ist M_0 eine beliebige, aber fest gewählte DTM):

$$M_w = \begin{cases} M, & \text{falls eine DTM } M \text{ mit } w_M = w \text{ existiert} \\ M_0, & \text{sonst} \end{cases}$$

Unentscheidbarkeit des Halteproblems

Definition

Das Halteproblem ist die Sprache

$$H = \left\{ w\#x \;\middle|\; \begin{array}{l} w, x \in \{0,1\}^* \text{ und} \\ \text{die DTM } M_w \text{ hält} \\ \text{bei Eingabe } x \end{array} \right\}$$

und das spezielle Halteproblem ist

$$K = \left\{ w \in \{0,1\}^* \;\middle|\; \begin{array}{l} \text{die DTM } M_w \\ \text{hält bei Eingabe } w \end{array} \right\}$$

χ_H	w_1	w_2	w_3	$\cdots$
w_1	0	1	0	$\cdots$
w_2	0	1	1	$\cdots$
w_3	1	1	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

χ_K				
w_1	0			
w_2		1		
w_3			0	
$\vdots$				$\ddots$

Satz

$K, H \in \mathrm{RE}$

Beweis von $K, H \in \mathrm{RE}$

- Sei w_h die Kodierung einer DTM, die sofort hält, und betrachte die Funktionen $f_K : \{0,1\}^* \to \{0,1\}^*$ und $g_H : \{0,1,\#\}^* \to \{0,1,\#\}^*$ mit

$$f_K(x) = \begin{cases} w, & x \text{ ist die Binärkodierung einer haltenden Rechnung einer DTM } M_w \text{ bei Eingabe } w, \\ w_h, & \text{sonst} \end{cases}$$

  und

$$g_H(x) = \begin{cases} w\#w', & x = w\#x' \text{ und } x' \text{ ist die Binärkodierung einer haltenden Rechnung der DTM } M_w \text{ bei Eingabe } w', \\ w_h\#\varepsilon, & \text{sonst} \end{cases}$$

- Da f_K und g_H in FREC sind und $img(f_K) = K$ sowie $img(g_H) = H$ ist, folgt $K, H \in \mathrm{RE}$ □

Unentscheidbarkeit des speziellen Halteproblems

Satz

$K \notin \mathsf{RE}$ und somit $\mathsf{REC} \subsetneq \mathsf{RE} \neq \text{co-}\mathsf{RE}$

Beweisidee

- Sei $B = (b_{ij})$ die durch $b_{ij} = \chi_H(w_i\#w_j) \in \{0,1\}$ definierte Binärmatrix
- Dann kann keine Zeile $b_{i1}b_{i2}\ldots$ von B mit der invertierten Diagonalen $\bar{b}_{11}\bar{b}_{22}\ldots$ von B übereinstimmen, da sonst $b_{ii} = \bar{b}_{ii}$ sein müsste
- Da die i-te Zeile von B wegen

$$b_{ij} = \chi_H(w_i\#w_j) = \chi_{dom(M_{w_i})}(w_j)$$

  die Binärsprache $dom(M_{w_i}) \in \mathsf{RE}$ und die invertierte Diagonale wegen

$$\bar{b}_{ii} = \chi_{\bar{H}(w_i\#w_i)} = \chi_{\bar{K}}(w_i)$$

  die Sprache $\bar{K}$ repräsentiert, folgt $\bar{K} \neq dom(M_{w_i})$ für alle $i \geq 1$
- Dies impliziert $\bar{K} \notin \mathsf{RE}$, da die Gesamtheit der Zeilen von B wegen

$$\{dom(M_{w_i}) \mid i \geq 1\} = \{A \subseteq \{0,1\}^* \mid A \in \mathsf{RE}\}$$

  die Klasse aller Binärsprachen in RE repräsentiert

Unentscheidbarkeit des speziellen Halteproblems

Beweis von $\bar{K} \notin \mathsf{RE}$

- Angenommen, die Sprache

$$\bar{K} = \{w \in \{0,1\}^* \mid M_w(w)\uparrow\} \qquad (*)$$

  wäre semi-entscheidbar

- Dann existiert eine DTM M_{w_i} mit

$$dom(M_{w_i}) = \bar{K} \qquad (**)$$

- Dies führt jedoch auf einen Widerspruch:

χ_H	w_1	w_2	w_3	$\cdots$
w_1	**0**	1	0	...
w_2	0	**1**	1	...
w_3	1	1	**0**	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	
w_i	1	0	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	

$$w_i \in \bar{K} \underset{(*)}{\Leftrightarrow} M_{w_i}(w_i)\uparrow \Leftrightarrow w_i \notin dom(M_{w_i}) \underset{(**)}{\Leftrightarrow} w_i \notin \bar{K} \qquad \lightning$$

$\square$

Bemerkung

- Die Methode in obigem Beweis wird als Diagonalisierung bezeichnet
- Mit dieser Beweistechnik lässt sich auch eine Sprache in $\mathsf{REC} \setminus \mathsf{CSL}$ definieren (siehe Übungen)

Der Reduktionsbegriff

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt auf $B \subseteq \Gamma^*$ reduzierbar (kurz: $A \leq B$), falls eine berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ ex., so dass gilt:

$$\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B$$

Beispiel

- Es gilt $K \leq H$ mittels $f : w \mapsto w\#w$, da für alle $w \in \{0,1\}^*$ gilt:

$$w \in K \quad \Leftrightarrow \quad M_w(w)\downarrow \quad \Leftrightarrow \quad w\#w \in H$$

- Es gilt sogar $A \leq H$ für jede Binärsprache $A \in \mathrm{RE}$ mittels $f : x \mapsto w\#x$, wobei w die Kodierung einer DTM M_w mit $dom(M_w) = A$ ist:

$$x \in A \quad \Leftrightarrow \quad M_w(x)\downarrow \quad \Leftrightarrow \quad w\#x \in H$$

$\triangleleft$

Der Vollständigkeitsbegriff

Definition

- Eine Sprache B heißt hart für eine Sprachklasse $\mathcal{C}$ (kurz: $\mathcal{C}$-hart oder $\mathcal{C}$-schwer), falls jede Sprache $A \in \mathcal{C}$ auf B reduzierbar ist:

$$\forall A \in \mathcal{C} : A \leq B$$

- Eine $\mathcal{C}$-harte Sprache B, die zu $\mathcal{C}$ gehört, heißt $\mathcal{C}$-vollständig

Beispiel (H ist hart für $\{A \subseteq \{0,1\}^* \mid A \in \mathrm{RE}\}$ und vollständig für RE)

- Wir wissen bereits, dass alle Binärsprachen in RE auf H reduzierbar sind
- Sei nun $A \in \mathrm{RE}$ eine Sprache über einem bel. Alphabet
- Dann ist A auf die Binärsprache $A' = \{bin(x) \mid x \in A\}$ mittels der Fkt. $x \mapsto bin(x)$ reduzierbar und mit A ist auch A' in RE
- Somit folgt $A \leq A' \leq H$, was wg. der Transitivität von $\leq$ (s. Übungen) $A \leq H$ impliziert ◁

Abgeschlossenheit von REC unter $\leq$

Definition

Eine Sprachklasse $\mathcal{C}$ heißt unter $\leq$ abgeschlossen, wenn für beliebige Sprachen A, B gilt:

$$A \leq B \wedge B \in \mathcal{C} \Rightarrow A \in \mathcal{C}$$

Satz

Die Klasse REC ist unter $\leq$ abgeschlossen

Beweis

- Gelte $A \leq B$ mittels f und sei $B \in \text{REC}$
- Wegen $B \in \text{REC}$ ex. eine DTM M_B, die χ_B berechnet
- Betrachte folgende DTM M_A:
  - M_A berechnet bei Eingabe x zuerst den Wert $f(x)$ und
  - simuliert dann M_B bei Eingabe $f(x)$

Abgeschlossenheit von REC und RE unter $\leq$

Satz

Die Klasse REC ist unter $\leq$ abgeschlossen

Beweis.

- Gelte $A \leq B$ mittels f und sei $B \in \text{REC}$
- Dann ex. eine DTM M_B, die χ_B berechnet
- Betrachte folgende DTM M_A:
  - M_A berechnet bei Eingabe x zuerst den Wert $f(x)$ und
  - simuliert dann M_B bei Eingabe $f(x)$
- Wegen $x \in A \Leftrightarrow f(x) \in B$ ist $\chi_A(x) = \chi_B(f(x))$ und daher folgt

$$M_A(x) = M_B(f(x)) = \chi_B(f(x)) = \chi_A(x)$$

- Also berechnet M_A die Funktion χ_A, d.h. $A \in \text{REC}$ $\square$

Bemerkung

Die Abgeschlossenheit von RE unter $\leq$ folgt analog (siehe Übungen)

H ist nicht entscheidbar

Korollar

- $A \leq B \wedge A \notin \mathrm{REC} \Rightarrow B \notin \mathrm{REC}$
- $A \leq B \wedge A \notin \mathrm{RE} \Rightarrow B \notin \mathrm{RE}$

Beweis

Aus der Annahme, dass B entscheidbar (bzw. semi-entscheidbar) ist, folgt wegen $A \leq B$, dass dies auch auf A zutrifft (Widerspruch). $\square$

Bemerkung

Wegen $K \leq H$ überträgt sich somit die Unentscheidbarkeit von K auf H

Korollar

$H \notin \mathrm{REC}$, d.h. H ist unentscheidbar

Das Halteproblem bei leerem Band

Definition

Das Halteproblem bei leerem Band ist die Sprache

$$H_0 = \left\{ w \in \{0,1\}^* \;\middle|\; \begin{array}{l}\text{die DTM } M_w \\ \text{hält bei Eingabe } \varepsilon\end{array} \right\}$$

Satz

H_0 ist RE-vollständig und somit unentscheidbar

Beweis

- $H_0 \in \mathrm{RE}$ folgt wegen $H_0 \leq H \in \mathrm{RE}$ mittels der Reduktionsfunktion $w \mapsto w\#\varepsilon$

χ_H	w_1	w_2	w_3	$\cdots$
w_1	**0**	1	0	$\cdots$
w_2	**0**	1	1	$\cdots$
w_3	**1**	1	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

χ_{H_0}	w_1 $(=\varepsilon)$
w_1	**0**
w_2	**0**
w_3	**1**
$\vdots$	$\vdots$

H_0 ist RE-vollständig

Beweis

- $H_0 \in \mathrm{RE}$ folgt wegen $H_0 \le H \in \mathrm{RE}$ mittels der Funktion $w \mapsto w\#\varepsilon$
- Sei $A \in \mathrm{RE}$ und sei M_A eine DTM mit $dom(M_A) = A$
- Da jede Sprache $A \in \mathrm{RE}$ auf eine Binärsprache in RE reduzierbar ist, können wir o.B.d.A. $A \subseteq \{0,1\}^*$ annehmen
- Um A auf H_0 zu reduzieren, transformieren wir x in die Kodierung w_x einer DTM M_{w_x}, die ihre Eingabe ignoriert und $M_A(x)$ simuliert
- Dann gilt

$$x \in A \;\Leftrightarrow\; M_A(x)\!\downarrow \;\Leftrightarrow\; M_{w_x}(\varepsilon)\!\downarrow \;\Leftrightarrow\; w_x \in H_0$$

- Da zudem die Funktion $x \mapsto w_x$ total und berechenbar ist, folgt $A \le H_0$ $\square$

Bemerkung

Derselbe Beweis zeigt, dass alle Spalten und auch die Diagonale der Matrix $B = (b_{ij})$ mit $b_{ij} = \chi_H(w_i\#w_j)$ RE-vollständige Sprachen repräsentieren

Der Satz von Rice

- Oft möchte man wissen, ob die von einer DTM akz. Sprache (bzw. die von ihr ber. partielle Funktion) eine gewisse Eigenschaft hat oder nicht
- Solche Eigenschaften werden semantisch genannt, da sie nur vom Akzeptanz- bzw. Ein/Ausgabeverhalten der geg. DTM abhängen
- Der Satz von Rice besagt, dass so gut wie alle semantischen Eigenschaften von Turingmaschinen unentscheidbar sind
- Zum Beispiel ist das Problem unentscheidbar, ob eine gegebene DTM bei allen Eingaben hält (also eine totale Funktion berechnet)
- Formal lassen sich semantische Eigenschaften durch eine Menge $\mathcal{F}$ von partiellen Funktionen bzw. durch eine Sprachklasse $\mathcal{S}$ beschreiben
- Eine DTM M_w hat dann die Eigenschaft $\mathcal{F}$ (bzw. $\mathcal{S}$), wenn sie eine Funktion in $\mathcal{F}$ berechnet (bzw. eine Sprache in $\mathcal{S}$ akzeptiert)
- Eine semantische Eigenschaft $\mathcal{F}$ (bzw. $\mathcal{S}$) heißt trivial, falls entweder alle DTMs oder keine diese Eigenschaft haben

Definition

- Zu einer Klasse $\mathcal{F}$ von partiellen Funktionen definieren wir die Sprache

$$L_{\mathcal{F}} = \left\{ w \in \{0,1\}^* \;\middle|\; \text{die DTM } M_w \text{ ber. eine partielle Fkt. in } \mathcal{F} \right\}$$

- Die Eigenschaft $\mathcal{F}$ heißt trivial, wenn $L_{\mathcal{F}} = \emptyset$ oder $L_{\mathcal{F}} = \{0,1\}^*$ ist

Der Satz von Rice besagt, dass jede semantische Eigenschaft von DTMs entweder trivial oder unentscheidbar ist

Satz (Satz von Rice)

Für jede nicht triviale Eigenschaft $\mathcal{F}$ ist die Sprache $L_{\mathcal{F}}$ unentscheidbar

Der Satz von Rice

Beispiel

- Betrachte die beiden Sprachen
$$L_1 = \{w \in \{0,1\}^* \mid M_w(0^n) = 0^{n+1} \text{ für alle } n \geq 0\} \text{ und}$$
$$L_2 = \{w \in \{0,1\}^* \mid M_w(x) = \hat{\chi}_K(x) \text{ für alle } x \in \{0,1\}^*\}$$
- Dann gilt $L_i = L_{\mathcal{F}_i}$ für die Eigenschaften
$$\mathcal{F}_1 = \{f \in \mathrm{FREC}_p \mid f(0^n) = 0^{n+1} \text{ für alle } n \geq 0\} \text{ und}$$
$$\mathcal{F}_2 = \{f \in \mathrm{FREC}_p \mid f(x) = \hat{\chi}_K(x) \text{ für alle } x \in \{0,1\}^*\}$$
- $\mathcal{F}_1$ ist nicht trivial, da die partiellen Fkten $f, u : \{0,1\}^* \to \{0,1\}^* \cup \{\uparrow\}$ mit $f(x) = x0$ und $u(x) = \uparrow$ für alle $x \in \{0,1\}^*$ berechenbar sind und $f \in \mathcal{F}_1$ sowie $u \notin \mathcal{F}_1$ ist
- Auch $\mathcal{F}_2$ ist nicht trivial, da $\hat{\chi}_K$ wegen $K \in \mathrm{RE}$ eine berechenbare partielle Funktion in $\mathcal{F}_2$ ist, während $f, u \notin \mathcal{F}_2$ sind
- Also sind die beiden Sprachen L_1 und L_2 nach dem Satz von Rice unentscheidbar

Der Satz von Rice

Beispiel (Fortsetzung)

- Dagegen liefert der Satz von Rice nicht die Unentscheidbarkeit folgender Sprachen:

$$L_4 = \{w \in \{0,1\}^* \mid M_w(x) = \hat{\chi}_{\bar{K}}(x) \text{ für alle } x \in \{0,1\}^*\} \text{ und}$$
$$L_5 = \{w \in \{0,1\}^* \mid M_w(0^n) \text{ hält für alle } n \geq 0 \text{ nach } n \text{ Schritten}\}$$

- Es gilt $L_4 = L_{\mathcal{F}_4}$ für die Eigenschaft $\mathcal{F}_4 = \{\hat{\chi}_{\bar{K}}\}$, d.h. L_4 beschreibt zwar eine semantische Eigenschaft von DTMs, die sich nur auf deren Ein-/Ausgabeverhalten bezieht
- Da aber $\bar{K} \notin \mathsf{RE}$ und somit $\hat{\chi}_{\bar{K}}$ nicht berechenbar ist, handelt es sich bei $\mathcal{F}_4$ um eine triviale Eigenschaft: $L_4 = L_{\mathcal{F}_4} = \varnothing$
- Die Sprache L_5 bezieht sich nicht nur auf das Ein-/Ausgabeverhalten von DTMs, sondern auch auf deren Laufzeit
- Daher existiert für L_5 keine Funktionenklasse $\mathcal{F}$ mit $L_5 = L_{\mathcal{F}}$

Der Satz von Rice

Satz (Satz von Rice)

Für jede nicht triviale Eigenschaft $\mathcal{F}$ ist die Sprache $L_{\mathcal{F}}$ unentscheidbar

Beweisidee

- Die Idee besteht darin, H_0 auf $L_{\mathcal{F}}$ (oder auf $\overline{L}_{\mathcal{F}}$) zu reduzieren, indem wir für eine gegebene DTM M_w eine DTM $M_{w'}$ konstruieren mit

  $$w \in H_0 \Leftrightarrow M_{w'} \text{ berechnet (k)eine partielle Funktion in } \mathcal{F}$$

- Hierzu lassen wir $M_{w'}$ bei Eingabe x zunächst einmal die DTM M_w bei Eingabe ε simulieren
- Falls $w \notin H_0$ ist, berechnet $M_{w'}$ also die überall undefinierte Funktion $u(x) = \uparrow$ für alle $x \in \{0,1\}^*$
- Damit die Reduktion gelingt, müssen wir nur noch dafür sorgen, dass $M_{w'}$ im Fall $w \in H_0$ eine partielle Funktion f berechnet, die sich bzgl. der Eigenschaft $\mathcal{F}$ von u unterscheidet, d.h. $f \in \mathcal{F} \Leftrightarrow u \notin \mathcal{F}$
- Da $\mathcal{F}$ nicht trivial ist, ex. eine DTM M, die ein solches f berechnet

Der Satz von Rice

Satz (Satz von Rice)

Für jede nicht triviale Eigenschaft $\mathcal{F}$ ist die Sprache $L_{\mathcal{F}}$ unentscheidbar

Beweis

- Sei M eine DTM, die eine Funktion f mit $f \in \mathcal{F} \Leftrightarrow u \notin \mathcal{F}$ berechnet
- Betrachte die Reduktionsfunktion

  $h(w) = w'$, wobei w' die Kodierung einer DTM ist, die bei Eingabe x zunächst die DTM $M_w(\varepsilon)$ simuliert und im Fall, dass $M_w(\varepsilon)$ hält, mit der Simulation von $M(x)$ fortfährt

- Dann ist $h : w \mapsto w'$ eine totale berechenbare Funktion und es gilt

$$\begin{aligned} w \in H_0 \quad &\Rightarrow \quad M_{w'} \text{ berechnet } f \\ w \notin H_0 \quad &\Rightarrow \quad M_{w'} \text{ berechnet } u \end{aligned}$$

- Dies zeigt, dass h das Problem H_0 auf $L_{\mathcal{F}}$ (oder auf $\overline{L}_{\mathcal{F}}$) reduziert, und da H_0 unentscheidbar ist, muss auch $L_{\mathcal{F}}$ unentscheidbar sein $\square$

Der Satz von Rice für Akzeptoren

Der Satz von Rice gilt auch für Eigenschaften, die das Akzeptanzverhalten einer gegebenen Turingmaschine betreffen

Satz (Satz von Rice für Spracheigenschaften)

Für eine beliebige Sprachklasse $\mathcal{S}$ sei

$$L_{\mathcal{S}} = \{w \in \{0,1\}^* \mid L(M_w) \in \mathcal{S}\}$$

Dann ist $L_{\mathcal{S}}$ unentscheidbar, außer wenn $L_{\mathcal{S}} = \emptyset$ oder $L_{\mathcal{S}} = \{0,1\}^*$ ist

Beweis

Siehe Übungen

Entscheidungsprobleme für Sprachklassen

Neben dem Wortproblem sind für eine Sprachklasse $\mathcal{C}$ auch folgende Entscheidungsprobleme interessant:

Das Leerheitsproblem ($\mathrm{LP}_{\mathcal{C}}$)

Gegeben: Eine Sprache L aus $\mathcal{C}$.
Gefragt: Ist $L \neq \varnothing$?

Das Äquivalenzproblem ($\mathrm{\ddot{A}P}_{\mathcal{C}}$)

Gegeben: Zwei Sprachen L_1 und L_2 aus $\mathcal{C}$.
Gefragt: Gilt $L_1 = L_2$?

Das Schnittproblem ($\mathrm{SP}_{\mathcal{C}}$)

Gegeben: Zwei Sprachen L_1 und L_2 aus $\mathcal{C}$.
Gefragt: Ist $L_1 \cap L_2 \neq \varnothing$?

Hierbei repräsentieren wir Sprachen in $\mathcal{C} = \mathsf{REG}, \mathsf{CFL}, \mathsf{CSL}, \mathsf{RE}$ durch entsprechende Grammatiken und Sprachen in $\mathcal{C} = \mathsf{DCFL}, \mathsf{DCSL}$ durch entsprechende Akzeptoren (also DPDAs bzw. DLBAs).

Das Postsche Korrespondenzproblem

Postsches Korrespondenzproblem über Σ (kurz PCP_Σ)

gegeben: n Wortpaare $(x_1, y_1), \ldots, (x_n, y_n) \in \Sigma^+ \times \Sigma^*$

gefragt: Gibt es eine Folge $s = (i_1, \ldots, i_k)$ von $k \geq 1$ Indizes $i_j \in \{1, \ldots, n\}$ mit $x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k}$?

- Das modifizierte PCP über Σ (kurz MPCP_Σ) fragt nach einer Lösung $s = (i_1, \ldots, i_k)$ mit $i_1 = 1$
- Wir notieren eine PCP-Instanz meist in Form einer Matrix $\binom{x_1 \ldots x_n}{y_1 \ldots y_n}$ und kodieren sie durch das Wort $x_1 \# y_1 \# \ldots \# x_n \# y_n$ (o.B.d.A. sei $\# \notin \Sigma$)

Beispiel

Die Instanz $I = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} a & ab & caa \\ aca & bc & aa \end{pmatrix}$ besitzt wegen

$$x_1 x_3 x_2 x_3 = acaaabcaa$$
$$y_1 y_3 y_2 y_3 = acaaabcaa$$

die PCP-Lösung $s = (1, 3, 2, 3)$, die auch eine MPCP-Lösung ist $\triangleleft$

Das binäre PCP

Lemma

Für jedes Alphabet Σ gilt $\mathrm{PCP}_\Sigma \le \mathrm{PCP}_{\{0,1\}}$.

Beweis

- Sei $\Sigma = \{a_0, \ldots, a_{m-1}\}$ und sei $r = \max(1, \lceil \log_2(m) \rceil)$
- Wir kodieren a_i durch eine r-stellige Binärzahl $bin_r(a_i)$ mit dem Wert i und ein Wort $w = w_1 \ldots w_\ell \in \Sigma^\ell$ durch $bin(w) = bin_r(w_1) \ldots bin_r(w_\ell)$
- Nun folgt $\mathrm{PCP}_\Sigma \le \mathrm{PCP}_{\{0,1\}}$ mittels der Reduktionsfunktion

$$f : \begin{pmatrix} x_1 \ldots x_n \\ y_1 \ldots y_n \end{pmatrix} \mapsto \begin{pmatrix} bin(x_1) \ldots bin(x_n) \\ bin(y_1) \ldots bin(y_n) \end{pmatrix}$$

$\square$

Beispiel

Sei $\Sigma = \{a, b, c\}$. Dann ist $r = \max(1, \lceil \log_2(3) \rceil) = 2$ und $bin_2(a) = 00$, $bin_2(b) = 01$ und $bin_2(c) = 10$. Somit ist

$$f\begin{pmatrix} a & ab & caa \\ aca & bc & aa \end{pmatrix} = \begin{pmatrix} 00 & 0001 & 100000 \\ 001000 & 0110 & 0000 \end{pmatrix}$$

$\triangleleft$

Reduktion des MPCP auf das PCP

Wir schreiben für $\mathrm{PCP}_{\{0,1\}}$ auch PCP (bzw. MPCP für $\mathrm{MPCP}_{\{0,1\}}$)

Satz

$\mathrm{MPCP} \leq \mathrm{PCP}$

Beweis

- Wir zeigen $\mathrm{MPCP} \leq \mathrm{PCP}_\Sigma$ für $\Sigma = \{0, 1, \langle, |, \rangle\}$
- Für ein Wort $w = w_1 \dots w_\ell \in \{0,1\}^\ell$ sei

$\overset{\leftarrow\!\rvert}{w}$	$\overleftarrow{w}$	$\overset{\vert\!\overline{\;\;}}{w}$	$\overset{\overline{\;\;}\!\vert}{w}$
$\langle w_1 \vert \dots \vert w_\ell \vert$	$\langle w_1 \vert \dots \vert w_\ell$	$\vert w_1 \vert \dots \vert w_\ell$	$w_1 \vert \dots \vert w_\ell \vert$

- Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ:

$$f : \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{pmatrix} \mapsto \begin{pmatrix} \overset{\leftarrow\!\rvert}{x_1} & \overset{\overline{\;\;}\!\vert}{x_1} & \dots & \overset{\overline{\;\;}\!\vert}{x_n} & \rangle \\ \overleftarrow{y_1} & \overset{\vert\!\overline{\;\;}}{y_1} & \dots & \overset{\vert\!\overline{\;\;}}{y_n} & \vert\rangle \end{pmatrix}$$

Reduktion des MPCP auf das PCP

Beweis

- Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ:

$$f : \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{pmatrix} \mapsto \begin{pmatrix} \overleftarrow{x_1} & \overrightarrow{x_1} & \dots & \overrightarrow{x_n} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \dots & \overleftarrow{y_n} & |\rangle \end{pmatrix}$$

Beispiel

- Betrachte die Reduktion der MPCP-Instanz

$$I = \begin{pmatrix} 00 & 1 & 101 & 11 \\ 001 & 11 & 0 & 1 \end{pmatrix} \text{ auf } f(I) = \begin{pmatrix} \langle|0|0| & 0|0| & 1| & 1|0|1| & 1|1| & \rangle \\ \langle|0|0|1 & |0|0|1 & |1|1 & |0 & |1 & |\rangle \end{pmatrix}$$

- Dann entspricht der MPCP-Lösung $s = (1, 3, 2)$ mit dem Lösungswort

$$x_1\, x_3\, x_2 = 001011 = 001011 = y_1\, y_3\, y_2$$

  für I die PCP-Lösung $s' = (1, 4, 3, 6)$ mit dem Lösungswort

$$\overleftarrow{x_1}\,\overrightarrow{x_3}\,\overrightarrow{x_2}\rangle = \langle|0|0|1|0|1|1|\rangle = \langle|0|0|1|0|1|1|\rangle = \overleftarrow{y_1}\,\overleftarrow{y_3}\,\overleftarrow{y_2}|\rangle$$

  für $f(I)$ und umgekehrt ◁

Beweis von $\mathrm{MPCP} \leq \mathrm{PCP}$

- Wir reduzieren MPCP mittels folgender Funktion f auf PCP_Σ:

$$f : \begin{pmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{pmatrix} \mapsto \begin{pmatrix} \overleftrightarrow{x_1} & \overrightarrow{x_1} & \dots \overrightarrow{x_n} & \rangle \\ \overleftarrow{y_1} & \overleftarrow{y_1} & \dots \overleftarrow{y_n} & |\rangle \end{pmatrix} = \begin{pmatrix} x'_1 & x'_2 & \dots & x'_{n+2} \\ y'_1 & y'_2 & \dots & y'_{n+2} \end{pmatrix}$$

- Da jede MPCP-Lösung $s = (1, i_2, \dots, i_k)$ für I auf eine PCP-Lösung $s' = (1, i_2 + 1, \dots, i_k + 1, n + 2)$ für $f(I)$ führt, folgt

$$I \in \mathrm{MPCP} \Rightarrow f(I) \in \mathrm{PCP}_\Sigma$$

- Für die Rückrichtung sei $s' = (i_1, \dots, i_k)$ eine PCP-Lösung für $f(I)$
- Dann muss $i_1 = 1$ sein, da nur die beiden Einträge $x'_1 = \overleftrightarrow{x_1}$ und $y'_1 = \overleftarrow{y_1}$ in der ersten Spalte von $f(I)$ mit dem gleichen Zeichen beginnen
- Zudem muss $i_k = n + 2$ sein, da nur die beiden Einträge $x'_{n+2} = \rangle$ und $y'_{n+2} = |\rangle$ in der letzten Spalte von $f(I)$ mit dem gleichen Zeichen enden
- Wählen wir die Lösungsfolge s' von minimaler Länge, so gilt zudem $i_j \in \{2, \dots, n+1\}$ für $j = 2, \dots, k-1$
- Folglich ist $s = (i_1, i_2 - 1, \dots, i_{k-1} - 1)$ eine MPCP-Lösung für I $\square$

Unentscheidbarkeit des PCP

Satz

PCP ist RE-vollständig und damit unentscheidbar

Beweis.

- PCP ist semi-entscheidbar, da eine DTM systematisch nach einer Lösung suchen kann
- Um zu zeigen, dass PCP RE-hart ist, sei A eine beliebige Sprache in RE und sei $G = (V, \Sigma, P, S)$ eine Typ-0 Grammatik für A
- Wir zeigen $A \leq \mathrm{MPCP}_\Gamma$ für $\Gamma = V \cup \Sigma \cup \{\langle, |, \rangle\}$
- Wegen $\mathrm{MPCP}_\Gamma \leq \mathrm{PCP}$ folgt hieraus $A \leq \mathrm{PCP}$

Unentscheidbarkeit des PCP

Beweis (Fortsetzung)

- Sei $G = (V, \Sigma, P, S)$ eine Typ-0 Grammatik für A
- Wir zeigen $A \le \mathrm{MPCP}_\Gamma$ für $\Gamma = V \cup \Sigma \cup \{\langle, |, \rangle\}$
- Idee: Transformiere $w \in \Sigma^*$ in eine Instanz $f(w) = \binom{x_1 \dots x_n}{y_1 \dots y_n}$, so dass für jede Indexfolge $s = (i_1, \dots, i_k)$ gilt:

  s ist genau dann eine MPCP-Lösung für $f(w)$, wenn das zugehörige Lösungswort $z = x_{i_1} \dots x_{i_k} = y_{i_1} \dots y_{i_k}$ eine Ableitung von w in G kodiert, d.h. z hat die Form $z = \langle\,|\,\alpha_0\,|\,\alpha_1\,|\dots|\,\alpha_m\,|\,\rangle$ und es gilt

  $$S = \alpha_0 \Rightarrow^* \alpha_1 \Rightarrow^* \cdots \Rightarrow^* \alpha_m = w$$

- Konkret bilden wir $f(w)$ aus folgenden Wortpaaren:
  - $(\langle\,, \langle\,|\,S)$ „Startpaar“
  - für jede Regel $l \to r$ in P: (l, r) „Ableitungspaare“
  - für alle $a \in V \cup \Sigma \cup \{|\}$: (a, a) „Kopierpaare“
  - sowie das Paar $(w\,|\,\rangle, \rangle)$ „Abschlusspaar“

Unentscheidbarkeit des PCP

Beispiel

- Sei $G = (\{S\}, \{a, b\}, \{S \to aSbS, \varepsilon\}, S)$ und $w = aabb$
- Die MPCP-Instanz $f(aabb)$ enthält dann die acht Wortpaare

$$f(aabb) = \begin{pmatrix} \langle & S & S & S & a & b & | & aabb|\rangle \\ \langle|S & aSbS & \varepsilon & S & a & b & | & \rangle \end{pmatrix}$$

- Der Ableitung $\underline{S} \Rightarrow a\underline{S}bS \Rightarrow aa\underline{S}b\underline{S}bS \Rightarrow aa\underline{S}bbS \Rightarrow aabb\underline{S} \Rightarrow aabb$ entspricht dann die MPCP-Lösung

$$(1, 7, 2, 7, 5, 2, 6, 4, 7, 5, 5, 4, 6, 3, 6, 4, 7, 5, 5, 3, 6, 6, 4, 7, 5, 5, 6, 6, 3, 7, 8)$$

mit dem Lösungswort

$$\langle|S|aSbS|aaSbSbS|aaSbbS|aabbS|aabb|\rangle$$
$$\langle|S|aSbS|aaSbSbS|aaSbbS|aabbS|aabb|\rangle$$

- Das kürzeste MPCP-Lösungswort für $f(aabb)$ ist

$$\langle|S|aSbS|aaSbSb|aabb|\rangle$$
$$\langle|S|aSbS|aaSbSb|aabb|\rangle$$

- Dieses entspricht der „parallelisierten" Ableitung

$$\underline{S} \Rightarrow a\underline{S}b\underline{S} \Rightarrow^2 aa\underline{S}b\underline{S}b \Rightarrow^2 aabb$$

◁

Unentscheidbarkeit des PCP

Beweis (Schluss)

- Wir bilden $f(w)$ aus folgenden Wortpaaren:
  - $(\langle\,, \langle\,|\,S)$ „Startpaar“
  - für jede Regel $l \to r$ in P: (l, r) „Ableitungspaare“
  - für alle $a \in V \cup \Sigma \cup \{|\}$: (a, a) „Kopierpaare“
  - sowie das Paar $(w\,|\,\rangle, \rangle)$ „Abschlusspaar“
- Nun lässt sich leicht aus einer Ableitung $S = w_0 \Rightarrow \cdots \Rightarrow w_m = w$ von w in G eine MPCP-Lösung s für $f(w)$ mit dem Lösungswort

  $$\langle\,|\,w_0\,|\,w_1\,|\ldots|\,w_m\,|\,\rangle$$

  angeben
- Umgekehrt lässt sich aus jeder MPCP-Lösung s für $f(w)$ auch eine Ableitung von w in G gewinnen, womit

  $$w \in L(G) \Leftrightarrow f(w) \in \mathrm{MPCP}_\Gamma$$

  gezeigt ist $\square$

Unentscheidbarkeit des Schnittproblems für CFL

Das Schnittproblem für kontextfreie Grammatiken ($\mathrm{SP}_{\mathsf{CFL}}$)

Gegeben: Zwei kontextfreie Grammatiken G_1 und G_2.

Gefragt: Ist $L(G_1) \cap L(G_2) \neq \varnothing$?

Satz

Das Schnittproblem für kontextfreie Grammatiken ist RE-vollständig und somit unentscheidbar

Unentscheidbarkeit des Schnittproblems für CFL

Satz

Das Schnittproblem für kontextfreie Grammatiken ist RE-vollständig und somit unentscheidbar

Beweis

- Das Problem $\mathrm{SP}_{\mathsf{CFL}}$ ist semi-entscheidbar, da eine DTM systematisch nach einem Wort $x \in L(G_1) \cap L(G_2)$ suchen kann
- Um PCP auf $\mathrm{SP}_{\mathsf{CFL}}$ zu reduzieren, betrachten wir für eine Folge $z = (x_1, \dots, x_k)$ von Strings $x_i \in \{0,1\}^*$ die Sprache

  $$L_z = \{i_n \dots i_1 \# x_{i_1} \cdots x_{i_n} \mid n \geq 1, 1 \leq i_1, \dots, i_n \leq k\}$$

  über dem Alphabet $\Sigma = \{0, 1, \dots, k, \#\}$
- Die Sprache L_z wird von der Grammatik $G_z = (\{A\}, \Sigma, P_z, A)$ mit der Regelmenge

  $$P_z\colon\ A \to 1Ax_1, \dots, kAx_k, 1\#x_1, \dots, k\#x_k$$

  erzeugt

Unentscheidbarkeit des Schnittproblems für CFL

Reduktion von PCP auf das Schnittproblem für CFL

- Zu einer PCP-Instanz $I = \binom{x_1 \ldots x_k}{y_1 \ldots y_k}$ bilden wir das Paar (G_{z_1}, G_{z_2}), wobei $z_1 = (x_1, \ldots, x_k)$ und $z_2 = (y_1, \ldots, y_k)$ ist
- Dann ist $L(G_{z_1}) \cap L(G_{z_2})$ die Sprache

$$\{i_n \ldots i_1 \# x_{i_1} \ldots x_{i_n} \mid 1 \leq n,\ x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}\}$$

- Folglich ist $s = (i_1, \ldots, i_n)$ genau dann eine Lösung für I, wenn $i_n \ldots i_1 \# x_{i_1} \ldots x_{i_n} \in L(G_{z_1}) \cap L(G_{z_2})$ ist, d.h. es gilt

$$I \in \mathrm{PCP} \Leftrightarrow L(G_{z_1}) \cap L(G_{z_2}) \neq \varnothing$$

- Also vermittelt $f : I \mapsto (G_{z_1}, G_{z_2})$ eine Reduktion von PCP auf das Schnittproblem für CFL $\square$

Unentscheidbarkeit des Schnittproblems für CFL

Beispiel

- Die PCP-Instanz

$$I = \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} 0 & 001 & 01100 \\ 00110 & 01011 & 00 \end{pmatrix}$$

wird auf das Grammatikpaar (G_{z_1}, G_{z_2}) mit folgenden Regeln reduziert:

$$P_{z_1}\colon A \to 1A0,\ 2A001,\ 3A01100,\ 1\#0,\ 2\#001,\ 3\#01100$$

$$P_{z_2}\colon A \to 1A00110,\ 2A01011,\ 3A00,\ 1\#00110,\ 2\#01011,\ 3\#00$$

- Der PCP-Lösung $s = (1, 3, 2, 3)$ entspricht dann das Wort

$$\begin{aligned} 3231\#x_1x_3x_2x_3 &= 3231\#00110000101100 \\ &= 3231\#00110000101100 = 3231\#y_1y_3y_2y_3 \end{aligned}$$

im Schnitt $L(G_{z_1}) \cap L(G_{z_2})$ $\triangleleft$

Unentscheidbarkeit des Schnittproblems für DCFL

Das Schnittproblem für DPDAs ($\mathrm{SP}_{\mathsf{DPDA}}$)

Gegeben: Zwei DPDAs M_1 und M_2.

Gefragt: Gilt $L(M_1) \cap L(M_2) \neq \varnothing$?

Korollar

$\mathrm{SP}_{\mathsf{DPDA}}$ ist RE-vollständig und daher unentscheidbar

Beweis

Für die Sprache $L_z = \{i_n \ldots i_1 \# x_{i_1} \cdots x_{i_n} \mid n \geq 1, 1 \leq i_1, \ldots, i_n \leq k\}$ lässt sich leicht ein DPDA M_z angeben mit $L(M_z) = L_z$ $\square$

Das Leerheitsproblem für DLBAs

Das Leerheitsproblem für DLBAs ($\mathrm{LP}_{\mathsf{DLBA}}$)

Gegeben: Ein DLBA M.

Gefragt: Ist $L(M) \neq \varnothing$?

Satz

$\mathrm{LP}_{\mathsf{DLBA}}$ ist RE-vollständig und daher unentscheidbar

Beweisidee

- Es ist leicht zu sehen, dass $\mathrm{LP}_{\mathsf{DLBA}} \in \mathsf{RE}$ ist
- Wir reduzieren PCP auf $\mathrm{LP}_{\mathsf{DLBA}}$
- Hierzu überführen wir eine PCP-Instanz $I = \binom{z_1}{z_2}$ in einen DLBA M mit

  $$L(M) = L_{z_1} \cap L_{z_2}$$
- Dann ist die Funktion $f : I \mapsto M$ berechenbar und es gilt

  $$I \in \mathrm{PCP} \;\Leftrightarrow\; L_{z_1} \cap L_{z_2} \neq \varnothing \;\Leftrightarrow\; L(M) \neq \varnothing \;\Leftrightarrow\; M \in \mathrm{LP}_{\mathsf{DLBA}} \qquad \square$$

Das Äquivalenzproblem für kontextfreie Sprachen

Das Äquivalenzproblem für kontextfreie Grammatiken ($\ddot{\mathrm{A}}\mathrm{P}_{\mathsf{CFL}}$)

Gegeben: Zwei kontextfreie Grammatiken G_1 und G_2.

Gefragt: Gilt $L(G_1) = L(G_2)$?

Satz

$\ddot{\mathrm{A}}\mathrm{P}_{\mathsf{CFL}}$ ist unentscheidbar

Beweisidee

- Wir reduzieren $\overline{\mathrm{PCP}}$ auf $\ddot{\mathrm{A}}\mathrm{P}_{\mathsf{CFL}}$
- Für $I = \binom{z_1}{z_2}$ gilt

$$I \notin \mathrm{PCP} \;\Leftrightarrow\; L_{z_1} \cap L_{z_2} = \varnothing \;\Leftrightarrow\; \overline{L}_{z_1} \cup \overline{L}_{z_2} = \Sigma^*$$

- Daher vermittelt die Funktion $f : I \mapsto \langle G_1, G_2 \rangle$ die gewünschte Reduktion, wobei G_1 und G_2 kontextfreie Grammatiken sind mit

$$L(G_1) = \overline{L}_{z_1} \cup \overline{L}_{z_2} \text{ und } L(G_2) = \Sigma^*$$

$\square$

Dagegen ist es nicht schwer,

- für eine kontextsensitive Grammatik G und ein Wort x zu entscheiden, ob $x \in L(G)$ ist (Wortproblem $\mathrm{WP}_{\mathsf{CSL}}$)
- für eine kontextfreie Grammatik G zu entscheiden, ob $L(G) \neq \varnothing$ ist (Leerheitsproblem $\mathrm{LP}_{\mathsf{CFL}}$)
- für zwei reguläre Grammatiken G_1 und G_2 zu entscheiden, ob $L(G_1) = L(G_2)$ ist (Äquivalenzproblem $\mathrm{\ddot{A}P}_{\mathsf{REG}}$)
- für zwei reguläre Grammatiken G_1 und G_2 zu entscheiden, ob $L(G_1) \cap L(G_2) \neq \varnothing$ ist (Schnittproblem $\mathrm{SP}_{\mathsf{REG}}$)

Satz

Die Probleme $\mathrm{WP}_{\mathsf{CSL}}$, $\mathrm{LP}_{\mathsf{CFL}}$, $\mathrm{\ddot{A}P}_{\mathsf{REG}}$ und $\mathrm{SP}_{\mathsf{REG}}$ sind entscheidbar

Beweis.

siehe Übungen □

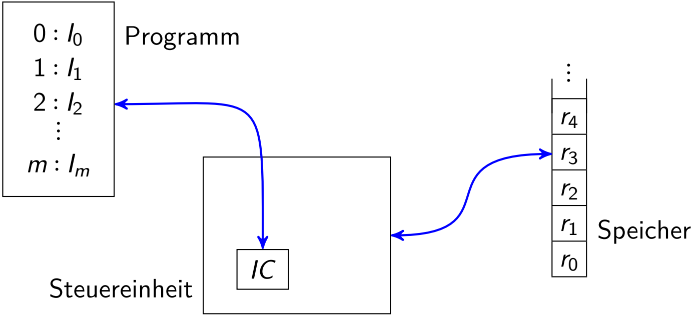
Überblick der (Un-)Entscheidbarkeitsresultate

Folgende Tabelle zeigt, welche der betrachteten Entscheidungsprobleme für die verschiedenen Stufen der Chomsky-Hierarchie entscheidbar sind:

	Wort-problem $x \in L$?	Leerheits-problem $L = \emptyset$?	Äquivalenz-problem $L_1 = L_2$?	Schnitt-problem $L_1 \cap L_2 \neq \emptyset$?
REG	ja	ja	ja	ja
DCFL	ja	ja	ja[a]	nein
CFL	ja	ja	nein	nein
DCSL	ja	nein	nein	nein
CSL	ja	nein	nein	nein
RE	nein	nein	nein	nein

[a] Bewiesen in 1997 von Géraud Sénizergues (Univ. Bordeaux)

Die Registermaschine (random access machine, RAM)



- führt ein Programm $P = (I_0, \ldots, I_m)$ aus, das aus einer endlichen Folge von Befehlen (instructions) I_j, $j = 0, \ldots, m$ besteht
- hat einen Befehlszähler (instruction counter) IC, der die Nummer des nächsten Befehls angibt (zu Beginn ist $IC = 0$)
- verfügt über einen frei adressierbaren Speicher (random access memory) mit unendlich vielen Speicherzellen (Registern) r_i, $i \geq 0$, die beliebig große natürliche Zahlen aufnehmen können

In GOTO-Programmen sind folgende Befehle zulässig (wobei $i, j, c \in \mathbb{N} = \{0, 1, 2, \dots\}$):

Befehl	Semantik
$r_i := r_j + c$	setzt Register r_i auf den Wert $r_j + c$
$r_i := r_j \dot{-} c$	setzt Register r_i auf den Wert $\max(0, r_j - c)$
GOTO j	setzt den Befehlszähler IC auf den Wert j
IF $r_i = c$ **THEN GOTO** j	setzt IC auf j, falls r_i den Wert c hat
HALT	beendet die Programmausführung

Bei Ausführung der ersten beiden Befehle wird zudem der Befehlszähler IC um eins erhöht

GOTO-Berechenbarkeit

Definition

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ heißt GOTO-berechenbar, falls es ein GOTO-Programm $P = (I_0, \ldots, I_m)$ mit folgender Eigenschaft gibt:

- Wird P auf einer RAM mit den Werten $r_i = n_i$ für $i = 1, \ldots, k$, sowie $IC = 0$ und $r_i = 0$ für $i = 0, k+1, k+2, \ldots$ gestartet, so
- hält P genau dann, wenn $(n_1, \ldots, n_k) \in dom(f)$ ist, und
- sobald P hält, hat r_0 den Wert $f(n_1, \ldots, n_k)$

Beispiel

Folgendes GOTO-Programm berechnet die Funktion $f(x, y) = xy$:

0 **IF** $r_1 = 0$ **THEN GOTO** 4
1 $r_1 := r_1 \dot{-} 1$
2 $\cancel{r_0 := r_0 + r_2}$ **GOTO** 5
3 **GOTO** 0
4 **HALT**
5 $r_3 := r_2$
6 **IF** $r_3 = 0$ **THEN GOTO** 3
7 $r_3 := r_3 \dot{-} 1$
8 $r_0 := r_0 + 1$
9 **GOTO** 6

WHILE- und LOOP-Programme

- Die Syntax von WHILE-Programmen ist induktiv wie folgt definiert (wobei $i, j, c \in \mathbb{N}$):
  - Jede Wertzuweisung der Form $x_i := x_j + c$ oder $x_i := x_j \dot{-} c$ ist ein WHILE-Programm.
  - Falls P und Q WHILE-Programme sind, so auch
    - $P; Q$ und
    - **IF** $x_i = c$ **THEN** P **ELSE** Q **END**
    - **WHILE** $x_i \neq c$ **DO** P **END**
- Die Syntax von LOOP-Programmen ist genauso definiert, nur dass Schleifen der Form **LOOP** x_i **DO** P **END** an die Stelle von WHILE-Schleifen treten
- Die Semantik von WHILE-Programmen ist selbsterklärend
- Eine LOOP-Schleife **LOOP** x_i **DO** P **END** wird so oft ausgeführt, wie der Wert von x_i zu Beginn der Schleife angibt

WHILE- und LOOP-Berechenbarkeit

- Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ heißt WHILE-berechenbar, falls es ein WHILE-Programm P mit folgender Eigenschaft gibt:
  - Wird P mit den Werten $x_i = n_i$ für $i = 1, \ldots, k$ gestartet, so
  - hält P genau dann, wenn $(n_1, \ldots, n_k) \in dom(f)$ ist, und
  - sobald P hält, hat x_0 den Wert $f(n_1, \ldots, n_k)$
- Die LOOP-Berechenbarkeit von f ist entsprechend definiert

Beispiel

Die Funktion $f(n_1, n_2) = n_1 n_2$ wird von dem WHILE-Programm

$$\begin{array}{l} \textbf{WHILE } x_1 \neq 0 \textbf{ DO} \\ \quad \cancel{x_0 := x_0 + x_2;} \\ \quad x_1 := x_1 \dot{-} 1 \\ \textbf{END} \end{array} \qquad \left\{ \begin{array}{l} x_3 := x_2; \\ \textbf{WHILE } x_3 \neq 0 \textbf{ DO} \\ \quad x_0 := x_0 + 1;\ x_3 := x_3 \dot{-} 1 \\ \textbf{END} \end{array} \right.$$

sowie von folgendem LOOP-Programm berechnet:

$$\textbf{LOOP } x_1 \textbf{ DO } \cancel{x_0 := x_0 + x_2} \textbf{ END} \quad \textbf{LOOP } x_2 \textbf{ DO } x_0 := x_0 + 1 \textbf{ END}$$ ◁

Satz

Eine partielle Funktion $f \colon \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ ist genau dann GOTO-berechenbar, wenn sie WHILE-berechenbar ist

Simulation eines WHILE- durch ein GOTO-Programm

- Sei P ein WHILE-Programm, das $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ berechnet
- Wir übersetzen P wie folgt in ein äquivalentes GOTO-Programm P'
- P' speichert den Variablenwert x_i im Register r_i
- Damit lassen sich alle Wertzuweisungen von P direkt in entsprechende Befehle von P' transformieren
- Eine Schleife der Form **WHILE** $x_i \neq c$ **DO** Q **END** simulieren wir durch folgendes GOTO-Programmstück:

  M_1 **IF** $r_i = c$ **THEN GOTO** M_2
  Q'
  GOTO M_1
  M_2 $\vdots$

- Ähnlich lässt sich die Verzweigung **IF** $x_i = c$ **THEN** Q_1 **ELSE** Q_2 **END** in ein GOTO-Programmstück transformieren
- Zudem fügen wir ans Ende von P' den HALT-Befehl an

Simulation eines GOTO- durch ein WHILE-Programm

- Sei $P = (I_0, \ldots, I_m)$ ein GOTO-Programm, das $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ berechnet, und sei r_z, $z > k$, ein Register, das in P nicht benutzt wird
- Wir übersetzen P wie folgt in ein äquivalentes WHILE-Programm P':

  $x_z := 0$;
  WHILE $x_z \neq m + 1$ **DO**
    IF $x_z = 0$ **THEN** P'_0 **END**;
      $\vdots$
    IF $x_z = m$ **THEN** P'_m **END**
  END

- Dabei ist P'_ℓ abhängig vom Befehl I_ℓ folgendes WHILE-Programm:

I_ℓ	P'_ℓ
$r_i{:=}r_j{+}c$	$x_i{:=}x_j{+}c$; $x_z{:=}x_z{+}1$
$r_i{:=}r_j{\dot{-}}c$	$x_i{:=}x_j{\dot{-}}c$; $x_z{:=}x_z{+}1$
GOTO j	$x_z{:=}j$
IF $r_i{=}c$ **THEN GOTO** j	**IF** $x_i{=}c$ **THEN** $x_z{:=}j$ **ELSE** $x_z{:=}x_z{+}1$ **END**
HALT	$x_z{:=}m{+}1$

- Man beachte, dass P' nur eine WHILE-Schleife enthält $\square$

Vergleich von LOOP- und WHILE-Berechenbarkeit

- Offensichtlich lässt sich jedes LOOP-Programm durch ein WHILE-Programm simulieren
- Andererseits können LOOP-Programme nur totale Funktionen berechnen, d.h. nicht jedes WHILE-Programm ist durch ein LOOP-Programm simulierbar
- Es gibt auch totale WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind
- Eine solche Funktion kann mittels Diagonalisierung definiert werden
- Ein Beispiel für eine "natürliche" Funktion mit dieser Eigenschaft ist die Ackermannfunktion $a : \mathbb{N}^2 \to \mathbb{N}$, die wie folgt definiert ist

$$a(x,y) = \begin{cases} y+1, & x = 0 \\ a(x-1,1), & x \geq 1, y = 0 \\ a(x-1, a(x, y-1)), & x, y \geq 1 \end{cases}$$

- Als nächstes wollen wir die Äquivalenz von Turing- und GOTO-Berechenbarkeit zeigen
- Da DTMs auf Wörtern und GOTO-Programme auf Zahlen operieren, müssen wir Wörter durch Zahlen kodieren (und umgekehrt)
- Sei $\Sigma = \{a_0, \ldots, a_{m-1}\}$ ein Alphabet
- Dann können wir jedes Wort $x = a_{i_1} \ldots a_{i_n} \in \Sigma^*$ wie folgt durch eine natürliche Zahl kodieren:

$$num_\Sigma(x) = \sum_{j=0}^{n-1} m^j + \sum_{j=1}^{n} i_j m^{n-j} = \begin{cases} n, & m = 1 \\ \frac{m^n-1}{m-1} + (i_1 \ldots i_n)_m, & m \geq 2 \end{cases}$$

- Da die Abbildung $num_\Sigma : \Sigma^* \to \mathbb{N}$ bijektiv ist, können wir umgekehrt jede natürliche Zahl n durch das Wort $str_\Sigma(n) = num_\Sigma^{-1}(n)$ kodieren

Numerische Repräsentation von Wörtern

Beispiel

- Für das unäre Alphabet $\Sigma = \{a\}$ ist $num_\Sigma(a^n) = n$ und $num_\Sigma^{-1}(n) = a^n$

w	ε	a	aa	aaa	$\ldots$
$num_\Sigma(w)$	0	1	2	3	$\ldots$

- Im Fall $\Sigma = \{a, b, c\}$ erhalten wir folgende Werte

w	ε	a	b	c	aa	ab	ac	ba	bb	bc	ca	cb	cc	aaa	$\ldots$
$num_\Sigma(w)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	$\ldots$

- Im Fall $\Sigma = \{0, 1\}$ schreiben wir kurz num und str für num_Σ und str_Σ

x	ε	0	1	00	01	10	$\cdots$
$num(x)$	0	1	2	3	4	5	$\cdots$

n	0	1	2	3	4	5	$\cdots$
$str(n)$	ε	0	1	00	01	10	$\cdots$

$\triangleleft$

- Die Kodierungsfunktion $str : \mathbb{N} \to \{0,1\}^*$ lässt sich wie folgt zu einer Kodierungsfunktion $str_k : \mathbb{N}^k \to \{0,1,\#\}^*$ erweitern:

$$str_k(n_1,\dots,n_k) = str(n_1)\#\dots\#str(n_k)$$

- Nun können wir eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ durch folgende partielle Wortfunktion $\hat{f} : \{0,1,\#\}^* \to \{0,1\}^* \cup \{\uparrow\}$ repräsentieren:

$$\hat{f}(w) = \begin{cases} str(n), & w = str_k(n_1,\dots,n_k) \text{ und } f(n_1,\dots,n_k) = n \in \mathbb{N} \\ \uparrow, & \text{sonst} \end{cases}$$

- Wir nennen $\hat{f}$ die String-Repräsentation von f und f die numerische Repräsentation von $\hat{f}$

Beispiel

- Die Fkt. $f : (x, y) \mapsto x + y$ wird durch folgende Wortfkt. repräsentiert:

$$\hat{f}(w) = \begin{cases} str(x+y), & w = str_2(x,y) \\ \uparrow, & \text{sonst} \end{cases}$$

- Da f eine totale Funktion ist, gilt

$$\hat{f}(w) \neq \uparrow \;\Leftrightarrow\; w \in img(str_2),$$

also genau dann, wenn w die Form $x\#y$ mit $x, y \in \{0,1\}^*$ hat

- Beispielsweise ist $\hat{f}(w)$ für folgende Wörter w definiert:

w	#	0#	1#	#0	#1	00#	01#	0#0	0#1	10#	$\cdots$
(x, y)	(0,0)	(1,0)	(2,0)	(0,1)	(0,2)	(3,0)	(4,0)	(1,1)	(1,2)	(5,0)	$\cdots$
$x + y$	0	1	2	1	2	3	4	2	3	5	$\cdots$
$\hat{f}(w)$	ε	0	1	0	1	00	01	1	00	10	$\cdots$

◁

Äquivalenz von Turing- und GOTO-Berechenbarkeit

Satz

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ ist genau dann GOTO-berechenbar, wenn ihre String-Repräsentation $\hat{f}$ Turing-berechenbar ist.

Beweis

- Sei P ein GOTO-Programm, das eine partielle Fkt. $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ auf einer RAM R berechnet
- Dann existiert eine Zahl m, so dass P nur Register r_i mit $i \leq m$ benutzt
- Daher lässt sich eine Konfiguration von R durch Angabe der Inhalte des Befehlszählers IC und der Register $r_0, \ldots, r_m$ beschreiben
- Wir konstruieren eine $(m+2)$-DTM M, die
  - den Inhalt von IC in ihrem Zustand,
  - die Registerwerte $r_1, \ldots, r_m$ auf den Bändern $1, \ldots, m$ und
  - den Wert von r_0 auf dem Ausgabeband $m+2$ speichert
- Ein Registerwert r_i wird hierbei in der Form $str(r_i)$ gespeichert
- Band $m+1$ wird zur Ausführung von Hilfsberechnungen benutzt

Simulation eines GOTO-Programms durch eine DTM

- Die Aufgabe von M ist es, bei Eingabe $w \in \{0,1,\#\}^*$ das Wort $str(f(n_1,\ldots,n_k))$ auszugeben, wenn $w = str_k(n_1,\ldots,n_k)$ für ein Tupel $(n_1,\ldots,n_k) \in dom(f)$ ist, und andernfalls nicht zu halten
- Zuerst überprüft M, ob in w das $\#$-Zeichen $(k-1)$-mal vorkommt
- Dann kopiert M die Teilwörter $str(n_i)$ für $i = 2,\ldots,k$ auf das i-te Band und löscht auf dem 1. Band alle Eingabezeichen bis auf $str(n_1)$
- Für $i = 1,\ldots,m$ sind nun auf Band i die Registerinhalte $r_i = n_i$ und auf Band $m+2$ der Wert $r_0 = 0$ gespeichert
- Danach führt M das Programm P Befehl für Befehl aus
- Es ist klar, dass M jeden Befehl I in P durch eine geeignete Folge von Anweisungen simulieren kann, die die Registerinhalte und den Wert von IC entsprechend modifizieren
- Sobald P stoppt, hält auch M und gibt das auf Band $m+2$ befindliche Wort $str(r_0) = str(f(n_1,\ldots,n_k)) = \hat{f}(w)$ aus

Simulation einer DTM durch ein GOTO-Programm

- Sei $M = (Z, \Sigma, \Gamma, \delta, q_0, E)$ eine DTM,die die String-Repräsentation $\hat{f}$ einer partiellen Funktion $f : \mathbb{N}^k \to \mathbb{N} \cup \{\uparrow\}$ berechnet
- M gibt also bei Eingabe w das Wort $str(f(n_1, \ldots, n_k))$ aus, falls w die Form $w = str_k(n_1, \ldots, n_k)$ hat und $f(n_1, \ldots, n_k)$ definiert ist, und hält andernfalls nicht
- Wir konstruieren ein GOTO-Programm P, das bei Eingabe $(n_1, \ldots, n_k)$ die DTM M bei Eingabe $w = str_k(n_1, \ldots, n_k)$ simuliert
- Wir können annehmen, dass M eine 1-DTM ist
- Sei $Z = \{q_0, \ldots, q_r\}$ und $\Gamma = \{a_0, \ldots, a_{m-1}\}$, wobei wir annehmen, dass $a_0 = \sqcup$, $a_1 = 0$, $a_2 = 1$ und $a_3 = \#$ ist
- Eine Konfiguration $K = uq_iv$ von M mit $u = a_{i_1} \ldots a_{i_s}$ und $v = a_{j_1} \ldots a_{j_t}$ wird wie folgt in den Registern r_0, r_1, r_2 gespeichert:
  - $r_0 = (i_1 \ldots i_s)_m$
  - $r_1 = i$
  - $r_2 = (j_t \ldots j_1)_m$

Simulation einer DTM durch ein GOTO-Programm

- Eine Konfiguration $K = uq_iv$ von M mit $u = a_{i_1} \dots a_{i_s}$ und $v = a_{j_1} \dots a_{j_t}$ wird wie folgt in den Registern r_0, r_1, r_2 gespeichert:
  - $r_0 = (i_1 \dots i_s)_m$
  - $r_1 = i$
  - $r_2 = (j_t \dots j_1)_m$
- P besteht aus 3 Programmteilen $P = P_1, P_2, P_3$:
  - P_1 stellt in den drei Registern r_0, r_1, r_2 die Startkonfiguration $K_w = q_0 w$ von M bei Eingabe $w = str_k(n_1, \dots, n_k)$ her, d.h. P_1 berechnet in Register r_2 die Zahl $(j_t \dots j_1)_m$, wobei $w = a_{j_1} \dots a_{j_t}$ ist, und setzt r_0 und r_1 auf den Wert 0
  - P_2 überführt die in r_0, r_1, r_2 gespeicherte Konfiguration von M solange in die zugehörige Nachfolgekonfiguration bis M hält (siehe nächste Folie)
  - Danach transformiert P_3 noch den aktuellen Inhalt $(i_1 \dots i_s)_m$ von Register r_0 in die Zahl $num(a_{i_1} \dots a_{i_s})$ und hält

Simulation einer DTM durch ein GOTO-Programm

- Das Programmstück P_2 hat die Form

$$\begin{array}{ll} M_2 & r_3 := r_2 \textbf{ MOD } m \\ & \textbf{IF } r_1 = 0 \wedge r_3 = 0 \textbf{ THEN GOTO } M_{0,0} \\ & \quad \vdots \\ & \textbf{IF } r_1 = r \wedge r_3 = m-1 \textbf{ THEN GOTO } M_{r,m-1} \end{array}$$

- Die Befehle ab Position $M_{i,j}$ hängen von $\delta(q_i, a_j)$ ab:
  - Im Fall $\delta(q_i, a_j) = \varnothing$ markiert $M_{i,j}$ den Beginn von P_3
  - Im Fall $\delta(q_i, a_j) = \{(q_{i'}, a_{j'}, L)\}$ werden folgende Befehle ausgeführt:

$$\begin{array}{lll} M_{i,j} & r_1 := i' & r_2 := r_2 m + (r_0 \textbf{ MOD } m) \\ & r_2 := r_2 \textbf{ DIV } m & r_0 := r_0 \textbf{ DIV } m \\ & r_2 := r_2 m + j' & \textbf{GOTO } M_2 \end{array}$$

  - Die übrigen Fälle sind ähnlich

- Makros wie die **MOD**- und **DIV**- Befehle können durch entsprechende GOTO-Programmstücke ersetzt werden □

Zeitkomplexität von Turingmaschinen

Die Laufzeit einer NTM M bei Eingabe x ist die maximale Anzahl an Rechenschritten, die $M(x)$ ausführt

Definition

- Die Laufzeit einer NTM M bei Eingabe x ist definiert als

$$time_M(x) = \sup\{t \geq 0 \mid \exists K : K_x \vdash^t K\},$$

  wobei $\sup \mathbb{N} = \infty$ ist
- Sei $t : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion
- Dann ist M $t(n)$-zeitbeschränkt, falls für alle Eingaben x gilt:

$$time_M(x) \leq t(|x|)$$

Die Zeitschranke $t(n)$ beschränkt also die Laufzeit bei allen Eingaben der Länge n (worst-case Komplexität)

Zeitkomplexitätsklassen

Wir fassen alle Sprachen und Funktionen, die in einer vorgegebenen Zeitschranke $t(n)$ entscheidbar bzw. berechenbar sind, in folgenden Komplexitätsklassen zusammen

Definition

- Die in deterministischer Zeit $t(n)$ entscheidbaren Sprachen bilden die Sprachklasse

$$\mathsf{DTIME}(t(n)) = \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte DTM}\}$$

- Die in nichtdeterministischer Zeit $t(n)$ entscheidbaren Sprachen bilden die Sprachklasse

$$\mathsf{NTIME}(t(n)) = \{L(M) \mid M \text{ ist eine } t(n)\text{-zeitbeschränkte NTM}\}$$

- Die in deterministischer Zeit $t(n)$ berechenbaren Funktionen bilden die Funktionenklasse

$$\mathsf{FTIME}(t(n)) = \left\{ f \;\middle|\; \begin{array}{l} \text{es gibt eine } t(n)\text{-zeitbeschränkte} \\ \text{DTM } M \text{, die } f \text{ berechnet} \end{array} \right\}$$

Die wichtigsten Zeitkomplexitätsklassen

- Die wichtigsten deterministischen Zeitkomplexitätsklassen sind

$$
\begin{aligned}
\text{LINTIME} &= \bigcup_{c\geq 1} \text{DTIME}(cn+c) && \text{„Linearzeit“}\\
\text{P} &= \bigcup_{c\geq 1} \text{DTIME}(n^c+c) && \text{„Polynomialzeit“}\\
\text{E} &= \bigcup_{c\geq 1} \text{DTIME}(2^{cn+c}) && \text{„Lineare Exponentialzeit“}\\
\text{EXP} &= \bigcup_{c\geq 1} \text{DTIME}(2^{n^c+c}) && \text{„Exponentialzeit“}
\end{aligned}
$$

- Die nichtdeterministischen Klassen NLINTIME, NP, NE, NEXP und die Funktionenklassen FLINTIME, FP, FE, FEXP sind analog definiert
- Für eine Klasse $\mathcal{F}$ von Funktionen sei DTIME($\mathcal{F}$) $= \bigcup_{t\in\mathcal{F}} \text{DTIME}(t(n))$ (die Klassen NTIME($\mathcal{F}$) und FTIME($\mathcal{F}$) sind analog definiert)

Asymptotische Laufzeit und Landau-Notation

Definition

Seien f und g Funktionen von $\mathbb{N}$ nach $\mathbb{R}^+ \cup \{0\} = [0, \infty)$

- Wir schreiben $f(n) = \mathcal{O}(g(n))$, falls es Zahlen n_0 und c gibt mit

  $$\forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

  Bedeutung: „f wächst nicht wesentlich schneller als g“

- Formal bezeichnet der Term $\mathcal{O}(g(n))$ die Klasse aller Funktionen f, die obige Bedingung erfüllen, d.h.

  $$\mathcal{O}(g(n)) = \{f\colon \mathbb{N} \to [0, \infty) \mid \exists n_0, c \in \mathbb{N}\, \forall n \geq n_0 \colon f(n) \leq c \cdot g(n)\}$$

- Die Gleichung $f(n) = \mathcal{O}(g(n))$ drückt also in Wahrheit eine Element-Beziehung $f \in \mathcal{O}(g(n))$ aus
- O-Terme können auch auf der linken Seite vorkommen. In diesem Fall wird eine Inklusionsbeziehung ausgedrückt
- So steht $n^2 + \mathcal{O}(n) = \mathcal{O}(n^2)$ für die Aussage

  $$\{n^2 + f \mid f \in \mathcal{O}(n)\} \subseteq \mathcal{O}(n^2)$$

Asymptotische Laufzeit und Landau-Notation

Beispiel

- $7\log(n) + n^3 = \mathcal{O}(n^3)$ ist richtig
- $7\log(n)n^3 = \mathcal{O}(n^3)$ ist falsch
- $2^{n+\mathcal{O}(1)} = \mathcal{O}(2^n)$ ist richtig
- $2^{\mathcal{O}(n)} = \mathcal{O}(2^n)$ ist falsch (siehe Übungen)

Mit der O-Notation lassen sich die wichtigsten deterministischen Zeitkomplexitätsklassen wie folgt charakterisieren:

$$\begin{aligned}
\mathsf{LINTIME} &= \mathsf{DTIME}(\mathcal{O}(n)) && \text{„Linearzeit“} \\
\mathsf{P} &= \mathsf{DTIME}(n^{\mathcal{O}(1)}) && \text{„Polynomialzeit“} \\
\mathsf{E} &= \mathsf{DTIME}(2^{\mathcal{O}(n)}) && \text{„Lineare Exponentialzeit“} \\
\mathsf{EXP} &= \mathsf{DTIME}(2^{n^{\mathcal{O}(1)}}) && \text{„Exponentialzeit“}
\end{aligned}$$

Das P-NP-Problem

- Wie wir gesehen haben, sind NTMs nicht mächtiger als DTMs, d.h. jede NTM kann von einer DTM simuliert werden
- Die Frage, wieviel Zeit eine DTM zur Simulation einer NTM benötigt, ist eines der wichtigsten offenen Probleme der Informatik
- Wegen $\mathsf{NTIME}(t) \subseteq \mathsf{DTIME}(2^{\mathcal{O}(t)})$ erhöht sich die Laufzeit im schlimmsten Fall exponentiell
- Insbesondere die Klasse NP enthält viele für die Praxis überaus wichtige Probleme, für die kein Polynomialzeitalgorithmus bekannt ist
- Für viele dieser Probleme A konnte folgende Implikation gezeigt werden:

  $$A \in \mathsf{P} \Rightarrow \mathsf{P} = \mathsf{NP}$$

- Da jedoch nur Probleme in P als effizient lösbar angesehen werden, hat das P-NP-Problem, also die Frage, ob NP = P ist, eine immense praktische Bedeutung

Die Polynomialzeitreduktion

Definition

- Eine Sprache $A \subseteq \Sigma^*$ ist auf $B \subseteq \Gamma^*$ in Polynomialzeit reduzierbar ($A \leq^p B$), falls eine Funktion $f : \Sigma^* \to \Gamma^*$ in FP existiert mit

$$\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B$$

- Eine Sprache A heißt $\leq^p$-hart für eine Sprachklasse $\mathcal{C}$ (kurz: $\mathcal{C}$-hart oder $\mathcal{C}$-schwer), falls gilt:

$$\forall L \in \mathcal{C} : L \leq^p A$$

- Eine $\mathcal{C}$-harte Sprache A, die zu $\mathcal{C}$ gehört, heißt $\mathcal{C}$-vollständig (bzgl. $\leq^p$)
- NPC bezeichnet die Klasse aller NP-vollständigen Sprachen

Lemma

- Aus $A \leq^p B$ folgt $A \leq B$
- Die Reduktionsrelation $\leq^p$ ist reflexiv und transitiv (s. Übungen)

Die Polynomialzeitreduktion

Satz

Die Klassen P und NP sind unter $\leq^p$ abgeschlossen

Beweis

- Sei $B \in$ P und gelte $A \leq^p B$ mittels einer Funktion $f \in$ FP
- Seien M und T DTMs mit $L(M) = B$ und $T(x) = f(x)$
- Weiter seien p und q polynomielle Zeitschranken für M und T
- Betrachte die DTM M', die bei Eingabe x zuerst T simuliert, um $f(x)$ zu berechnen, und danach M bei Eingabe $f(x)$ simuliert. Dann gilt

$$x \in A \Leftrightarrow f(x) \in B \Leftrightarrow f(x) \in L(M) \Leftrightarrow x \in L(M')$$

- Also ist $L(M') = A$ und wegen

$$time_{M'}(x) \leq time_T(x) + time_M(f(x)) \leq q(|x|) + p(q(|x|))$$

  ist M' polynomiell zeitbeschränkt und somit A in P
- Die Abgeschlossenheit von NP unter $\leq^p$ folgt analog $\square$

Satz

$A \leq^p B$ und A ist NP-hart $\Rightarrow$ B ist NP-hart

Beweis

- Sei $L \in \mathrm{NP}$
- Da A NP-hart ist, gilt $L \leq^p A$
- Da zudem $A \leq^p B$ gilt und $\leq^p$ transitiv ist, folgt $L \leq^p B$

Satz

Falls ein NP-hartes Problem A in P enthalten ist, folgt $\mathrm{P} = \mathrm{NP}$

Beweis

- Sei $L \in \mathrm{NP}$
- Da A NP-hart ist, gilt $L \leq^p A$
- Da $A \in \mathrm{P}$ ist und P unter $\leq^p$ abgeschlossen ist, folgt $L \in \mathrm{P}$ $\square$

Platzkomplexität von Turingmaschinen

- Als nächstes definieren wir den Platzverbrauch von NTMs
- Intuitiv ist dies die Anzahl aller von einer NTM M besuchten Bandfelder
- Wollen wir auch sublinearen Platz sinnvoll definieren, so dürfen wir die Bandfelder, auf denen die Eingabe steht, nicht mitzählen
- Um sicherzustellen, dass M das erste Band nur zum Lesen der Eingabe und nicht als Speicherplatz benutzt, darf M
  - die Felder auf dem Eingabeband nicht verändern und
  - sich höchstens ein Feld von der Eingabe entfernen

Definition

Eine k-NTM M heißt NTM mit Eingabeband (kurz offline-NTM), falls für jede von M bei Eingabe x erreichbare Konfiguration

$$K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k)$$

gilt, dass $u_1 a_1 v_1$ ein Teilwort von $\sqcup x \sqcup$ ist

Definition

- Der Platzverbrauch einer offline-NTM M bei Eingabe x ist definiert als

$$space_M(x) = \sup\left\{ s \geq 1 \;\middle|\; \begin{array}{l} \exists K = (q, u_1, a_1, v_1, \ldots, u_k, a_k, v_k) \\ \text{mit } K_x \vdash^* K \text{ und } s = \sum_{i=2}^{k} |u_i a_i v_i| \end{array} \right\}$$

- Sei $s : \mathbb{N} \to \mathbb{N}$ eine monoton wachsende Funktion
- M heißt $s(n)$-platzbeschränkt, falls für alle Eingaben x gilt:

$$space_M(x) \leq s(|x|) \text{ und } time_M(x) < \infty$$

Wir fassen alle Sprachen, die in einer vorgegebenen Platzschranke $s(n)$ entscheidbar sind, in folgenden Platzkomplexitätsklassen zusammen

Definition

- Die auf deterministischem Platz $s(n)$ entscheidbaren Sprachen bilden die Klasse

$$\mathsf{DSPACE}(s(n)) = \{L(M) \mid M \text{ ist eine } s(n)\text{-platzb. offline-DTM}\}$$

- Die auf nichtdeterministischem Platz $s(n)$ entscheidbaren Sprachen bilden die Klasse

$$\mathsf{NSPACE}(s(n)) = \{L(M) \mid M \text{ ist eine } s(n)\text{-platzb. offline-NTM}\}$$

Die wichtigsten Platzkomplexitätsklassen

- Die wichtigsten deterministischen Platzkomplexitätsklassen sind

$$\begin{aligned}
\mathsf{L} &= \mathsf{DSPACE}(\mathcal{O}(\log n)) && \text{„Logarithmischer Platz“} \\
\mathsf{LINSPACE} &= \mathsf{DSPACE}(\mathcal{O}(n)) && \text{„Linearer Platz“} \\
\mathsf{PSPACE} &= \mathsf{DSPACE}(n^{\mathcal{O}(1)}) && \text{„Polynomieller Platz“} \\
\mathsf{EXPSPACE} &= \mathsf{DSPACE}(2^{n^{\mathcal{O}(1)}}) && \text{„Exponentieller Platz“}
\end{aligned}$$

- Die nichtdeterministischen Klassen NL, NLINSPACE, NPSPACE und NEXPSPACE sind analog definiert

Elementare Beziehungen zwischen Komplexitätsklassen

Frage

Welche elementaren Beziehungen gelten zwischen den verschiedenen Zeit- und Platzklassen?

Satz

- Für jede Funktion $t(n) \geq n + 2$ gilt

$$\mathrm{DTIME}(t) \subseteq \mathrm{NTIME}(t) \subseteq \mathrm{DSPACE}(\mathcal{O}(t))$$

- Für jede Funktion $s(n) \geq \log n$ gilt

$$\mathrm{DSPACE}(s) \subseteq \mathrm{NSPACE}(s) \subseteq \mathrm{DTIME}(2^{\mathcal{O}(s)}) \text{ und}$$

$$\mathrm{NSPACE}(s) \subseteq \mathrm{DSPACE}(s^2) \qquad \text{(Satz von Savitch)}$$

Korollar

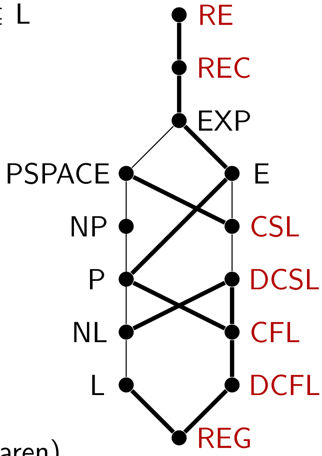
- $\mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{NPSPACE} \subseteq \mathrm{EXP} \subseteq \mathrm{NEXP} \subseteq \mathrm{EXPSPACE}$
- $\mathrm{PSPACE} = \mathrm{NPSPACE}$ und $\mathrm{EXPSPACE} = \mathrm{NEXPSPACE}$

Komplexität der Stufen der Chomsky-Hierarchie

- REG = DSPACE($\mathcal{O}(1)$) = NSPACE($\mathcal{O}(1)$) $\subsetneq$ L
- DCFL $\subsetneq$ LINTIME
- CFL $\subsetneq$ NLINTIME $\cap$ DTIME($\mathcal{O}(n^3)$) $\subsetneq$ P
- DCSL = LINSPACE $\subseteq$ CSL
- CSL = NLINSPACE $\subseteq$ PSPACE $\cap$ E
- REC = $\bigcup_f$ DTIME($f(n)$) = $\bigcup_f$ DSPACE($f(n)$)

  = $\bigcup_f$ NTIME($f(n)$) = $\bigcup_f$ NSPACE($f(n)$),

  wobei f alle (oder äquivalent: alle berechenbaren) Funktionen $f : \mathbb{N} \to \mathbb{N}$ durchläuft



Aussagenlogische Formeln

- Die Menge der booleschen (oder aussagenlogischen) Formeln über den Variablen $x_1, \ldots, x_n$, $n \geq 0$, ist induktiv wie folgt definiert:
  - Die Konstanten 0 und 1 sind boolesche Formeln
  - Jede Variable x_i ist eine boolesche Formel
  - Mit G und H sind auch die Konjunktion $(G \wedge H)$ und die Disjunktion $(G \vee H)$ von G und H sowie die Negation $\neg G$ von G Formeln
- Eine Belegung von $x_1, \ldots, x_n$ ist ein Wort $a = a_1 \ldots a_n \in \{0,1\}^n$
- Der Wert $F(a)$ von F unter a ist induktiv wie folgt definiert:

F	0	1	x_i	$\neg G$	$(G \wedge H)$	$(G \vee H)$
$F(a)$	0	1	a_i	$1 - G(a)$	$G(a)H(a)$	$G(a) + H(a) - G(a)H(a)$

- Durch die Formel F wird also eine n-stellige boolesche Funktion $F : \{0,1\}^n \to \{0,1\}$ definiert, die wir ebenfalls mit F bezeichnen

Notation

Wir benutzen die Implikation $G \to H$ als Abkürzung für die Formel $\neg G \vee H$ und die Äquivalenz $G \leftrightarrow H$ als Abkürzung für $(G \to H) \wedge (H \to G)$

Beispiel (Wahrheitswerttabelle)

Die Formel $F = (G \to H)$ mit den Teilformeln

- $G = (x_2 \wedge x_3)$ und
- $H = (\neg x_1 \vee \neg x_2)$

berechnet nebenstehende boolesche Funktion $F : \{0,1\}^3 \to \{0,1\}$.

a	$G(a)$	$H(a)$	$F(a)$
000	0	1	1
001	0	1	1
010	0	1	1
011	1	1	1
100	0	1	1
101	0	1	1
110	0	0	1
111	1	0	0

◁

Aussagenlogische Formeln

Definition

- Zwei Formeln F und G heißen (logisch) äquivalent (kurz $F \equiv G$), wenn sie dieselbe boolesche Funktion berechnen
- Eine Formel F heißt erfüllbar, falls es eine Belegung a mit $F(a) = 1$ gibt
- Gilt sogar für alle Belegungen a, dass $F(a) = 1$ ist, so heißt F Tautologie

Beispiel

- Die Formel $F = (G \to H)$ mit $G = (x_2 \wedge x_3)$ und $H = (\neg x_1 \vee \neg x_2)$ ist erfüllbar, da $F(000) = 1$ ist
- F ist aber keine Tautologie, da $F(111) = 0$ ist $\triangleleft$

Aussagenlogische Formeln

Präzedenzregeln zur Klammerersparnis

- Der Junktor $\wedge$ bindet stärker als der Junktor $\vee$ und dieser wiederum stärker als die Junktoren $\rightarrow$ und $\leftrightarrow$
- Formeln der Form $(F_1 \circ (F_2 \circ (F_3 \circ \cdots \circ F_n) \cdots)))$, $\circ \in \{\wedge, \vee\}$, kürzen wir durch $(F_1 \circ \cdots \circ F_n)$ ab und schreiben dafür auch $\bigwedge_{1 \le i \le n} F_i$ bzw. $\bigvee_{1 \le i \le n} F_i$

Beispiel (Formel für die mehrstellige Entweder-Oder Funktion)

- Folgende Formel nimmt unter einer Belegung $a = a_1 \ldots a_n$ genau dann den Wert 1 an, wenn $\sum_{i=1}^{n} a_i = 1$ ist:

$$G(x_1, \ldots, x_n) = (x_1 \vee \cdots \vee x_n) \wedge \bigwedge_{1 \le i < j \le n} \neg(x_i \wedge x_j)$$

- D.h. es gilt genau dann $G(a) = 1$, wenn genau eine Variable x_i mit dem Wert $a_i = 1$ belegt ist
- Diese Formel wird im Beweis des nächsten Satzes benötigt $\triangleleft$

Das aussagenlogische Erfüllbarkeitsproblem

Erfüllbarkeitsproblem für boolesche Formeln (*satisfiability*, SAT):

Gegeben: Eine boolesche Formel F

Gefragt: Ist F erfüllbar?

- Dabei kodieren wir boolesche Formeln F durch Binärstrings w_F und ordnen umgekehrt jedem Binärstring w eine Formel F_w zu
- Um die Notation zu vereinfachen, werden wir zukünftig jedoch F anstelle von w_F schreiben

Satz (Cook, Karp, Levin)

SAT ist NP-vollständig

SAT ist NP-vollständig

$\text{SAT} \in \text{NP}$

Eine NTM kann bei Eingabe einer booleschen Formel F zunächst eine Belegung a nichtdeterministisch raten und dann in Polynomialzeit testen, ob $F(a) = 1$ ist (*guess and verify* Strategie)

SAT ist NP-hart

- Sei L eine beliebige NP-Sprache und sei $M = (Z, \Sigma, \Gamma, \delta, q_0)$ eine durch ein Polynom p zeitbeschränkte k-NTM mit $L(M) = L$
- Da sich jede $t(n)$-zeitbeschränkte k-NTM in Zeit $O(t^2(n))$ durch eine 1-NTM simulieren lässt, können wir $k = 1$ annehmen
- Zudem können wir $Z = \{q_0, \dots, q_m\}$, $E = \{q_m\}$ und $\Gamma = \{a_1, \dots, a_l\}$ sowie $\delta(q_m, a) = \{(q_m, a, N)\}$ für alle $a \in \Gamma$ annehmen
- Unsere Aufgabe besteht nun darin, zu jedem Wort $w = w_1 \dots w_n \in \Sigma^*$ eine Formel F_w mit folgenden Eigenschaften zu konstruieren:
  - $w \in L(M) \iff F_w \in \text{SAT}$
  - die Reduktionsfunktion $w \mapsto F_w$ ist in FP berechenbar

SAT ist NP-hart

Idee:

Konstruiere F_w so, dass F_w unter einer Belegung a genau dann wahr wird, wenn a eine akzeptierende Rechnung von $M(w)$ beschreibt

- Wir bilden F_w über den Variablen

$$\begin{array}{ll} x_{t,i}, & \text{für } 0 \le t \le p(n), 0 \le i \le m \\ y_{t,j}, & \text{für } 0 \le t \le p(n), -p(n) \le j \le p(n) \\ z_{t,j,a}, & \text{für } 0 \le t \le p(n), -p(n) \le j \le p(n), a \in \Gamma \end{array}$$

- Diese Variablen stehen für folgende Aussagen:

$$\begin{array}{rl} x_{t,i}: & \text{zum Zeitpunkt } t \text{ befindet sich } M \text{ im Zustand } q_i \\ y_{t,j}: & \text{zur Zeit } t \text{ besucht } M \text{ das Feld mit der Nummer } j \\ z_{t,j,a}: & \text{zur Zeit } t \text{ steht das Zeichen } a \text{ auf dem Feld mit der Nr. } j \end{array}$$

- Konkret sei $F_w = R \wedge S_w \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$

- Konkret sei $F_w = R \wedge S_w \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$
- Dabei stellt die Formel $R = \bigwedge_{t=0}^{p(n)} R_t$ (Randbedingungen) sicher, dass wir jeder erfüllenden Belegung von F_w eindeutig eine Folge von Konfigurationen $K_0, \ldots, K_{p(n)}$ zuordnen können:

$$
\begin{aligned}
R_t = \; & G(x_{t,0}, \ldots, x_{t,m}) \wedge G(y_{t,-p(n)}, \ldots, y_{t,p(n)}) \\
& \wedge \bigwedge_{j=-p(n)}^{p(n)} G(z_{t,j,a_1}, \ldots, z_{t,j,a_l})
\end{aligned}
$$

- Die Teilformel R_t sorgt also dafür, dass zum Zeitpunkt t
  - genau ein Zustand $q_i \in Z$ eingenommen wird
  - genau ein Bandfeld $j \in \{-p(n), \ldots, p(n)\}$ besucht wird und
  - auf jedem Feld j genau ein Zeichen $a_k \in \Gamma = \{a_1, \ldots, a_l\}$ steht

SAT ist NP-hart

- Die Formel S_w (wie Startbedingung) stellt sicher, dass zum Zeitpunkt 0 tatsächlich die Startkonfiguration vorliegt:

$-p(n)$		-1	0		$n-1$	n		$p(n)$
$\sqcup$	$\dots$	$\sqcup$	w_1	$\dots$	w_n	$\sqcup$	$\dots$	$\sqcup$
			$\uparrow$					
			q_0					

$$S_w = x_{0,0} \wedge y_{0,0} \wedge \bigwedge_{j=-p(n)}^{-1} z_{0,j,\sqcup} \wedge \bigwedge_{j=0}^{n-1} z_{0,j,w_{j+1}} \wedge \bigwedge_{j=n}^{p(n)} z_{0,j,\sqcup}$$

- Die Formel $\ddot{U}_1$ sorgt dafür, dass der Inhalt von nicht besuchten Feldern beim Übergang von K_t zu K_{t+1} unverändert bleibt:

$$\ddot{U}_1 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{j=-p(n)}^{p(n)} \bigwedge_{a\in\Gamma} \left(\neg y_{t,j} \wedge z_{t,j,a} \to z_{t+1,j,a}\right)$$

- $\ddot{U}_2$ achtet darauf, dass sich bei jedem Rechenschritt der Zustand, die Kopfposition und das gerade gelesene Zeichen gemäß einer Anweisung in δ verändern:

$$\ddot{U}_2 = \bigwedge_{t=0}^{p(n)-1} \bigwedge_{j=-p(n)}^{p(n)} \bigwedge_{a \in \Gamma} \bigwedge_{i=0}^{m} \Big(x_{t,i} \wedge y_{t,j} \wedge z_{t,j,a} \rightarrow \bigvee_{(q_{i'},a',D) \in \delta(q_i,a)} x_{t+1,i'} \wedge y_{t+1,j+D} \wedge z_{t+1,j,a'} \Big),$$

wobei

$$j + D = \begin{cases} j-1, & D = L \\ j, & D = N \\ j+1, & D = R \end{cases}$$

- Schließlich überprüft E, ob $M(w)$ nach (spätestens) $p(n)$ Schritten den Endzustand q_m erreicht hat:

$$E = x_{p(n),m}$$

SAT ist NP-hart

- Da der Aufbau der Formel $f(w) = F_w$ einem einfachen Bildungsgesetz folgt und ihre Länge polynomiell in n ist, folgt $f \in \mathrm{FP}$
- Es ist klar, dass F_w im Fall $w \in L(M)$ erfüllbar ist, indem wir die Variablen von F_w gemäß einer akz. Rechnung von $M(w)$ belegen
- Umgekehrt führt eine Belegung a mit $F_w(a) = 1$ wegen $R(a) = 1$ eindeutig auf eine Konfigurationenfolge $K_0, \ldots, K_{p(n)}$
- Für diese gilt:
  - K_0 ist Startkonfiguration von $M(w)$ (wegen $S_w(a) = 1$)
  - $K_i \vdash K_{i+1}$ für $i = 0, \ldots, p(n) - 1$ (wegen $\ddot{U}_1(a) = \ddot{U}_2(a) = 1$)
  - der Zustand von $K_{p(n)}$ ist m (wegen $E(a) = 1$)
- Also gilt für alle $w \in \Sigma^*$ die Äquivalenz

$$w \in L(M) \Leftrightarrow F_w \in \textsc{Sat},$$

d.h. die FP-Funktion $f : w \mapsto F_w$ reduziert $L(M)$ auf SAT $\square$

Boolesche Schaltkreise

Als nächstes betrachten wir das Erfüllbarkeitsproblem für Schaltkreise

Definition

- Ein boolescher Schaltkreis über den Variablen $x_1, \ldots, x_n$ ist eine Folge $S = (g_1, \ldots, g_m)$ von Gattern

$$g_l \in \{0, 1, x_1, \ldots, x_n, (\neg, j), (\wedge, j, k), (\vee, j, k)\} \text{ mit } 1 \leq j, k < l$$

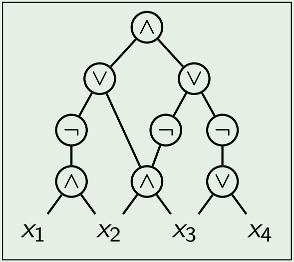
- Jedes Gatter g_l berechnet eine n-stellige boolesche Funktion $g_l\colon \{0,1\}^n \to \{0,1\}$
- Für $a = a_1 \ldots a_n \in \{0,1\}^n$ ist $g_l(a)$ induktiv wie folgt definiert:

g_l	0	1	x_i	$(\neg, j)$	$(\wedge, j, k)$	$(\vee, j, k)$
$g_l(a)$	0	1	a_i	$1 - g_j(a)$	$g_j(a)g_k(a)$	$g_j(a) + g_k(a) - g_j(a)g_k(a)$

- S berechnet die boolesche Funktion $S(a) = g_m(a)$
- S heißt erfüllbar, wenn eine Eingabe $a \in \{0,1\}^n$ ex. mit $S(a) = 1$

Boolesche Schaltkreise

Beispiel



Graphische Darstellung des Schaltkreises $S = (x_1, x_2, x_3, x_4, (\wedge, 1, 2), (\wedge, 2, 3), (\vee, 3, 4), (\neg, 5), (\neg, 6), (\neg, 7), (\vee, 6, 8), (\vee, 9, 10), (\wedge, 11, 12))$.

Bemerkung

- Die Anzahl der Eingänge eines Gatters g wird als Fanin von g bezeichnet
- die Anzahl der Ausgänge von g (also die Anzahl der Gatter, die g als Eingabe benutzen) als Fanout
- Boolesche Formeln entsprechen also genau den booleschen Schaltkreisen $S = (g_1, \ldots, g_m)$, bei denen jedes Gatter g_i, $1 \le i \le m - 1$, Fanout 1 hat
- Eine boolesche Formel F kann somit leicht in einen äquivalenten Schaltkreis S mit $S(a) = F(a)$ für alle Belegungen a transformiert werden

Das Erfüllbarkeitsproblem für boolesche Schaltkreise

Erfüllbarkeitsproblem für boolesche Schaltkreise ($\textsc{CirSat}$):

Gegeben: Ein boolescher Schaltkreis S
Gefragt: Ist S erfüllbar?

Satz

$\textsc{CirSat}$ ist NP-vollständig

Beweis

Klar, da $\textsc{Sat} \leq^p \textsc{CirSat}$ und $\textsc{CirSat} \in \mathrm{NP}$ gilt. □

Bemerkung

- Da $\textsc{Sat}$ NP-vollständig ist, ist auch $\textsc{CirSat}$ auf $\textsc{Sat}$ reduzierbar
- Dies bedeutet, dass sich jeder Schaltkreis S in Polynomialzeit in eine erfüllbarkeitsäquivalente Formel F_S überführen lässt
- F_S und S müssen aber nicht logisch äquivalent sein
- $\textsc{CirSat}$ ist sogar auf eine spezielle $\textsc{Sat}$-Variante reduzierbar

Formeln in konjunktiver Normalform (KNF)

- Ein Literal ist eine Variable x_i oder eine negierte Variable $\neg x_i$, die wir auch kurz mit $\bar{x}_i$ bezeichnen
- Eine Klausel ist eine Disjunktion $C = \bigvee_{j=1}^{k} l_j$ von Literalen $l_1, \ldots, l_k$
- Hierbei ist auch $k = 0$ zulässig, d.h. die leere Klausel repräsentiert die Konstante 0 und wird üblicherweise mit $\Box$ bezeichnet
- Eine boolesche Formel F ist in konjunktiver Normalform (kurz KNF), falls $F = \bigwedge_{j=1}^{m} C_j$ eine Konjunktion von Klauseln $C_1, \ldots, C_m$ ist
- Auch hier ist $m = 0$ zulässig, wobei die leere Konjunktion die Konstante 1 repräsentiert
- Enthält jede Klausel höchstens k Literale, so heißt F in k-KNF
- Klauseln werden oft als Menge $C = \{l_1, \ldots, l_k\}$ ihrer Literale und KNF-Formeln als Menge $F = \{C_1, \ldots, C_m\}$ ihrer Klauseln dargestellt
- Enthält F die leere Klausel, so ist F unerfüllbar
- Dagegen ist die leere Formel eine Tautologie

Erfüllbarkeitsprobleme für KNF-Formeln

Erfüllbarkeitsproblem für k-KNF Formeln (k-SAT):

Gegeben: Eine boolesche Formel F in k-KNF

Gefragt: Ist F erfüllbar?

Beispiel

- Der 3-KNF Formel $F = (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_3) \wedge (x_2 \vee \bar{x}_3 \vee x_4)$ entspricht die Klauselmenge $F = \{\{x_1, \bar{x}_2\}, \{\bar{x}_1, x_3\}, \{x_2, \bar{x}_3, x_4\}\}$
- Offenbar ist $F(0000) = 1$, d.h. $F \in$ 3-SAT

Satz

k-SAT ist für $k \leq 2$ in P entscheidbar und für $k \geq 3$ NP-vollständig

3-Sat ist NP-vollständig

Reduktion von CirSat auf 3-Sat

- Wir transformieren einen Schaltkreis $S = (g_1, \ldots, g_m)$ mit n Eingängen in eine Formel F_S über den Variablen $x_1, \ldots, x_n, y_1, \ldots, y_m$
- F_S enthält die Klausel $\{y_m\}$ und für jedes Gatter g_i die Klauseln einer 3-KNF F_i, die zu folgender Formel G_i äquivalent ist:

Gatter g_i	G_i	Klauseln von F_i
0	$y_i \leftrightarrow 0$	$\{\bar{y}_i\}$
1	$y_i \leftrightarrow 1$	$\{y_i\}$
x_j	$y_i \leftrightarrow x_j$	$\{\bar{y}_i, x_j\}, \{\bar{x}_j, y_i\}$
$(\neg, j)$	$y_i \leftrightarrow \bar{y}_j$	$\{\bar{y}_i, \bar{y}_j\}, \{y_j, y_i\}$
$(\wedge, j, k)$	$y_i \leftrightarrow y_j \wedge y_k$	$\{\bar{y}_i, y_j\}, \{\bar{y}_i, y_k\}, \{\bar{y}_j, \bar{y}_k, y_i\}$
$(\vee, j, k)$	$y_i \leftrightarrow y_j \vee y_k$	$\{\bar{y}_j, y_i\}, \{\bar{y}_k, y_i\}, \{\bar{y}_i, y_j, y_k\}$

3-SAT ist NP-vollständig

Reduktion von CIRSAT auf 3-SAT

- Wir zeigen, dass für alle $a \in \{0,1\}^n$ folgende Äquivalenz gilt:

  $$S(a) = 1 \Leftrightarrow \exists b \in \{0,1\}^m : F_S(ab) = 1$$

- Ist nämlich $a \in \{0,1\}^n$ eine Eingabe mit $S(a) = 1$, so erhalten wir mit

  $$b_i = g_i(a) \text{ für } i = 1, \ldots, m$$

  eine erfüllende Belegung $ab_1 \ldots b_m$ für F_S

- Ist umgekehrt $ab_1 \ldots b_m$ eine erfüllende Belegung für F_S, so muss
  - $b_m = 1$ sein, da $\{y_m\}$ eine Klausel in F_S ist, und
  - durch Induktion über $i = 1, \ldots, m$ folgt

    $$g_i(a) = b_i,$$

    d.h. insbesondere folgt $S(a) = g_m(a) = b_m = 1$

3-Sat ist NP-vollständig

Reduktion von CirSat auf 3-Sat

- Wir wissen bereits, dass für alle $a \in \{0,1\}^n$ die Äquivalenz

  $$S(a) = 1 \Leftrightarrow \exists b \in \{0,1\}^m : F_S(ab) = 1$$

  gilt
- Dies bedeutet, dass der Schaltkreis S und die 3-KNF-Formel F_S erfüllbarkeitsäquivalent sind, d.h.

  $$S \in \textsc{CirSat} \Leftrightarrow F_S \in 3\text{-}\textsc{Sat}$$

- Da zudem die Reduktionsfunktion $S \mapsto F_S$ in FP berechenbar ist, folgt $\textsc{CirSat} \leq^p 3\text{-}\textsc{Sat}$ □

Notation – ungerichtete Graphen

- Ein (ungerichteter) Graph ist ein Paar $G = (V, E)$, wobei

  V - eine endliche Menge von Knoten/Ecken und

  E - die Menge der Kanten ist
- Hierbei gilt

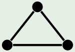
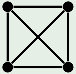
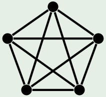
$$E \subseteq \binom{V}{2} := \big\{\{u, v\} \subseteq V \mid u \neq v\big\}$$

- Die Knotenzahl von G ist $n(G) = \|V\|$
- Die Kantenzahl von G ist $m(G) = \|E\|$
- Die Nachbarschaft von $v \in V$ ist $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$ und die Nachbarschaft von $U \subseteq V$ ist $N_G(U) = \bigcup_{u \in U} N_G(u)$
- Der Grad von $v \in V$ ist $\deg_G(v) = \|N_G(v)\|$
- Der Minimalgrad von G ist $\delta(G) := \min_{v \in V} \deg_G(v)$ und
- der Maximalgrad von G ist $\Delta(G) := \max_{v \in V} \deg_G(v)$

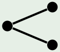
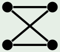
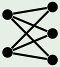
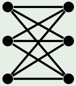
Falls G aus dem Kontext ersichtlich ist, schreiben wir auch einfach n, m, $N(v)$, $\deg(v)$, δ usw

Beispiel

- Der vollständige Graph (V, E) mit $\|V\| = n$ und $E = \binom{V}{2}$ wird mit K_n und der leere Graph $(V, \varnothing)$ wird mit E_n bezeichnet:

K_1:  K_2:  K_3:  K_4:  K_5: 

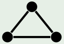
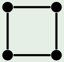
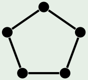
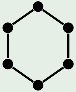
- Der vollständige bipartite Graph (A, B, E) auf $a + b$ Knoten, d.h. $A \cap B = \varnothing$, $\|A\| = a$, $\|B\| = b$ und $E = \{\{u, v\} \mid u \in A, v \in B\}$ wird mit $K_{a,b}$ bezeichnet:

$K_{1,1}$:  $K_{1,2}$:  $K_{2,2}$:  $K_{2,3}$:  $K_{3,3}$: 

- Der Pfadgraph (oder lineare Graph) mit n Knoten heißt P_n:

P_2:  P_3:  P_4:  P_5: 

- Der Kreisgraph (kurz Kreis) mit n Knoten heißt C_n:

C_3:  C_4:  C_5:  C_6: 

Notation – ungerichtete Graphen (Fortsetzung)

- Ein Graph $H = (V', E')$ heißt Sub-/Teil-/Untergraph von $G = (V, E)$, falls $V' \subseteq V$ und $E' \subseteq E$ ist
- Ein Weg ist eine Folge von Knoten $v_0, \ldots, v_j$ mit $\{v_i, v_{i+1}\} \in E$ für $i = 0, \ldots, j-1$
- Ein Weg heißt einfach oder Pfad, falls alle durchlaufenen Knoten verschieden sind
- Die Länge des Weges ist die Anzahl der Kanten, also j
- Ein Weg $v_0, \ldots, v_j$ heißt auch v_0-v_j-Weg
- Ein Zyklus ist ein u-v-Weg der Länge $j \geq 2$ mit $u = v$
- Ein Kreis ist ein Zyklus $v_0, v_1 \ldots, v_{j-1}, v_0$ der Länge $j \geq 3$, für den $v_0, v_1, \ldots, v_{j-1}$ paarweise verschieden sind

Cliquen, Stabilität und Matchings

- Eine Knotenmenge $U \subseteq V$ heißt Clique, wenn E alle Kanten enthält, die beide Endpunkte in U haben, d.h. es gilt $\binom{U}{2} \subseteq E$
- Die Cliquenzahl ist

$$\omega(G) = \max\{\|U\| \mid U \text{ ist Clique in } G\}$$

- Eine Knotenmenge $U \subseteq V$ heißt stabil oder unabhängig, wenn keine Kante in G beide Endpunkte in U hat, d.h. es gilt $E \cap \binom{U}{2} = \varnothing$
- Die Stabilitätszahl ist

$$\alpha(G) = \max\{\|U\| \mid U \text{ ist stabile Menge in } G\}$$

- Zwei Kanten $e, e' \in E$ heißen unabhängig, falls $e \cap e' = \varnothing$ ist
- Eine Kantenmenge $M \subseteq E$ heißt Matching in G, falls alle Kanten in M paarweise unabhängig sind
- Die Matchingzahl von G ist

$$\mu(G) = \max\{\|M\| \mid M \text{ ist ein Matching in } G\}$$

Knotenüberdeckungen und Färbungen

- Eine Knotenmenge $U \subseteq V$ heißt Knotenüberdeckung (engl. *vertex cover*), wenn jede Kante $e \in E$ mindestens einen Endpunkt in U hat, d.h. es gilt $e \cap U \neq \varnothing$ für alle Kanten $e \in E$
- Die Überdeckungszahl ist

$$\beta(G) = \min\{\|U\| \mid U \text{ ist eine Knotenüberdeckung in } G\}$$

- Eine Abbildung $f\colon V \to \mathbb{N}$ heißt Färbung von G, wenn $f(u) \neq f(v)$ für alle $\{u, v\} \in E$ gilt

  G heißt *k*-färbbar, falls eine Färbung $f\colon V \to \{1, \ldots, k\}$ existiert

  Die chromatische Zahl ist

$$\chi(G) = \min\{k \in \mathbb{N} \mid G \text{ ist } k\text{-färbbar}\}$$

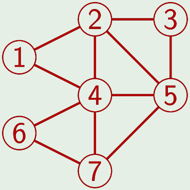
Eulerlinien und -touren

Definition

- Sei $s = (v_0, \ldots, v_l)$ ein Weg in einem Graphen $G = (V, E)$, der jede Kante genau einmal durchläuft, d.h. es gilt
$$\Big\{\{v_i, v_{i+1}\} \;\Big|\; i = 0, \ldots, l-1\Big\} = E \quad \text{und } l = \|E\|$$
- Dann heißt s im Fall $v_0 \neq v_l$ Eulerlinie (auch Eulerzug oder Eulerweg) von v_0 nach v_l in G
- Ist dagegen $v_0 = v_l$, d.h. s ist ein Zyklus, der jede Kante genau einmal durchläuft, so heißt s Eulerkreis (auch Eulerzyklus oder Eulertour)

Beispiel (Eulerlinie)

$s = (4, 1, 2, 3, 5, 7, 6, 4, 5, 2, 4, 7)$



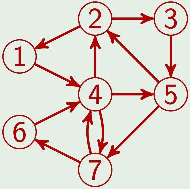
Gerichtete Eulerlinien

Definition

- Sei $s = (v_0, \ldots, v_l)$ ein gerichteter Weg in einem Digraphen $G = (V, E)$, der jede Kante genau einmal durchläuft, d.h. es gilt
$$\left\{ (v_i, v_{i+1}) \;\middle|\; i = 0, \ldots, l-1 \right\} = E \quad \text{und } l = \|E\|$$
- Dann heißt s im Fall $v_0 \neq v_l$ Eulerlinie (auch Eulerzug oder Eulerweg) von v_0 nach v_l in G
- Ist dagegen $v_l = v_0$, d.h. s ist ein Zyklus, der jede Kante genau einmal durchläuft, so heißt s Eulerkreis (auch Eulerzyklus oder Eulertour)

Beispiel (Eulerkreis in einem Digraphen)

$$s = (1, 4, 5, 2, 3, 5, 7, 4, 7, 6, 4, 2, 1)$$



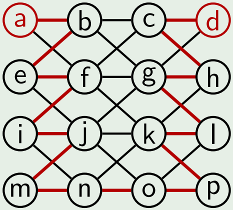
Hamiltonpfade und -kreise

Definition

- Sei $s = (v_0, v_1, \ldots, v_l)$ ein Pfad in einem Graphen $G = (V, E)$, d.h. $\{v_i, v_{i+1}\} \in E$ für $i = 0, \ldots, l-1$ und die Knoten $v_0, \ldots, v_l$ sind alle verschieden
- Dann heißt s Hamiltonpfad von v_0 nach v_l in G, falls s alle Knoten in V durchläuft, d.h. es gilt $l = \|V\| - 1$ und $V = \{v_0, \ldots, v_l\}$
- Ist zudem $\{v_0, v_l\} \in E$, d.h. $s' = (v_0, v_1, \ldots, v_l, v_0)$ ist ein Kreis, der alle Knoten in V durchläuft, so heißt s' Hamiltonkreis

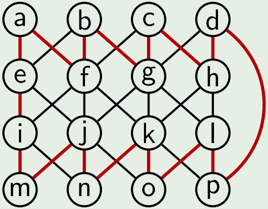
Beispiel (Hamiltonpfad)

$$s = (a, b, e, f, i, j, m, n, o, p, k, l, g, h, c, d)$$



Beispiel (Hamiltonkreis)

$$s = (a, f, b, g, c, h, d, p, l, o, k, n, j, m, i, e, a)$$



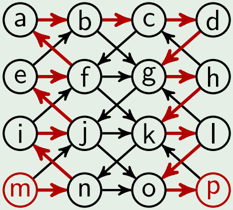
Gerichtete Hamiltonpfade

Definition

- Sei $s = (v_0, \ldots, v_l)$ ein gerichteter Pfad in einem Digraphen $G = (V, E)$, d.h. $(v_i, v_{i+1}) \in E$ für $i = 0, \ldots, l-1$ und $v_0, \ldots, v_l$ sind alle verschieden
- Dann heißt s Hamiltonpfad von v_0 nach v_l in G, falls s alle Knoten in V durchläuft, d.h. es gilt $V = \{v_0, \ldots, v_l\}$
- Ist zudem $(v_l, v_0) \in E$, d.h. $s' = (v_0, v_1, \ldots, v_l, v_0)$ ist ein gerichteter Kreis in G, der alle Knoten in V durchläuft, so heißt s' Hamiltonkreis

Beispiel (Hamiltonpfad in einem Digraphen)

$s = (m, n, i, j, e, f, a, b, c, d, g, h, k, l, o, p)$



Für einen gegebenen Graphen G und eine Zahl $k \geq 1$ betrachten wir folgende Probleme:

- CLIQUE
  Hat G eine Clique der Größe k?
- MATCHING
  Hat G ein Matching der Größe k?
- Independent Set (IS)
  Hat G eine stabile Menge der Größe k?
- Vertex Cover (VC)
  Hat G eine Knotenüberdeckung der Größe k?
- Färbbarkeit (COLORING)
  Ist G k-färbbar?

Weitere algorithmische Graphprobleme

Zudem betrachten wir für einen gegebenen Graphen G folgende Probleme:

- k-FÄRBBARKEIT (k-COLORING), $k \geq 1$
  Ist G k-färbbar?
- Das Eulerkreisproblem (EULERCYCLE)
  Hat G einen Eulerkreis?
- Das Hamiltonkreisproblem (HAMCYCLE)
  Hat G einen Hamiltonkreis?

und für einen Graphen G und zwei Knoten s und t folgende Probleme:

- Das Eulerlinienproblem (EULERPATH)
  Hat G eine Eulerlinie von s nach t?
- Das Hamiltonpfadproblem (HAMPATH)
  Hat G einen Hamiltonpfad von s nach t?

Zudem betrachten wir für einen gegebenen Digraphen G folgende Probleme:

- Das gerichtete Eulerkreisproblem ($\textsc{DiEulerCycle}$)
  Hat G einen Eulerkreis?
- Das gerichtete Hamiltonkreisproblem ($\textsc{DiHamCycle}$)
  Hat G einen Hamiltonkreis?

und für einen gegebenen Digraphen G und zwei Knoten s und t:

- Das gerichtete Eulerlinienproblem ($\textsc{DiEulerPath}$)
  Hat G eine Eulerlinie von s nach t?
- Das gerichtete Hamiltonpfadproblem ($\textsc{DiHamPath}$)
  Hat G einen Hamiltonpfad von s nach t?

Satz

- CLIQUE, IS, VC, COLORING, 3-COLORING, HAMCYCLE, HAMPATH, DIHAMPATH und DIHAMCYCLE sind NP-vollständig
- 2-COLORING, MATCHING, EULERCYCLE, EULERPATH, DIEULERCYCLE und DIEULERPATH sind in P entscheidbar

IS ist NP-vollständig

Reduktion von 3-SAT auf IS

- Sei $F = \{C_1, \ldots, C_m\}$ mit $C_j = \{l_{j,1}, \ldots, l_{j,k_j}\}$ für $j = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$
- Betrachte den Graphen $G = (V, E)$ mit

$$V = \{v_{ji} \mid 1 \le j \le m, 1 \le i \le k_j\} \text{ und}$$
$$E = \left\{\{v_{ji}, v_{j'i'}\} \in \binom{V}{2} \middle| j = j' \text{ oder } l_{ji} = \bar{l}_{j'i'} \text{ oder } l_{j'i'} = \bar{l}_{ji}\right\}$$

- Nun gilt

$$\begin{aligned}
F \in 3\text{-}\mathrm{SAT} &\Leftrightarrow \text{es gibt eine Belegung, die in jeder Klausel } C_j \\
&\ \text{(mindestens) ein Literal } l_{j,i_j} \text{ wahr macht} \\
&\Leftrightarrow \text{es gibt } m \text{ Literale } l_{1,i_1}, \ldots, l_{m,i_m}, \text{ die paarweise} \\
&\ \text{nicht komplementär sind (d.h. } l_{ji_j} \ne \bar{l}_{j'i_{j'}} \text{ für } j \ne j') \\
&\Leftrightarrow \text{es gibt } m \text{ Knoten } v_{1,i_1}, \ldots, v_{m,i_m}, \text{ die nicht durch} \\
&\ \text{Kanten verbunden sind} \\
&\Leftrightarrow G \text{ besitzt eine stabile Menge von } m \text{ Knoten}
\end{aligned}$$

□

CLIQUE ist NP-vollständig

Korollar

CLIQUE ist NP-vollständig

Beweis

- Es ist klar, dass jede Clique in einem Graphen $G = (V, E)$ eine stabile Menge in dem zu G komplementären Graphen $\bar{G} = (V, \bar{E})$ mit $\bar{E} = \binom{V}{2} \setminus E$ ist und umgekehrt
- Daher lässt sich IS mittels

$$f : (G, k) \mapsto (\bar{G}, k)$$

auf CLIQUE reduzieren $\square$

VC ist NP-vollständig

Korollar

VC ist NP-vollständig

Beweis

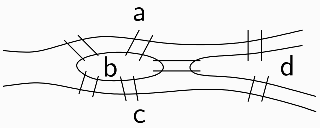
- Offensichtlich ist eine Menge I genau dann stabil, wenn ihr Komplement $V \setminus I$ eine Knotenüberdeckung ist
- Daher lässt sich IS mittels

$$f : (G, k) \mapsto (G, n(G) - k)$$

auf VC reduzieren □

Das Königsberger Brückenproblem

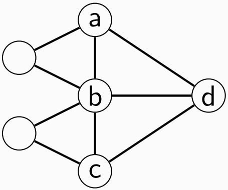
Die 7 Königsberger Brücken



Frage

Gibt es einen Spaziergang über alle 7 Brücken, bei dem keine Brücke mehrmals überquert wird und der zum Ausgangspunkt zurückführt?

Gelöst von Euler (1707 – 1783) durch Betrachtung des folgenden Graphen, der offenbar genau dann einen Eulerkreis hat, wenn die Antwort auf obige Frage „ja“ ist



Satz (Euler, 1736)

Ein zusammenhängender Graph $G = (V, E)$ besitzt genau dann einen Eulerkreis, wenn all seine Knoten geraden Grad haben

Eulerkreise und -linien

Satz (Euler, 1736)

Ein zusammenhängender Graph $G = (V, E)$ besitzt genau dann einen Eulerkreis, wenn all seine Knoten geraden Grad haben

Beweis

- Falls G einen Eulerkreis s besitzt, existiert zu jeder Kante, auf der s zu einem Knoten gelangt, eine weitere Kante, auf der s den Knoten wieder verlässt
- Daher muss jeder Knoten geraden Grad haben
- Ist umgekehrt G zusammenhängend und hat jeder Knoten geraden Grad, so können wir wie folgt einen Eulerkreis s konstruieren

Eulerkreise und -linien

- Ist umgekehrt G zusammenhängend und hat jeder Knoten geraden Grad, so können wir wie folgt einen Eulerkreis s konstruieren

Algorithmus zur Berechnung eines Eulerkreises in $G = (V, E)$

Wähle $u \in V$ beliebig und initialisiere s zu $s = (u)$

Wähle einen beliebigen Knoten u auf dem Weg s, der mit einer unmarkierten Kante verbunden ist

Folge ausgehend von u den unmarkierten Kanten auf einem beliebigen Weg z solange wie möglich und markiere dabei jede durchlaufene Kante (da von jedem erreichten Knoten $v \neq u$ ungerade viele markierte Kanten ausgehen, muss der Weg z zum Ausgangspunkt u zurückführen)

Füge den Zyklus z an der Stelle u in s ein

Wenn noch nicht alle Kanten markiert sind, gehe zu 2

Output: s

$\square$

Eulerkreise und -linien

Satz (Euler, 1736)

(i) Ein zusammenhängender Graph $G = (V, E)$ besitzt im Fall $s \neq t$ genau dann eine Eulerlinie von s nach t, wenn s und t ungeraden und alle übrigen Knoten geraden Grad haben

(ii) Ein stark zusammenhängender Digraph besitzt genau dann einen Eulerkreis, wenn für jeden Knoten u der Eingangs- und der Ausgangsgrad übereinstimmen

Beweis

(i) Da G im Fall $s \neq t$ genau dann eine Eulerlinie von s nach t hat, wenn der Graph $G' = (V \cup \{u_{neu}\}, E \cup \{\{t, u_{neu}\}, \{u_{neu}, s\}\})$ einen Eulerkreis hat, folgt dies aus dem vorigen Satz

(ii) Dies folgt vollkommen analog zum ungerichteten Fall $\square$

Satz

HamPath, HamCycle, DiHamPath und DiHamCycle sind NP-vollständig

Bemerkung

Bevor wir den Satz beweisen, betrachten wir ein weiteres Problem, das mit dem Hamiltonkreisproblem eng verwandt ist und große praktische Bedeutung hat

Das Problem des Handlungsreisenden

- Gegeben sind die Entfernungen d_{ij} zwischen n Städten $i, j \in \{1, \dots, n\}$
- Gesucht ist eine Rundreise $(i_1, \dots, i_n)$ mit minimaler Länge $d_{i_1,i_2} + \dots + d_{i_{n-1},i_n} + d_{i_n,i_1}$, die jede Stadt genau einmal besucht
- Die Entscheidungsvariante dieses Optimierungsproblems ist wie folgt definiert

Problem des Handlungsreisenden (TSP; *traveling salesman problem*)

Gegeben: Eine $n \times n$ Matrix $D = (d_{i,j}) \in \mathbb{N}^{n \times n}$ und eine Zahl k

Gefragt: Existiert eine Permutation $\pi : \{1, \dots, n\} \to \{1, \dots, n\}$, so dass die Rundreise $(\pi(1), \dots, \pi(n))$ die Länge $\leq k$ hat?

Hamiltonkreise, Hamiltonpfade und TSP

Satz

3-SAT $\le^p$ DIHAMPATH $\le^p$ HAMPATH $\le^p$ HAMCYCLE $\le^p$ TSP

Reduktion von HAMCYCLE auf TSP

- Sei $G = (V, E)$ ein Graph, wobei wir $V = \{1, \ldots, n\}$ annehmen
- Dann lässt sich G in Polynomialzeit auf die TSP Instanz (D, n) mit $D = (d_{i,j})$ und

$$d_{i,j} = \begin{cases} 1, & \text{falls } \{i, j\} \in E, \\ 2, & \text{sonst,} \end{cases}$$

  transformieren
- Diese Reduktion ist korrekt, da G genau dann einen Hamiltonkreis hat, wenn es in dem Distanzgraphen D eine Rundreise $(\pi(1), \ldots, \pi(n))$ der Länge $L(\pi) \le n$ gibt $\square$

Hamiltonkreise, Hamiltonpfade und TSP

Satz

3-SAT $\leq^p$ DIHAMPATH $\leq^p$ HAMPATH $\leq^p$ HAMCYCLE $\leq^p$ TSP

Reduktion von HAMPATH auf HAMCYCLE

- Seien ein Graph $G = (V, E)$ und zwei Knoten $s, t \in V$ gegeben
- Wir transformieren G in den Graphen $G' = (V', E')$ mit

$$V' = V \cup \{u_{\text{neu}}\} \text{ und}$$

$$E' = E \cup \Big\{ \{t, u_{\text{neu}}\}, \{u_{\text{neu}}, s\} \Big\}$$

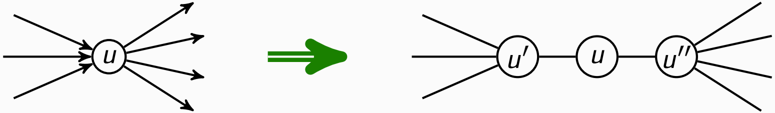
- Offenbar ist G' in Polynomialzeit aus G berechenbar und besitzt genau dann einen Hamiltonkreis, wenn G einen s-t-Hamiltonpfad besitzt $\square$

Satz

3-SAT $\leq^p$ DIHAMPATH $\leq^p$ HAMPATH $\leq^p$ HAMCYCLE $\leq^p$ TSP

Reduktion von DIHAMPATH auf HAMPATH

- Seien ein Digraph $G = (V, E)$ und zwei Knoten $s, t \in V$ gegeben
- Zuerst entfernen wir alle Kanten, die von t ausgehen oder in s enden
- Dann konstruieren wir den Graphen G', indem wir lokal für jeden Knoten $u \in V$ die folgende Ersetzung durchführen:



- Hierbei lassen wir u' (bzw. u'') weg, falls keine Kanten in u enden (bzw. von u ausgehen)
- Dann ist die Funktion $G \mapsto G'$ in FP berechenbar und G enthält genau dann einen Hamiltonpfad von s nach t, wenn dies auf G' zutrifft □

Reduktion von 3-SAT auf das Hamiltonpfadproblem

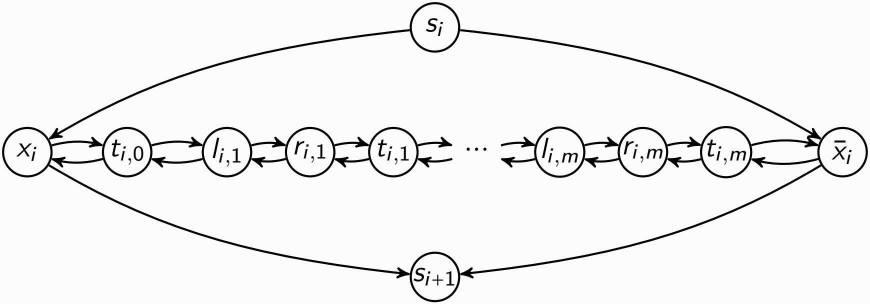
Satz

3-SAT $\leq^p$ DIHAMPATH $\leq^p$ HAMPATH $\leq^p$ HAMCYCLE $\leq^p$ TSP

Reduktion von 3-SAT auf DIHAMPATH

- Sei $F = \{C_1, \ldots, C_m\}$ mit $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ für $j = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$
- Wir transformieren F in Polynomialzeit in einen gerichteten Graphen $G_F = (V, E)$ mit zwei ausgezeichneten Knoten s und t, der genau dann einen hamiltonschen s-t-Pfad besitzt, wenn F erfüllbar ist
- Jede Klausel C_j repräsentieren wir durch einen Knoten c_j und jede Variable x_i repräsentieren wir durch folgenden Graphen X_i

- Jede Klausel C_j repräsentieren wir durch einen Knoten c_j und jede Variable x_i repräsentieren wir durch folgenden Graphen X_i:



- Ein Pfad von s_1 nach s_{n+1} kann ausgehend von s_i ($i = 1, \ldots, n$) entweder zuerst den Knoten x_i oder zuerst den Knoten $\bar{x}_i$ besuchen
- Daher können wir jedem s_1-s_{n+1}-Pfad P eine Belegung $b_P = b_1 \cdots b_n$ zuordnen mit $b_i = 1$ gdw. P den Knoten x_i vor dem Knoten $\bar{x}_i$ besucht

Reduktion von 3-SAT auf DIHAMPATH

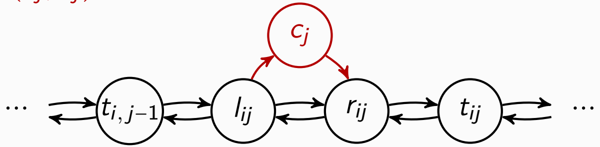
- Die Knotenmenge V von G_F ist also

$$V = \{c_1, \ldots, c_m\} \cup \{s_1, \ldots, s_{n+1}\} \cup \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\} \cup \bigcup_{i=1}^{n} \{t_{i,0}, l_{i,1}, r_{i,1}, t_{i,1}, \cdots, l_{i,m}, r_{i,m}, t_{i,m}\}$$

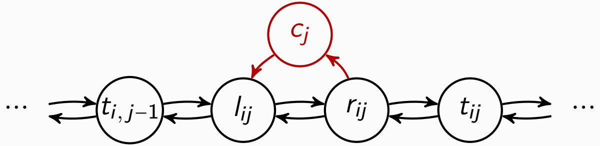
- Dabei haben die Graphen X_{i-1} und X_i den Knoten s_i gemeinsam
- Als Startknoten wählen wir $s = s_1$ und als Zielknoten $t = s_{n+1}$
- Jetzt fehlen nur noch die Verbindungskanten zwischen den Teilgraphen X_i und den Klauselknoten c_j
- Diese Kanten sollen einem s-t-Pfad P genau dann einen Abstecher nach c_j ermöglichen, wenn die Belegung b_P die Klausel C_j erfüllt

Reduktion von 3-SAT auf DIHAMPATH

- Für jedes Literal $l \in C_j$ fügen wir zu E im Fall $l = x_i$ die beiden Kanten (l_{ij}, c_j) und (c_j, r_{ij}),



  und im Fall $l = \bar{x}_i$ die Kanten (r_{ij}, c_j) und (c_j, l_{ij}) hinzu:



- Man beachte, dass ein s-t-Pfad P über diese Kanten genau dann einen Abstecher zu c_j machen kann, wenn die Belegung b_P das Literal l wahr macht

Reduktion von 3-SAT auf DIHAMPATH

- Zunächst ist klar, dass die Reduktionsfunktion $F \mapsto (G_F, s, t)$ in Polynomialzeit berechenbar ist
- Es bleibt also zu zeigen, dass F genau dann erfüllbar ist, wenn in G_F ein Hamiltonpfad von $s = s_1$ nach $t = s_{n+1}$ existiert
- Falls $F(b) = 1$ ist, so erhalten wir einen Hamiltonpfad, indem wir in jeder Klausel C_j ein wahres Literal $l = x_i$ bzw. $l = \bar{x}_i$ auswählen und in den zu b gehörigen s-t-Pfad P einen „Abstecher" vom Knotenpaar l_{ij}, r_{ij} zum Klauselknoten c_j einbauen
- Ist umgekehrt P ein s-t-Hamiltonpfad in G_F, so müssen der Vorgänger- und Nachfolgerknoten jedes Klauselknotens c_j ein Paar l_{ij}, r_{ij} bilden, da P andernfalls nicht beide Pufferknoten $t_{i,j-1}$ und $t_{i,j}$ besuchen kann
- Da aber P alle Klauselknoten besucht und ausgehend von dem Paar l_{ij}, r_{ij} nur dann ein Abstecher zu c_j möglich ist, wenn die Belegung b_P die Klausel C_j erfüllt, folgt $F(b_P) = 1$ $\square$

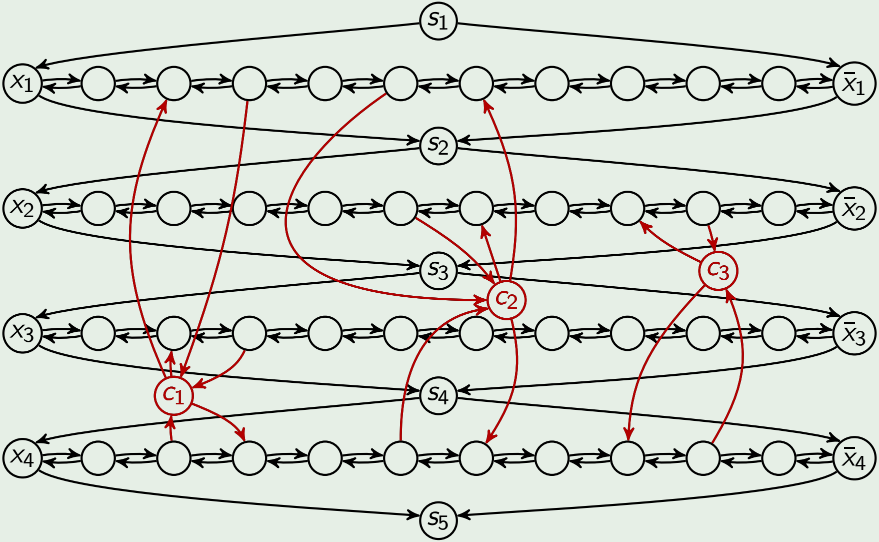
Beispiel

Für die 3-KNF-Formel

$$F = (\bar{x}_1 \lor \bar{x}_3 \lor x_4) \land (x_1 \lor x_2 \lor x_4) \land (\bar{x}_2 \lor \bar{x}_4)$$

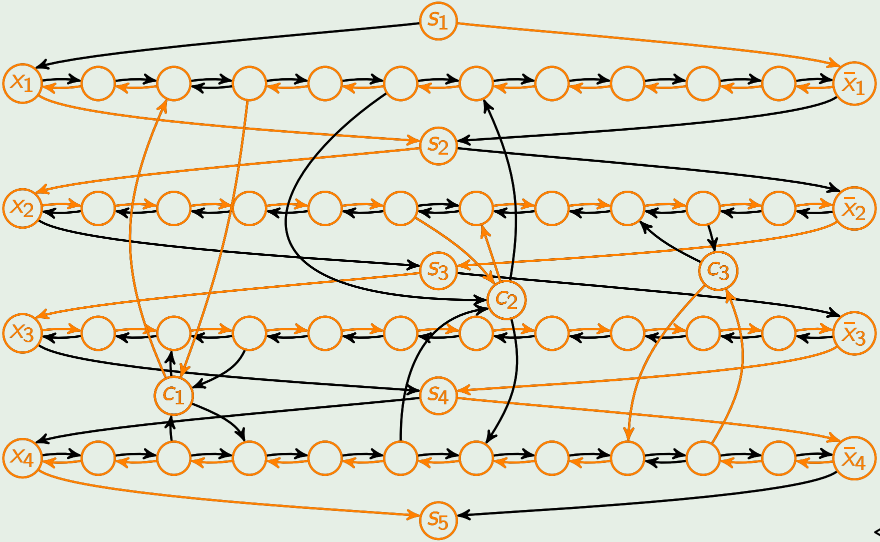
erhalten wir folgenden Digraphen G mit Startknoten $s = s_1$ und Zielknoten $t = s_5$:

Für $F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4)$ erhalten wir folgenden Digraphen G mit Startknoten $s = s_1$ und Zielknoten $t = s_5$:



Reduktion von 3-SAT auf DIHAMPATH

Der Belegung $b = 0110$ von $F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4)$ entspricht beispielsweise folgender Hamiltonpfad von s_1 nach s_5:



◁

Das Rucksack-Problem

- Wie schwierig ist es, einen Rucksack der Größe w mit einer Auswahl aus k Gegenständen der Größe $u_1, \dots, u_k$ möglichst voll zu packen?
- Dieses Optimierungsproblem lässt sich leicht auf folgendes Entscheidungsproblem reduzieren

RUCKSACK:

Gegeben: Eine Folge $(u_1, \dots, u_k, v, w)$ von natürlichen Zahlen

Gefragt: Ex. eine Auswahl $S \subseteq \{1, \dots, k\}$ mit $v \leq \sum_{i \in S} u_i \leq w$?

Beim SubsetSum-Problem möchte man dagegen nur wissen, ob der Rucksack randvoll gepackt werden kann:

SUBSETSUM:

Gegeben: Eine Folge $(u_1, \dots, u_k, w)$ von natürlichen Zahlen

Gefragt: Ex. eine Auswahl $S \subseteq \{1, \dots, k\}$ mit $\sum_{i \in S} u_i = w$?

Rucksack und SubsetSum sind NP-vollständig

Satz

Rucksack und SubsetSum sind NP-vollständig

Beweis

- Es ist klar, dass beide Probleme in NP enthalten sind
- Zum Nachweis der NP-Härte zeigen wir die folgenden Reduktionen:

$$\text{3-Sat} \leq^p \text{SubsetSum} \leq^p \text{Rucksack}$$

Reduktion von SubsetSum auf Rucksack

Da SubsetSum einen Spezialfall des Rucksack-Problems darstellt, lässt es sich leicht darauf reduzieren:

$$(u_1, \ldots, u_k, w) \mapsto (u_1, \ldots, u_k, w, w)$$

□

SubsetSum ist NP-vollständig

Reduktion von 3-Sat auf SubsetSum

- Sei $F = \{C_1, \ldots, C_m\}$ mit $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ für $j = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$
- Betrachte die Reduktionsfunktion

  $$f : F \mapsto (u_1, \ldots, u_n, u'_1, \ldots, u'_n, v_1, \ldots, v_m, v'_1, \ldots, v'_m, w)$$

  mit den Dezimalzahlen

  $$u_i = b_{i1} \cdots b_{im} 0^{i-1} 1 0^{n-i} \qquad v_j = v'_j = 0^{j-1} 1 0^{m-j-1} 0^n$$

  $$u'_i = b'_{i1} \cdots b'_{im} 0^{i-1} 1 0^{n-i} \qquad w = \underbrace{3 \cdots 3}_{m\text{-mal}} \underbrace{1 \cdots 1}_{n\text{-mal}}$$

  und den Ziffern

  $$b_{ij} = \begin{cases} 1 & x_i \in C_j \\ 0 & \text{sonst} \end{cases} \qquad \text{und} \qquad b'_{ij} = \begin{cases} 1 & \bar{x}_i \in C_j \\ 0 & \text{sonst} \end{cases}$$

- Hierbei können führende Nullen natürlich auch weggelassen werden

SubsetSum ist NP-vollständig

Beispiel

- Betrachte die 3-KNF Formel

$$F = (\bar{x}_1 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4)$$

- Die zu F gehörige SubsetSum-Instanz $f(F)$ ist

$$(u_1, u_2, u_3, u_4, u'_1, u'_2, u'_3, u'_4, v_1, v_2, v_3, v'_1, v'_2, v'_3, w)$$

mit

$$\begin{array}{llll} u_1 = 010\,1000 & u_2 = 010\,0100 & u_3 = 000\,0010 & u_4 = 110\,0001 \\ u'_1 = 100\,1000 & u'_2 = 001\,0100 & u'_3 = 100\,0010 & u'_4 = 001\,0001 \end{array}$$

und

$$\begin{array}{lll} v_1 = 100\,0000 & v_2 = 010\,0000 & v_3 = 001\,0000 \\ v'_1 = 100\,0000 & v'_2 = 010\,0000 & v'_3 = 001\,0000 \end{array}$$

sowie

$$w = 333\,1111$$

- Der erfüllenden Belegung $a = 0100$ entspricht dann die Auswahl $(u'_1, u_2, u'_3, u'_4, v_1, v_2, v'_2, v_3, v'_3)$ ◁

SubsetSum ist NP-vollständig

Beweis von $F \in$ 3-SAT $\Rightarrow f(F) \in$ SUBSETSUM

- Sei $a = a_1 \cdots a_n$ eine erfüllende Belegung für F
- Da a in jeder Klausel mindestens ein und höchstens drei Literale wahr macht, hat die Zahl

$$u = \sum_{a_i=1} u_i + \sum_{a_i=0} u_i'$$

  eine Dezimaldarstellung der Form $b_1 \cdots b_m 1 \cdots 1$ mit $1 \le b_j \le 3$ für $j = 1, \dots, m$
- Durch Addition der Zahl

$$v = \sum_{b_j \le 2} v_j + \sum_{b_j=1} v_j'$$

  erhalten wir $u + v = w$

SubsetSum ist NP-vollständig

Beweis von $f(F) \in \textsc{SubsetSum} \Rightarrow F \in 3\text{-}\textsc{Sat}$

- Sei $S = P \cup N \cup I \cup J$ eine Auswahlmenge für $f(F)$ mit

$$\sum_{i \in P} u_i + \sum_{i \in N} u_i' + \sum_{j \in I} v_j + \sum_{j \in J} v_j' = \underbrace{3\cdots3}_{m\text{-mal}}\underbrace{1\cdots1}_{n\text{-mal}}$$

- Da die Teilsumme $\sum_{j \in I} v_j + \sum_{j \in J} v_j'$ die Form $c_1 \cdots c_m 0 \cdots 0$ mit $c_j \le 2$ hat, muss die Teilsumme $\sum_{i \in P} u_i + \sum_{i \in N} u_i'$ die Form $b_1 \cdots b_m 1 \cdots 1$ mit $b_j \ge 1$ haben
- Da keine Überträge auftreten, muss also $P = \{1, \ldots, n\} - N$ sein, und jede Klausel C_j muss mindestens ein Literal aus der Menge $\{x_i \mid i \in P\} \cup \{\bar{x}_i \mid i \in N\}$ enthalten
- Folglich erfüllt folgende Belegung $a_1 \cdots a_n$ die Formel F:

$$a_i = \begin{cases} 1, & i \in P, \\ 0, & i \in N \end{cases}$$

- Damit haben wir die Korrektheit von f gezeigt $\square$

In vielen Anwendungen tritt das Problem auf, eine ganzzahlige Lösung für ein System linearer Ungleichungen zu finden

Ganzzahlige Programmierung (IP; *integer programming*)

Gegeben: Eine ganzzahlige $m \times n$ Matrix $A = (a_{ij}) \in \mathbb{Z}^{m\times n}$ und ein ganzzahliger Vektor $\boldsymbol{b} \in \mathbb{Z}^m$

Gefragt: Existiert ein ganzzahliger Vektor $\boldsymbol{x} \in \mathbb{Z}^n$ mit

$$A\boldsymbol{x} \geq \boldsymbol{b},$$

wobei $\geq$ komponentenweise zu verstehen ist

Satz

IP ist NP-hart

Reduktion von 3-SAT auf IP

- Sei $F = \{C_1, \ldots, C_m\}$ mit $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ für $j = 1, \ldots, m$ eine 3-KNF-Formel über den Variablen $x_1, \ldots, x_n$
- Wir transformieren F in ein Ungleichungssystem $A\boldsymbol{x} \geq \boldsymbol{b}$ für den Lösungsvektor $\boldsymbol{x} = (x_1, \ldots, x_n, \bar{x}_1, \ldots, \bar{x}_n)$, das
  - für $i = 1, \ldots, n$ die vier Ungleichungen

$$x_i + \bar{x}_i \geq 1, \quad -x_i - \bar{x}_i \geq -1, \quad x_i \geq 0, \quad \bar{x}_i \geq 0 \qquad (*)$$

  - und für jede Klausel $C_j = \{l_{j1}, \ldots, l_{jk_j}\}$ folgende Ungleichung enthält:

$$l_{j1} + \cdots + l_{jk_j} \geq 1 \qquad (**)$$

- Die Ungleichungen $(*)$ sind für ganzzahlige $x_i, \bar{x}_i$ genau dann erfüllt, wenn x_i den Wert 0 und $\bar{x}_i$ den Wert 1 hat oder umgekehrt
- Die Klauselungleichungen $(**)$ stellen sicher, dass mindestens ein Literal in jeder Klausel C_j wahr wird
- somit entspricht jede Lösung $\boldsymbol{x}$ von $A\boldsymbol{x} \geq \boldsymbol{b}$ einer erfüllenden Belegung von F und umgekehrt $\square$

(Ganzzahlige) lineare Programmierung

Bemerkungen zu IP

- Es ist nicht offensichtlich, dass IP in NP entscheidbar ist
- Ein nichtdeterministischer Algorithmus kann zwar eine Lösung raten, aber a priori ist nicht klar, ob eine Lösung $\boldsymbol{x}$ ex., deren Binärkodierung polynomiell in der Länge der Eingabe $(A, \boldsymbol{b})$ ist
- Mit Methoden der linearen Algebra lässt sich jedoch zeigen, dass jede lösbare IP-Instanz $(A, \boldsymbol{b})$ auch eine Lösung $\boldsymbol{x}$ hat, deren Kodierung polynomiell in der Länge von $(A, \boldsymbol{b})$ ist
- Wenn wir nicht verlangen, dass die Lösung $\boldsymbol{x}$ der IP-Instanz ganzzahlig ist, dann spricht man von einem linearen Programm
- Für LP (Lineare Programmierung) gibt es Polynomialzeitalgorithmen (von Khachiyan 1979 und von Karmarkar 1984)

SAT als Optimierungsproblem

- In manchen Anwendungen ist es wichtig zu wissen, wieviele Klauseln einer KNF-Formel F maximal erfüllbar sind
- Hierbei können in F auch Klauseln mehrfach vorkommen
- Die Komplexität dieses Optimierungsproblems lässt sich durch folgendes Entscheidungsproblem charakterisieren

MAX-k-SAT:

Gegeben: Eine Formel F in k-KNF und eine Zahl l

Gefragt: Gibt es eine Belegung, die mindestens l Klauseln in F erfüllt?

Satz

1. MAX-1-SAT $\in$ P
2. MAX-2-SAT ist NP-vollständig

SAT als Optimierungsproblem

Reduktion von 3-SAT auf MAX-2-SAT

- Für eine Dreierklausel $C = \{l_1, l_2, l_3\}$, in der die Variable v nicht vorkommt, sei $G(l_1, l_2, l_3, v)$ die 2-KNF Formel, die aus folgenden 10 Klauseln besteht:

$$\{l_1\}, \{l_2\}, \{l_3\}, \{v\}, \{\overline{l_1}, \overline{l_2}\}, \{\overline{l_2}, \overline{l_3}\}, \{\overline{l_1}, \overline{l_3}\}, \{l_1, \overline{v}\}, \{l_2, \overline{v}\}, \{l_3, \overline{v}\}$$

- Die folgenden 3 Eigenschaften von G sind leicht zu verifizieren:
  - Keine Belegung von G erfüllt mehr als 7 Klauseln von G
  - Jede Belegung a von C mit $C(a) = 1$ ist zu einer Belegung a' von G erweiterbar, die 7 Klauseln von G erfüllt
  - Keine Belegung a von C mit $C(a) = 0$ ist zu einer Belegung a' von G erweiterbar, die 7 Klauseln von G erfüllt
- Sei F eine 3-KNF-Formel über $x_1, \ldots, x_n$ mit m Klauseln
- Wir nehmen an, dass $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$, $j = 1, \ldots, k$, die Dreier- und $C_{k+1}, \ldots, C_m$ die Einer- und Zweierklauseln von F sind

Sat als Optimierungsproblem

Reduktion von 3-Sat auf Max-2-Sat

- Wir nehmen an, dass $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$, $j = 1, \ldots, k$, die Dreier- und $C_{k+1}, \ldots, C_m$ die Einer- und Zweierklauseln von F sind
- Betrachte die 2-KNF Formel F' mit $10k + (m - k)$ Klauseln, die wie folgt aus F entsteht:
  - Ersetze jede Dreierklausel $C_j = \{l_{j1}, l_{j2}, l_{j3}\}$ in F durch die 10 Klauseln der 2-KNF-Formel $G_j = G(l_{j1}, l_{j2}, l_{j3}, v_j)$
- Dann gilt

$$F \in 3\text{-}\textsc{Sat} \Leftrightarrow (F', m + 6k) \in \textsc{Max}\text{-}2\text{-}\textsc{Sat}$$

- Die Vorwärtsrichtung ergibt sich unmittelbar aus der Tatsache, dass jede erfüllende Belegung a für F zu einer Belegung a' für F' erweiterbar ist, die $7k + m - k = m + 6k$ Klauseln von F' erfüllt

SAT als Optimierungsproblem

Reduktion von 3-SAT auf MAX-2-SAT

- Dann gilt

$$F \in \text{3-SAT} \Leftrightarrow (F', m + 6k) \in \text{MAX-2-SAT}$$

- Die Vorwärtsrichtung ergibt sich unmittelbar aus der Tatsache, dass jede erfüllende Belegung a für F zu einer Belegung a' für F' erweiterbar ist, die $7k + m - k = m + 6k$ Klauseln von F' erfüllt
- Für die Rückwärtsrichtung sei a eine Belegung, die mindestens $m + 6k$ Klauseln von F' erfüllt
- Da in jeder 10er-Gruppe G_j, $j = 1, \ldots, k$, maximal 7 Klauseln erfüllbar sind, muss a in jeder 10er-Gruppe genau 7 Klauseln und zudem alle Klauseln C_j für $j = k + 1, \ldots, m$ erfüllen
- Dies ist aber nur möglich, wenn a alle Klauseln C_j von F erfüllt
- Zudem ist klar, dass $g : F \mapsto (F', m + 6k)$ in FP berechenbar ist, woraus 3-SAT $\leq^p$ MAX-2-SAT folgt □