Informatik III

Prof. Dr. H. Prautzsch

Skript zur Vorlesung gehalten im Winter 1999/2000

Inhaltsverzeichnis

I Formale Sprachen **7**

1 Sprachen und Grammatiken **9**
1.1 Grundbegriffe 9
1.2 Grammatiken 10
1.3 Chomsky-Hierarchie 11
1.4 Abschlußeigenschaften 12

2 Reguläre Sprachen **15**
2.1 Deterministische endliche Akzeptoren 15
2.2 Nichtdeterministische endliche Automaten 16
2.3 Das Pump-Lemma 18
2.4 Reguläre Ausdrücke 19
2.5 Eine Äquivalenzrelation 20
2.6 Der Minimalautomat 22
2.7 Abschlußeigenschaften 24
2.8 Entscheidbare Probleme 24

3 Kontextfreie Sprachen **25**
3.1 Syntaxbäume 25
3.2 Das Pump-Lemma 26
3.3 Einfache Kellerautomaten 28
3.4 Einfache Kellerautomaten u. kontextfr. Sprachen 29
3.5 Allgemeine Kellerautomaten 30
3.6 Chomsky Normalform 32
3.7 Cocke-Kasami-Younger-Algorithmus 33
3.8 Abschlußeigenschaften 35
3.9 Entscheidbare Probleme 36

4 Sprachen vom Typ 0 und 1 **39**
4.1 Kontextsensitive Grammatiken 39
4.2 Turingmaschinen 40
4.3 Ein Beispiel 41
4.4 Linear beschränkte Turing-Maschine 42
4.5 Deterministische Turingmaschinen 44
4.6 Abschlusseigenschaften 44
4.7 Das Halteproblem 45
4.8 Das Wortproblem 46
4.9 Das Postsche Korrespondenzproblem 47

4.10 (Un)entscheidbare Probleme 48

5 Fixpunkttheorie **51**
5.1 Halbordnungen 51
5.2 Ein Fixpunkttheorem 53
5.3 Kontextfreie Sprachen als Fixpunkte 56
5.4 Gleichungssysteme 57

6 Syntaxanalyse **59**
6.1 Einführende Beispiele 59
6.2 SLL(k)-Grammatiken 60
6.3 LR(k)-Grammatiken 62
6.4 LR(k)-Analyse 63
6.5 Linke Kontexte 64
6.6 Kontextmengen 65
6.7 Kellerautomaten für LR(k)-Grammatiken 67

II Berechenbarkeitstheorie **69**

7 Berechenbarkeitstheorie **71**
7.1 Intuitiver Berechenbarkeitsbegriff 71
7.2 Turing-Berechenbarkeit 72
7.3 `loop`- und `while`-Berechenbarkeit 73
7.4 Makros 74
7.5 Vergleich von `while`- und Turing-Berechenbarkeit 76
7.6 `goto`-Programme 77
7.7 Zusammenfassung 78
7.8 Die Ackermannfunktion 78

8 Universelle Funktionen **81**
8.1 Aufzählung der `while`-Programme 81
8.2 Indexüberprüfung 82
8.3 Kompakte `while`-Programme 83
8.4 Der Interpreter 84

9 Elementare Techniken **87**
9.1 Beispiele 87
9.2 Das s-n-m-Theroem 88
9.3 Unentscheidbare Probleme 89

10 Rekursive `while`-Programme **91**
10.1 Beispiel 91
10.2 Fixpunkte von Programmumschreibefunktionen 92
10.3 Beispiele 93
10.4 Der Rekursionssatz 94
10.5 Zusammenfassung 96

11 Entscheid- und Aufzählbarkeit **99**
11.1 Rekursive und rekursiv aufzählbare Mengen 99
11.2 Zusammenhänge zwischen rek. und rek. a. Mengen 101
11.3 Andere Definitionen von rekursiv aufzählbaren Mengen 102
11.4 Reduktionen 103
11.5 Der Satz von Rice 105
11.6 Erweiterungen des Riceschen Satzes 106

12 Gödels Satz **109**
12.1 Arithmetische Formeln 109
12.2 Lemma 110
12.3 Satz 111
12.4 Gödels Unvollständigkeitssatz 111

III Komplexitätstheorie **113**

13 Komplexitätstheorie **115**
13.1 Komplexitätsklassen 115
13.2 NP-Vollständigkeit 116
13.3 Das Erfüllbarkeitsproblem 117
13.4 Beweis 118
13.5 3SAT 120
13.6 Gerichteter Hamiltonkreis 122
13.7 Ungerichteter Hamiltonkreis und Handlungsreisender 124
13.8 Das Problem des Handlungsreisenden im $\mathbb{R}^2$ 125
13.9 Das Rucksack-Problem 127
13.10Partition 128
13.11Das Behälterproblem 129
13.12Äquivalenz regulärer Ausdrücke 129
13.13Das Wortproblem für kontextsensitive Sprachen 130

IV Stochastische Algorithmen **133**

14 Stochastische Algorithmen **135**
14.1 Einführung 135
14.2 Sortieren durch stochastisches Teilen 135
14.3 Der minimale Schnitt 136
14.4 Komplexitätsklassen 138
14.5 Binäre Zerlegungen der Ebene 140
14.6 Binäre Raumzerlegung 142

15 Konvexe Hülle **145**
15.1 Grundbegriffe 145
15.2 Duale Graphen 146
15.3 Ein stochastischer inkrementeller Algorithmus 148
15.4 Der Aufwand 150

16 Sätze von Yao und Adleman **153**
16.1 Spieltheorie 153
16.2 Yaos Technik 155
16.3 Satz von Adleman 156

V Schluß **159**

17 Lindenmayersysteme **161**

18 Fraktale **163**
18.1 Kontraktionen 163
18.2 Iterierte Funktionensysteme 164

Ablauf

Formale Sprachen	Okt-Dez
Berechenbarkeitstheorie	Dez-Jun
Komplexitätstheorie	Jan
stochastische Algorithmen	Feb

Literaturempfehlung

- Schöning: Theoretische Informatik kurzgefasst, BI 1992
- Moll, Arbib, Kfoury: Introduction to formal languages, Springer 1988
- Kfoury, Moll, Arbib: A progr. approach to computability, Springer 1986
- Motwami, Raghavan: Randomized algorithms

Termine

- Klausuren: 22.02. und 14.04.
- Sprechstunden: Informatik-Neubau, Zimmer 126 (Mi 15-16 Uhr; am 17.11., 26.1. und 16.2. 17-17.30 Uhr)
- Homepage: `http://i33www.ira.uka.de/`
- Übungsleiter: Herr Dipl.-Inf. Linsen (Sprechstunde Mi 14-15 Uhr)

Teil I

Formale Sprachen

Kapitel 1

Sprachen und Grammatiken

1.1 Grundbegriffe

Alphabet: nichtleere Menge von Zeichen
$\mathcal{A}^+ :=$ Menge aller endlichen Zeichenketten über $\mathcal{A}$
$\mathcal{A}^* := \mathcal{A}^+ \cup \{\lambda\}$
$\lambda :=$ die leere Zeichenkette
Eine *Sprache* über $\mathcal{A}$ ist eine Teilmenge von $\mathcal{A}^*$.

Beispiel:

$\mathcal{A} := \{0,1\} \Rightarrow \mathcal{A}^+ := \{0,1,00,01,11,...\}$
$\{0\}^*$ist eine Sprache über $\mathcal{A}$.

Konkatenation: Zusammensetzen zweier Zeichenketten α, β zu einem Wort $\alpha\beta$

Beispiel:

1. $\alpha := 01, \beta := 111 \Rightarrow \alpha\beta = 01111, \beta\alpha = 11101$

2. $\alpha\lambda = \alpha = \lambda\alpha$

3. $\alpha^0 = \lambda, \alpha^2 = \alpha\alpha, \alpha^3 = \alpha\alpha\alpha$, usw.
   β ist ein *Teilwort* von $\omega \in \mathcal{A}^*$, falls $\alpha\,\gamma \in \mathcal{A}^*$ existiert mit $\omega = \alpha\beta\gamma$.
   Eine *Produktion* (oder Ersetzungsregel) ist ein Paar $(\alpha, \beta) \in \mathcal{A}^* \times \mathcal{A}^*$geschrieben $\alpha \to \beta$

Ein **Semi-Thue-System** besteht aus einem Alphabet $\mathcal{A}$ und einer endlichen Menge Π von Produktionen.

Sei $\rho = u\alpha v \in \mathcal{A}^*$, $\sigma = u\beta v \in \mathcal{A}^*$, $\alpha \to \beta \in \Pi$. Dann heißt σ *direkt* aus ρ *ableitbar*. Auch: ρ erzeugt direkt σ, kurz $\rho \Rightarrow \sigma$.
Sei $\rho \Rightarrow \omega_1 \Rightarrow \omega_2 \Rightarrow ... \Rightarrow \omega_n \Rightarrow \sigma$. Dann heißt σ aus ρ *ableitbar*. Auch: ρ erzeugt σ, kurz $\rho \overset{*}{\Rightarrow} \sigma$.

Beispiel:

$\mathcal{A} := \{0, 1, q\}$
$\Pi := \{q \to 0q0, q \to 1q1, q \to 0, q \to 1, q \to \lambda\}$
Dann erzeugt q alle *Palindrome*[1] über {0,1}.

Bemerkung:

$\rho \overset{*}{\Rightarrow} \sigma$ ist unentscheidbar.

1.2 Grammatiken

$(\mathcal{V}, \mathcal{A}, S, \Pi)$ ist eine *Grammatik*, falls

- $\mathcal{V} \cap \mathcal{A} = \emptyset$,
- $S \in \mathcal{V}$,
- Π endliche Menge von Produktionen $\alpha \to \beta$ ist, wobei α ein Zeichen aus $\mathcal{V}$ enthält und $\beta \in (\mathcal{V} \cup \mathcal{A})^*$.

$\mathcal{A}$ ist das *Terminalalphabet*, $a \in \mathcal{A}$ sind die *Terminalzeichen* (kleine Buchstaben oder Ziffern).
$\mathcal{V}$ ist das *Variablenalphabet*,
$A \in \mathcal{V}$ eine *Variable* (große Buchstaben),
S *Startsymbol*.

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine Grammatik, dann ist $\mathcal{L}(G) := \{\omega \in \mathcal{A}^* | S \overset{*}{\Rightarrow} \omega\}$ die von G *erzeugte Sprache*.

Beispiel 1

$\mathcal{V} := \{S\}$, $\mathcal{A} := \{a, b\}$, $\Pi := \{S \to ab, S \to aSb\}$.
Kurznotation für $G : S \to ab|aSb$
Dann ist $\mathcal{L}(G) = \{a^n b^n | n \in \mathbb{N}\}$.

Beispiel 2

G sei gegeben durch:
$S \to 0B|1A$
$A \to 0|0S|1AA$
$B \to 1|1S|0BB$
Dann ist $\mathcal{L}(G) = \{\omega \in \{0, 1\}^+ | \omega$ enthält genausoviele Einsen wie Nullen$\}$.

Beweis

als Aufgabe (z.B. durch vollständige Induktion!)

[1] Palindrome sind symmetrische Worte, beispielsweise Lagerregal, Hannah, Maoam oder Leg' in eine so helle Hose nie 'n Igel

Beispiel 3

$S \to aSB|ab$
$aB \to ab$
$bB \to bb$
Dann ist $\mathcal{L}(G) = \{a^n b^n | n \in \mathbb{N}\}$ wie in Beispiel 1.

Bemerkung:

Verschiedene Grammatiken können dieselbe Sprache erzeugen.

1.3 Chomsky-Hierarchie

Eine Grammatik $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ heißt:

- *kontextsensitiv* oder *erweiternd*, falls für alle Produktionen $\alpha \to \beta \in \Pi$ gilt $|\alpha| \leq |\beta|$,
- *kontextfrei*, falls die Produktionen von der Form $A \to \alpha$ sind, wobei $A \in \mathcal{V}$ und $\alpha \in (\mathcal{V} \cup \mathcal{A})^+$,
- *rechtslinear* (*linkslinear*), falls die Produktionen von der Form
  $A \to bC$ $(A \to Cb)$
  $A \to b$
  sind, wobei $A, C \in \mathcal{V}$ und $b \in \mathcal{A}$.

Ausnahme (λ-freie Grammatik): $S \to \lambda$ darf in Π vorkommen, falls S auf keiner rechten Seite einer Produktion vorkommt.

Definition

Eine *Grammatik* heißt vom Typ

0	bei beliebigen Produktionen,
1	falls G kontextsensitiv,
2	falls G kontextfrei,
3	falls G rechtslinear oder linkslinear.

Eine Sprache heißt *vom Typ i*, falls sie von einer Grammatik des Typs i erzeugt wird.

Chomsky-Hierarchie

Für $\mathcal{L}_i$, die Mengen aller Sprachen vom Typ i über $\mathcal{A}$, gilt offensichtlich nach Definition

$$\mathcal{L}_3 \subset \mathcal{L}_2 \subset \mathcal{L}_1 \subset \mathcal{L}_0 \subset 2^{\mathcal{A}^*}.$$

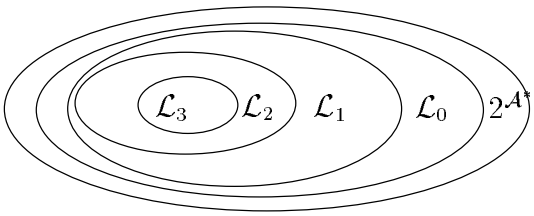


Abbildung 1.1: Teilmengen von $2^{\mathcal{A}^*}$

Satz und Neudefinition

Kontextfreie und rechtslineare Grammatiken mit zusätzlichen λ-Produktionen $A \to \lambda$ erzeugen kontextfreie bzw. rechtslineare Sprachen. Deshalb sollen diese Grammatiken auch kontextfrei bzw. rechtslinear genannt werden.

Beweis

Jede kontextfreie oder rechtslineare Grammatik G kann in eine kontextfreie bzw. rechtslineare Grammatik G' ohne λ-Produktionen umgewandelt werden mit $\mathcal{L}(G') = \mathcal{L}(G)$:

1. Für jede Produktion $A \Rightarrow \alpha B \beta$ mit $B \overset{*}{\Rightarrow} \lambda$ füge die Produktion $A \Rightarrow \alpha\beta$ zu Π hinzu.
2. Wiederhole 1., bis sich Π nicht mehr ändert.
3. Entferne alle λ-Produktionen.
4. Falls $\lambda \in \mathcal{L}(G)$, dann füge $S \to \lambda$ zu Π hinzu.

Satz

$\mathcal{L}_3 \neq \mathcal{L}_2 \neq \mathcal{L}_1 \neq \mathcal{L}_0$

Beweis

später

Satz

$\mathcal{L}_0 \neq 2^{\mathcal{A}^*}$

Beweis

Die Menge der Grammatiken vom Typ 0 mit den Variablen $\mathcal{V}_1 \ldots \mathcal{V}_n$ ist abzählbar $\Rightarrow$ $\mathcal{L}_0$ ist abzählbar.
Aber $2^{\mathcal{A}^*}$ ist überabzählbar (siehe Abbildung 1.1).

1.4 Abschlußeigenschaften

(Erzeugung neuer Sprache aus alten)

Seien $\mathcal{L}$ und $\mathcal{M}$ Sprachen.

Vereinigung: $\mathcal{L} \cup \mathcal{M}$

Konkatenation: $\mathcal{L} \cdot \mathcal{M} := \{\alpha\beta | \alpha \in \mathcal{L}, \beta \in \mathcal{M}\}$

Sternoperator: $\mathcal{L}^* := \{\alpha_1 \to \alpha_n | \alpha_1, \ldots, \alpha_n \in \mathcal{L}, n \in \mathbb{N}\} = \bigcup_{n \geq 0} \mathcal{L}^n$

Satz

Die Menge $\mathcal{L}_i$ aller Sprachen über $\mathcal{A}$ ist abgeschlossen bezüglich $\cup, \cdot, *$

Beweis

Seien $G_1 = (\mathcal{V}_1, \mathcal{A}, S_1, \Pi_1)$ und $G_2 = (\mathcal{V}_2, \mathcal{A}, S_2, \Pi_2)$ Grammatiken vom Typ $i, i = 0, 1, 2, 3$ mit $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$, $S_j = \mathcal{L}(G_j), j = 1, 2$.

(Vereinigung)

$\Pi := \Pi_1 \cup \Pi_2 \cup \{S \to S_1 | S_2\}$
$\mathcal{V} := \mathcal{V}_1 \cup \mathcal{V}_2 \cup \{S\}$
Dann ist $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ vom Typ i und $\mathcal{L}(G) = S_1 \cup S_2$

(Konkatenation für Typ 3)

$\Pi := \Pi_1 \cup \{A \to aS_2 | A \to a \in \Pi_1\} \cup \Pi_2 \setminus \{A \to aS_2 | A \to a \in \Pi_2\}$
$\mathcal{V} := \mathcal{V}_1 \cup \mathcal{V}_2$
Dann ist $G = (\mathcal{V}, \mathcal{A}, S_1, \Pi)$ rechtslinear und $\mathcal{L}(G) = S_1 \cdot S_2$

(Konkatenation für Typ 0, 1, 2)

$\Pi := \{S \to S_1 \cdot S_2\} \cup \Pi_1 \cup \Pi_2$ ist nicht gut genug, weil die Trennung zwischen S_1 und S_2 unklar ist. Daher Umbenennung:

$$\begin{aligned}
\mathcal{A}_1 &:= \mathcal{A} \quad \text{nach Umbenennung der } a \in \mathcal{A} \text{ in } a_1 \text{ bzw. } a_2, \\
\mathcal{A}_2 &:= \mathcal{A} \quad \text{nach Umbenennung der } a \in \mathcal{A} \text{ in } a_1 \text{ bzw. } a_2, \\
\Pi_1 &:= \Pi_1 \quad \text{nach Umbenennung der } a \in \mathcal{A} \text{ in } a_1 \text{ bzw. } a_2, \\
\Pi_2 &:= \Pi_2 \quad \text{nach Umbenennung der } a \in \mathcal{A} \text{ in } a_1 \text{ bzw. } a_2.
\end{aligned}$$

Setze

$$\Pi := \{S \to S_1 S_2\} \cup \Pi_1 \cup \Pi_2 \cup \{a_1 \to a | a \in \mathcal{A}\} \cup \{a_2 \to a | a \in \mathcal{A}\}$$

und

$$\mathcal{V} := \mathcal{V}_1 \cup \mathcal{V}_2 \cup \mathcal{A}_1 \cup \mathcal{A}_2$$

Dann ist

$$G \quad := \text{vom Typ } i \text{ mit } \mathcal{L}(G) = S_1 \cdot S_2.$$

(Sternoperator für rechtslineare Grammatiken)

$\Pi := \Pi_1 \cup \{A \to aS_1 | A \to a \in \Pi_1\} \cup \{S_1 \to \lambda\}$
$\mathcal{V} := \mathcal{V}_1 \cup \mathcal{V}_2$
Dann ist $G = (\mathcal{V}, \mathcal{A}, S_1, \Pi)$ rechtslinear und $\mathcal{L}(G) = S_1^*$

(Sternoperator für Typ 0, 1, 2)

wie bei Konkatenation mit $\Pi := \{S \to S_1 | S_1 S_2 | S_1 S_2 S_1 | \lambda, S_2 \to S_2 S_1 S_2\} \cup \Pi_1 \cup \Pi_2$, wobei $S_1 = S_2$ sein soll.

Kapitel 2

Reguläre Sprachen

2.1 Deterministische endliche Akzeptoren

Ein *deterministischer, endlicher Akzeptor* (DFA, endlicher Automat) ist ein Quintupel $M := (Q, \mathcal{A}, q_0, \delta, \mathcal{F})$ bestehend aus

A	Eingabealphabet,
Q	Zustandsmenge,
q_0	Startzustand,
$\mathcal{F} \subset Q$	Menge der Endzustände,
$\delta : Q \times \mathcal{A} \to Q$	Übergangsfunktion.

Diese Übergangsfunktion kann partiell oder total sein (d.h. auf einen Teil von $Q \times \mathcal{A}$ oder ganz $Q \times \mathcal{A}$ definiert). M heißt insbesondere *vollständig*, falls δ total ist.

Beispiel

(siehe Abbildung 2.1)
$Q := \{q_0, q_1\}, \quad \mathcal{F} := \{q_0\}$

δ	0	1
q_0	q_1	q_1
q_1	q_1	q_0

Die von M akzeptierte Sprache ist $\mathcal{L}(M) = (\{0^n1 | n \geq 1\} \cup \{10^n1 | n \geq 0\})^*$, kurz $((0 \times 1)0^*1)^*$.

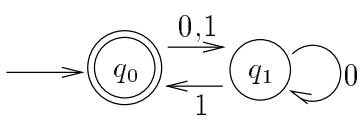


Abbildung 2.1: Deterministischer endlicher Automat (DFA)

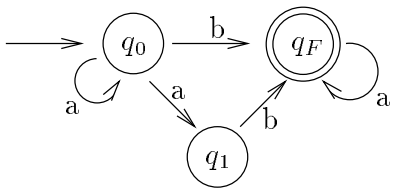


Abbildung 2.2: Nichtdeterministischer finaler Automat (NFA)

Von M akzeptierte Sprache

$\mathcal{L}(M) := \{\omega | \delta^*(q_0, \omega) \in \mathcal{F}\}$,
wobei $\delta^*(q, \lambda) = q$ und $\delta^*(q, \omega a) = \delta(\delta^*(q, \omega), a)$

Satz

Ist M ein DFA, dann ist $\mathcal{L}(M)$ rechtslinear.

Beweis

$\Pi := \{q \to ap | \delta(q, a) = p\} \cup \{q \to a | \delta(q, a) \in \mathcal{F}\}$ erzeugt $\mathcal{L}(M)$.

2.2 Nichtdeterministische endliche Automaten

Ziel: Umkehrung des letzten Satzes

Ein *nichtdeterministischer endlicher Automat* (NFA) ist ein Automat $M := (Q, \mathcal{A}, q_0, \delta, \mathcal{F})$ mit nichtdeterministischer Übergangsfunktion $\delta : Q \times \mathcal{A} \to 2^Q =:$ Potenzmenge von Q. Die anderen Eigenschaften sind wie bei einem DFA. Der NFA akzeptiert die Sprache $\mathcal{L}(M) := \{\omega \in \mathcal{A}^* | \delta^*(q_0, \omega) \cap \mathcal{F} \neq \emptyset\}$, wobei $\delta^*(q, \omega)$ =Menge aller Zustände, die von q bei Eingabe von ω erzeugt werden können.

Beispiel 1

(siehe Abbildung 2.2)
Die entsprechende rechtslineare Grammatik ist

$$\begin{aligned} q_0 &\to aq_0 | bq_F | b \mid aq_1 \\ q_1 &\to bq_F | b \\ q_F &\to aq_F \mid a \end{aligned}$$

Beispiel 2

Rechtslineare Grammatik:

$$\begin{aligned} S &\to aS | aA | bB \\ A &\to bB | b \\ B &\to aB | a \end{aligned}$$

Der entsprechende Automat ist unter Abbildung 2.3 zu finden.

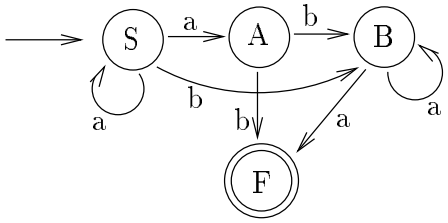


Abbildung 2.3: NFA, Zweites Beispiel

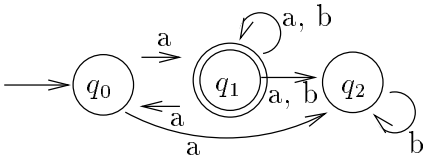


Abbildung 2.4: Ein nichtdeterministischer Automat

Folgerung

Zu jeder rechtslinearen Grammatik G gibt es einen NFA M mit $\mathcal{L}(M) = \mathcal{L}(G)$ und umgekehrt.

Satz

Zu jedem NFA $M = (Q, \mathcal{A}, q_0, \delta, \mathcal{F})$ gibt es einen DFA $M' = (Q', \mathcal{A}, q_0', \delta', \mathcal{F}')$ mit $\mathcal{L}(M) = \mathcal{L}(M')$.

Beweis

$$\begin{aligned} Q' &:= 2^Q \\ \mathcal{F}' &:= \{p \in Q | p \cap \mathcal{F} \neq \emptyset\} \\ \delta'(\{q_1, \ldots, q_n\}, a) &:= \delta(q_1, a) \cup \cdots \cup \delta(q_n, a) \end{aligned}$$

Beispiel (siehe Abbildung 2.4)

δ'	a	b
p_0	p_1	p_2
p_1	p_3	p_1
p_2	p_2	p_2
p_3	p_3	p_1

$$\begin{aligned} p_0 &:= \{q_0\} \\ p_1 &:= \{q_1, q_2\} \\ p_2 &:= \emptyset \\ p_3 &:= Q = \{q_0, q_1, q_2\} \\ \mathcal{F} &:= \{p_1, p_3\} \end{aligned}$$

Der neue Akzeptor ist damit $M' := (\{p_0, p_1, p_2, p_3\}, \mathcal{A}, p_0, \mathcal{F})$ (siehe auch Abbildung 2.5)

Folgerung

Jede Sprache vom Typ 3 wird durch einen DFA erkannt und umgekehrt.

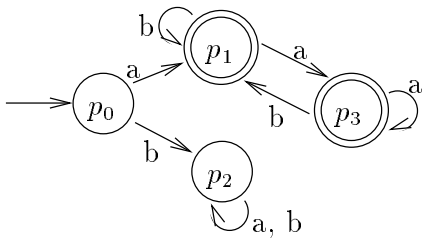


Abbildung 2.5: Deterministischer Automat zum Automaten aus Abbildung 2.4

2.3 Das Pump-Lemma

Pump-Lemma

Sei M ein endlicher Automat mit n Zuständen. Dann läßt sich jedes Wort $\omega \in \mathcal{L}(M)$, $|\omega| \geq n$ so zerlegen,

$$\omega = \alpha\beta\gamma,$$

daß

$$\begin{aligned} &|\alpha\gamma| < n, \\ &1 \leq |\beta|, \\ &\alpha\beta^*\gamma \subset \mathcal{L}(M). \end{aligned}$$

Beweis

Sei $\omega = a_1, \ldots, a_m$, wobei $m \geq n$,

$$Q = \{q_0, \ldots, q_{n-1}\}$$

undd

$$q_0 \overset{a_1}{\to} q_1 \overset{a_2}{\to} \ldots \overset{a_r}{\to} q_r \overset{a_{r+1}}{\to} \ldots \overset{a_s}{\to} q_s \to \ldots \overset{a_m}{\to} q_m.$$

$$a_{r+1} \ldots a_s$$

Wegen $m \geq n$ existiert $r \neq s$, so daß $q_r = q_s$ und $r + 1 + (m - s) \leq n$. Die Behauptung folgt also für

$$\begin{aligned} \alpha &= a_1 \ldots a_r, \\ \beta &= a_{r+1} \ldots a_s, \\ \gamma &= a_{s+1} \ldots a_m. \end{aligned}$$

Siehe auch `http://i12www.ira.uka.de/~pape/Animation`

Beispiel

$\mathcal{L} := \{a^n b^n \mid n \in \mathbb{N}\}$ ist nicht vom Typ 3, denn $\forall n \in \mathbb{N}$ gilt:
Ist $a^n b^n = \alpha\beta\gamma$ mit $\beta = a^i b^j \neq \lambda$, dann ist $\alpha\beta^2\gamma = a^{n-i}a^i b^j a^i b^j b^{n-j} = a^n b^j a^i b^n \notin \mathcal{L}$.

Folgerung

Es gibt kontextfreie Sprachen, die nicht vom Typ 3 sind, siehe Seite 10, Abschnitt 1.2, Beispiel 1

2.4 Reguläre Ausdrücke

R ist ein *regulärer Ausdruck* über $\mathcal{A}$, falls

$$\begin{aligned} R &= \emptyset, \\ R &= \lambda, \\ R &= a,\ a \in \mathcal{A}, \\ R &= (R_1 + R_2), \\ R &= R_1 \cdot R_2, \text{ oder} \\ R &= (R_1)^*, \end{aligned}$$

wobei R_1 und R_2 reguläre Ausdrücke sind.

Beispiele regulärer Ausdrücke

$$\begin{aligned} &a^* \\ &(a+b)^* \cdot (ab)^* \\ &(aa + bbb)^* \end{aligned}$$

Reguläre Sprachen

Ersetzt man in einem regulären Ausdruck

$$\begin{aligned} \lambda &\text{ durch } \{\lambda\}, \\ a &\text{ durch } \{a\} \text{ und} \\ + &\text{ durch } \cup, \end{aligned}$$

so erhält man eine Sprache über $\mathcal{A}$. Diese Sprachen heißen *regulär*.

Folgerung (aus den Abschlußeigenschaften 1.2)

Reguläre Sprachen sind vom Typ 3.

Umkehr

Sei $M = (Q, \mathcal{A}, \delta, q_0, \mathcal{F})$ ein DFA. Dann ist $\mathcal{L}(M)$ regulär.

Beweis

Sei

$$\begin{aligned} Q &:= \{q_0, \ldots, q_n\} \\ R_{ij} &:= \{\omega \in \mathcal{A}^* \mid \delta^*(q_i, \omega) = q_j\} \\ R_{ij}^k &:= \{\omega \in R_{ij} \mid \text{Der Pfad } q_i \overset{\omega}{\to} q_j \text{ führt über} \\ &\qquad \text{keinen der Zustände } q_k, \ldots, q_n\} \end{aligned}$$

Also ist

$$R_{ij}^{n+1} = R_{ij}$$

und

$$\mathcal{L}(M) = \bigcup_{q_j \in \mathcal{F}} R_{0j}.$$

Das heißt, es genügt zu zeigen, daß R_{ij}^k regulär ist.
Nun folgt der eigentliche *Beweis* durch vollständige Induktion über k:

- $R_{ij}^0 = \{a \mid (q_i, a) = q_j\}$ ist regulär, da R_{ij}^0 endlich ist.
- $R_{ij}^{k+1} = R_{ij}^k \cup R_{ik}^k \left(R_{kk}^k\right)^* R_{kj}^k$ ist regulär, wenn alle $R_{...}^k$ regulär sind.

Zusammenfassung

{rechtslineare Sprachen}={reguläre Sprachen}

Ebenso

{linkslineare Sprachen}={reguläre Sprachen}

2.5 Eine Äquivalenzrelation

Sei $\mathcal{L}$ eine Sprache über $\mathcal{A}$ und $\sim_\mathcal{L}$ folgende Äquivalenzrelation auf $\mathcal{A}^* : \alpha \sim_\mathcal{L} \beta$ genau dann, wenn $\forall \gamma \in \mathcal{A}^* : \alpha\gamma \in \mathcal{L} \Leftrightarrow \beta\gamma \in \mathcal{L}$.

Prüfe (Reflexivität, Symmetrie, Transitivität)

Bemerkung

$\forall \alpha \in \mathcal{L}$:Äquivalenzklasse $[\alpha] \subset \mathcal{L}$, denn $\alpha \sim_\mathcal{L} \beta \Rightarrow \alpha\gamma$ und $\beta\gamma = \beta \in \mathcal{L}$.

Satz

$\mathcal{L}$ ist eine reguläre Sprache genau dann, wenn Index $\sim_\mathcal{L}$ endlich ist (Index $\sim_\mathcal{L}$=Zahl der Äquivalenzklassen)

Beweis

„⇒“: Sei $M = (Q, \mathcal{A}, \delta, q_0, \mathcal{F})$ ein vollständiger DFA mit $\mathcal{L}(M) = \mathcal{L}$. Sei $\sim_M$ folgende Äquivalenzrelation: $\alpha \sim_M \beta :\Leftrightarrow \delta^*(q_0, \alpha) = \delta^*(q_0, \beta)$. Dann gilt

$$\begin{aligned} \alpha \sim_\mathcal{M} \beta &\Leftrightarrow \forall \gamma \in \mathcal{A}^* : \alpha\gamma \sim_M \beta\gamma \\ &\Rightarrow \alpha \sim_\mathcal{L} \beta. \end{aligned}$$

Also gilt: Index $\sim_\mathcal{L} \leq$ Index $\sim_\mathcal{M} \leq |Q| < \infty$

„⇐“: Seien $[\omega_1], \ldots, [\omega_k]$ die Äquivalenzklassen von $\sim_{\mathcal{L}}$.
Setze

$$M := (Q, \mathcal{A}, \delta, q_0, \mathcal{F}),$$

wobei

$$\begin{aligned} Q &:= \{[\omega_1], \ldots, [\omega_k]\} \\ \delta([\omega], a) &:= [\omega_a] \\ q_0 &:= [\lambda] \\ \mathcal{F} &:= \{[\omega] \mid [\omega] \subset \mathcal{L}\}. \end{aligned}$$

Dann gilt:

$$\begin{aligned} \omega \in \mathcal{L}(M) &\Leftrightarrow \delta^*([\lambda], \omega) \in \mathcal{F} \\ &\Leftrightarrow [\omega] \in \mathcal{F} \\ &\Leftrightarrow [\omega] \subset \mathcal{L} \\ &\overset{\text{Beweis oben}}{\Leftrightarrow} \omega \in \mathcal{L} \\ \alpha \sim_{\mathcal{L}} \beta &\Leftrightarrow \forall \gamma \in \mathcal{A}^* : \alpha\gamma \in \mathcal{L} \\ &\Leftrightarrow \beta\gamma \in \mathcal{L}. \end{aligned}$$

Beispiel

$\mathcal{L} = \{a^n b^n \mid n \geq 1\}$

Einige Äquivalenzklassen von $\mathcal{L}$:

$$\begin{aligned} [\lambda] &= \{\lambda\}, \text{ denn } \alpha ab \in \mathcal{L} \Leftrightarrow \alpha = \lambda, \\ [a] &= \{a\}, \text{ denn } \alpha ab^2 \in \mathcal{L} \Leftrightarrow \alpha = a, \\ [a^2] &= \{a^2\}, \text{ denn } \alpha ab^3 \in \mathcal{L} \Leftrightarrow \alpha = a^2, \\ [a^i] &= \{a^i\}. \end{aligned}$$

Also ist Index $\sim_{\mathcal{L}} = \infty$. D.h. $\mathcal{L}$ ist nicht regulär.

Beispiel

$\mathcal{L} := \{\omega 00 \mid \omega \in \{0,1\}^*\}$

Die Äquivalenzklassen von $\mathcal{L}$:

$$\begin{aligned} [\lambda] &= \{\omega \in \{0,1\}^* \mid \omega \text{ endet nicht mit einer } 0\}, \\ [0] &= \{\omega \in \{0,1\}^* \mid \omega \text{ endet mit einer } 0, \text{ aber nicht mit zwei Nullen}\}, \\ [00] &= \{\omega \in \{0,1\}^* \mid \omega \text{ endet mit zwei Nullen}\}. \end{aligned}$$

Also ist $\mathcal{L}$ regulär, denn Index $\sim_{\mathcal{L}} = 3 < \infty$. Siehe auch den zugehörigen Automaten (Abbildung 2.6).

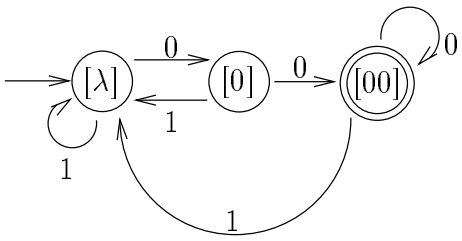


Abbildung 2.6: Automat aus den Äquivalenzklassen der Sprache $\mathcal{L}$

2.6 Der Minimalautomat

Folgerungen (aus 2.5)

- Jeder vollständige DFA, der eine reguläre Sprache $\mathcal{L}$ akzeptiert, hat mindestens $n =$ Index $\sim_{\mathcal{L}}$ Zustände.
- Der Äquivalenzklassenautomat ist minimal unter dem vollständigen Automaten.
- Für jeden minimalen vollständigen Automaten M sind $\sim_{\mathcal{M}}$ und $\sim_{\mathcal{L}}$ gleich.
- Alle minimalen vollständigen Automaten für $\mathcal{L}$ sind bis auf Umbenennungen gleich.

Jeder endliche Automat kann durch Verschmelzen von Zuständen minimiert werden. Zwei Zustände p und q können genau dann verschmolzen werden, wenn $\forall \omega \in \mathcal{A}^* : \delta^*(p, \omega) \in \mathcal{F} \Leftrightarrow \delta^*(q, \omega) \in \mathcal{F}$.

Ein Minimierungsalgorithmus

Gegeben: Ein endlicher Automat M

1. Entferne alle Zustände, die nicht von q_0 aus erreichbar sind bei Eingabe beliebiger Werte.
2. Markiere alle $(p, q) \in \mathcal{F} \times \overline{\mathcal{F}} \cup \overline{\mathcal{F}} \times \mathcal{F}$, wobei $\overline{\mathcal{F}} = Q - \mathcal{F}$[1].
3. Markiere weiter alle $(p, q) \in Q^2$, für die ein $a \in \mathcal{A}$ existiert, so daß $(\delta(p, a), \delta(q, a))$ markiert ist.
4. Wiederhole 3., bis keine Markierungen mehr hinzukommen.
5. Verschmelze alle unmarkierten Paare.

Beispiel

siehe Abbildungen 2.7, 2.8, 2.7. Abbildung 2.7 zeigt einen endlichen Automat. Abbildung 2.8 veranschaulicht den Minimierungsalgorithmus und Abbildung 2.7 zeigt den zugehörigen Minimalautomat. Die akzeptierte Sprache ist $\{\alpha 00\beta \mid \alpha, \beta \in \{0, 1\}^*\}$. Die Zustände q_0q_2, q_1q_3 und q_4 entsprechen den Äquivalenzklassen $[\lambda]$, $[0]$ und $[00]$.

[1] Komplement

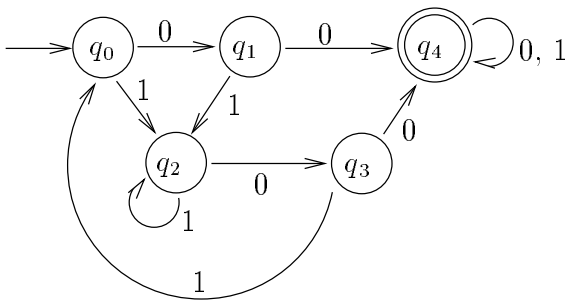


Abbildung 2.7: Nicht-minimierter Automat

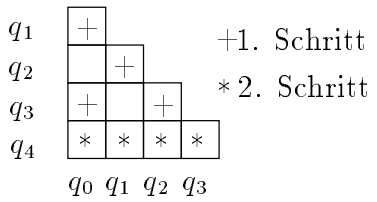


Abbildung 2.8: Tabelle: Minimierung des Automaten M aus Abbildung 2.7

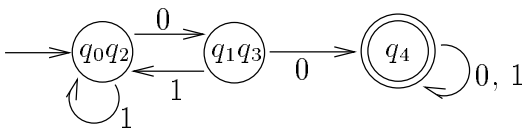


Abbildung 2.9: Minimalautomat (zu Abbildung 2.7)

2.7 Abschlußeigenschaften

Zur Erinnerung: Reguläre Sprachen sind abgeschlossen unter $\cup, \cdot, *$.

Außerdem

sind sie abgeschlossen bezüglich Komplement und Schnitt.

Beweis

- Die vollständigen DFAs $(Q, \mathcal{A}, \delta, q_0, \mathcal{F})$ und $(Q, \mathcal{A}, \delta, q_0, \overline{\mathcal{F}})$ akzeptieren komplementäre Sprachen.
- de Morgan: $\mathcal{L} \cap \mathcal{K} = \overline{\overline{\mathcal{L}} \cup \overline{\mathcal{K}}}$. Also ist auch der Schnitt regulär.

2.8 Entscheidbare Probleme

Sind reguläre Sprachen $\mathcal{L}$, $\mathcal{L}_1$, $\mathcal{L}_2$ durch Automaten oder rechtslineare Grammatiken gegeben, kann man Folgendes algorithmisch beantworten:

1. Wortproblem: $\omega \in \mathcal{L}$?
2. Leerheitsproblem: $\mathcal{L} = \emptyset$?
3. Gleichheit: $\mathcal{L}_1 = \mathcal{L}_2$?
4. Schnittproblem: $\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset$?
5. Endlichkeitsproblem: $|\mathcal{L}| = \infty$?

Beweis

1. $\surd$
2. Gibt es einen Endzustand, der von q_0 aus erreichbar ist?
3. Vergleiche Minimalautomaten.
4. Seien $M_i = (Q_i, \mathcal{A}, \delta_i, q_i, \mathcal{F}_i)$, $i = 1, 2$ endliche Automaten. Der *Produktautomat*

$$M_1 \times M_2 := Q_1 \times Q_2, \mathcal{A}, \delta_1 \times \delta_2, (q_1, q_2), \mathcal{F}_1 \times \mathcal{F}_2),$$

wobei

$$(\delta_1 \times \delta_2)((p, q), a) := (\delta_1(p, a), \delta_2(q, a)),$$

akzeptiert $\mathcal{L}(M_1) \cap \mathcal{L}(M_2)$.

5. $$\begin{aligned} |\mathcal{L}| = \infty \quad &\Leftrightarrow \quad \text{es existiert } \alpha\beta^*\gamma \subset \mathcal{L} \\ &\qquad\quad \text{mit } |\alpha\gamma| < n \\ &\qquad\quad \text{und } 1 \leq |\beta| < n = |Q|. \\ &\Leftrightarrow \quad \text{es existiert ein } \omega \in \mathcal{L},\ n \leq |\omega| < 2n. \end{aligned}$$

Kapitel 3

Kontextfreie Sprachen

3.1 Syntaxbäume

$G := (\mathcal{V}, \mathcal{A}, S, \pi)$ sei kontextfreie Grammatik; zur Erinnerung: $\Pi \subset \mathcal{V} \times (\mathcal{A} \cup \mathcal{V})^*$

Beispiel

$$\begin{aligned} S &\rightarrow 0A \\ A &\rightarrow 0AA \mid 1 \end{aligned}$$

Die kontextfreie Ableitung $S \rightarrow 0A \Rightarrow 00AA \Rightarrow 000AAA \stackrel{*}{\Rightarrow} 000111$ hat den in Abbildung 3.1 dargestellten *Syntax-* oder *Ableitungsbaum.*
Allgemein gilt für einen Ableitungsbaum das folgende:

- Wurzel $= S$
- Blätter $\in \mathcal{A}$
- sonstige Knoten $\in \mathcal{V}$
- Ist A ein Teilbaum (siehe Abbildung 3.2), dann ist $A \rightarrow k_1 \ldots k_n \in \Pi$

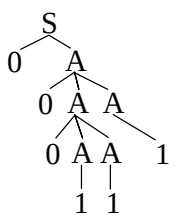


Abbildung 3.1: Teilbaum

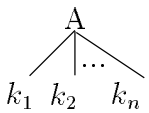


Abbildung 3.2: Teilbaum

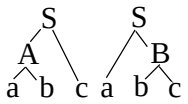


Abbildung 3.3: Beispiel für mehrdeutige Grammatik

Achtung

Im allgemeinen gibt es zu einem Ableitungsbaum mehrere Ableitungen, zum Beispiel

$$S \Rightarrow 0A \Rightarrow 00AA \Rightarrow 000AAA \Rightarrow \begin{cases} 0001AA \\ 000A1A \\ 000AA1 \end{cases} \stackrel{*}{\Rightarrow} 000111$$

Links-(Rechts)ableitung

Die am weitesten links (rechts) stehende Variable wird in jedem Ableitungsschritt zuerst ersetzt.

Bemerkung

Eine kontextfreie Grammatik heißt mehrdeutig, wenn es zu einem Wort mehrere Ableitungsbäume gibt, zum Beispiel wie in Abbildung 3.3.

3.2 Das Pump-Lemma

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine kontextfreie Grammatik,

$$\begin{aligned} m &:= \max\{|\alpha| \mid \alpha \text{ ist rechte Seite einer Produktion}\} \\ \mathcal{L} &:= \mathcal{L}(G) \end{aligned}$$

Lemma

Hat $\omega \in \mathcal{L}$ einen Ableitungsbaum der Höhe h, dann ist $|\omega| \leq m^h$.

Beweis

$\surd$

Pump-Lemma

Es gibt $n \in \mathbb{N}$, so daß für alle $\omega \in \mathcal{L}$, $|\omega| \geq n$, eine Zerlegung

$$\omega = \alpha\beta\gamma\delta\epsilon$$

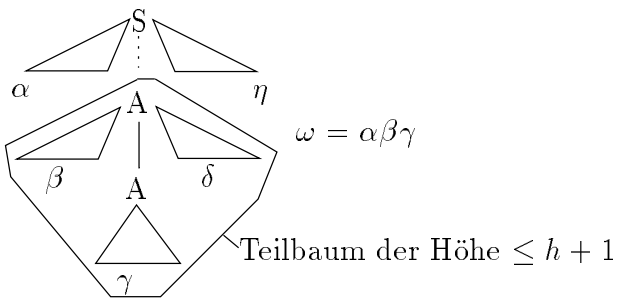


Abbildung 3.4: Ableitungsbaum von ω

existiert mit

- $|\beta\gamma\delta| \leq n$
- $\beta\delta \neq \lambda$
- $\alpha\beta^i\gamma\delta^i\epsilon \in \mathcal{L}$ für alle $i \geq 1$.

Beweis

Sei $h := |\mathcal{V}|, \omega \in \mathcal{L}, |\omega| \geq m^h$. Dann hat jeder Ableitungsbaum von ω eine Höhe $\geq h+1$. Jeder Pfad von der Wurzel zu einem Blatt der Länge $h+1 = |\mathcal{V}|+1$ hat eine Variable doppelt, anschaulich in Abbildung 3.4 zu sehen.
Lemma $\Rightarrow |\beta\gamma\delta| \leq m^{h+1}$ und offensichtlich

$$\begin{aligned} S &\stackrel{*}{\Rightarrow} \alpha\beta A\delta\epsilon \\ &\stackrel{*}{\Rightarrow} \alpha\beta^i A\delta^i\epsilon \\ &\stackrel{*}{\Rightarrow} \alpha\beta^i\gamma\delta^i\epsilon \end{aligned}$$

Bemerkung

Falls $\beta\delta = \lambda$, dann $A \stackrel{*}{\Rightarrow} A$ und man kann den Baum verkleinern.

Beispiel

$\mathcal{L} := \{a^n b^n c^n \mid n \geq 1\}$ ist nicht kontextfrei, denn für alle $n \in \mathbb{N}$ gilt:
$a^n b^n c^n = \alpha\beta\gamma\delta\epsilon$ mit $|\beta\gamma\delta| \leq n$ und $\beta\delta \neq \lambda$

$$\Rightarrow \quad \beta\gamma\delta \in a \text{ oder } c \text{ ist nicht Teil von } \beta\gamma\delta, \text{ d.h. } \alpha\beta^2\gamma\delta^2\epsilon \notin \mathcal{L}.$$

Also ist $\mathcal{L}$ nicht kontextfrei, weil das Pumping-Lemma nicht erfüllt ist.

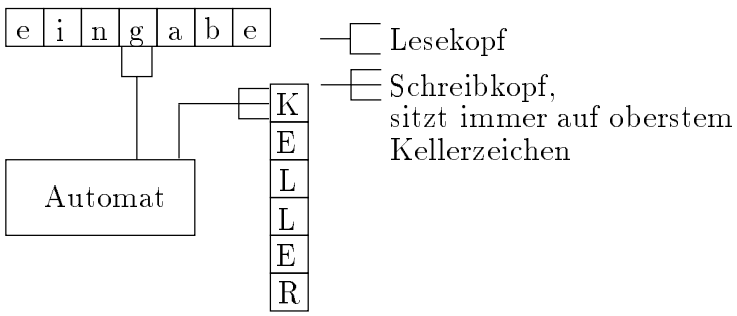


Abbildung 3.5: PDA (Push-down-Automat, Kellerautomat)

Bemerkung

$\mathcal{L}$ ist kontextsensitiv, denn

$$\begin{aligned} S &\rightarrow aBC \mid aSBC \\ CB &\rightarrow BC \\ aB &\rightarrow ab \\ bB &\rightarrow bb \\ bC &\rightarrow bc \\ cC &\rightarrow cc \end{aligned}$$

erzeugt $\mathcal{L}$.

Folgerung

Es gibt kontextsensitive Sprachen, die nicht kontextfrei sind.

3.3 Einfache Kellerautomaten

Ziel: Automaten für kontextfreie Sprachen

Wie: endliche Automaten mit Keller

Ein *einfacher Kellerautomat* (PDA, siehe Abbildung 3.5) ist ein Quartupel $M = (\mathcal{A}, \mathcal{K}, \delta, S)$ bestehend aus

$\mathcal{A}$	dem Eingabealphabet
$\mathcal{K}$	dem Kelleralphabet
δ	der Übertragungsfunktion : $\mathcal{A} \times \mathcal{K} \rightarrow$ {endliche Teilmengen von $\mathcal{K}^*$}
$S \in \mathcal{K}$	dem Kellerzeichen

Bemerkung

Ein allgemeiner Kellerautomat kann mehrere Zustände haben, einfache Kellerautomaten haben nur einen bzw. keinen.

Bedeutung von $\delta(g, K) \ni B_1 \dots B_k$

Ist g aktuelles Eingabezeichen und K oberstes Kellerzeichen, dann kann der Automat folgendes tun:

- er rückt Lesekopf um eine Position nach rechts (oder läßt Lesekopf stehen, falls $a \in \lambda$)
- und ersetzt das oberste Kellerzeichen durch $B_1 \dots B_k$ (Ist $k = 0$, ist $B_1 \dots B_k = \lambda$).

Wir schreiben

$$aA \quad \to \quad B_1 \dots B_k$$

für

$$\delta(a, A) \quad \ni \quad B_1 \dots B_k$$

und

$$a\omega \cdot AW \quad \Rightarrow \quad \omega \cdot B_1 \dots \omega \cdot B_k W$$

falls

$$\begin{aligned} aA &\to B_1 \dots B_k \\ \omega &\in \mathcal{A}^* \\ W &\in \mathcal{K}^* \end{aligned}$$

$\omega \cdot W \in \mathcal{A}^* \cdot \mathcal{K}^*$ heißt *Konfiguration*

$\omega \cdot S \in \mathcal{A}^* \cdot \{S\}$ heißt *Startkonfiguration*

Beispiel

$$\begin{aligned} aA \to \lambda &\quad \text{bedeutet} \quad ''\text{pop}'' \text{ A} \\ aA \to BA &\quad \text{bedeutet} \quad ''\text{push}'' \text{ B} \end{aligned}$$

Akzeptierte Sprache

$\mathcal{L}(M) := \{\omega \in \mathcal{A}^* \mid \omega \cdot S \overset{*}{\Rightarrow} \lambda \cdot \lambda\}$

Bemerkung

Ein PDA ist im allgemeinen indeterministisch.

3.4 Einfache Kellerautomaten und kontextfreie Sprachen

Satz 1

Jede kontextfreie Sprache wird durch einen Kellerautomaten akzeptiert.

Beweis

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine kontextfreie Grammatik. Setze $\mathcal{K} := \mathcal{A} \cup \mathcal{V}$ und δ so, daß M folgende Übergänge hat:

$$\begin{array}{cccl} & \lambda A & \to W & \text{für alle } A \to W \in \Pi \\ a & a & \to \lambda & \text{für alle } a \in \mathcal{A} \\ \text{lies} & \text{pop} & & \end{array}$$

Dann ist

$$\mathcal{L}(G) = \mathcal{L}(M)$$

mit $M = (\mathcal{A}, \mathcal{K}, \delta, S)$, denn jede Konfiguration ωW von M besteht aus einer Eingabe $\alpha\omega$ und einer aus S in G ableitbaren Satzform αW vermindert um einen gemeinsamen Anfang α.

Satz 2

Jeder Kellerautomat M akzeptiert eine kontextfreie Sprache.

Beweis

Setze $\Pi := \left\{ A \to^{\text{erzeuge}}_{a} W \mid^{\text{lies}}_{a} A \to W \text{ ein Übergang von } M \right\}$. Dann ist

$$\mathcal{L}(M) = \mathcal{L}(G)$$

mit $G = (\mathcal{K}, \mathcal{A}, S, \Pi)$.

3.5 Allgemeine Kellerautomaten

Ein (allgemeiner) *Kellerautomat* (PDA) ist ein Tupel $M := (Q, \mathcal{A}, \mathcal{K}, \delta, q_0, S)$, bestehend aus

Q		Zustandsmenge
$\mathcal{A}$		Eingabealphabet
$\mathcal{K}$		Kelleralphabet
δ	:	$\mathcal{A}_\lambda \times Q \times \mathcal{K} \to \{\text{endliche Teilmenge von } Q \times \mathcal{K}^*\}$
		Übergangsfunktion
q_0		Anfangszustand
S		Startzeichen

Bemerkung

Ein PDA ist ein NFA mit Keller und λ-Übergängen. Für $\delta(a, q, A) \ni (p, B_1 \ldots B_k)$ schreiben wir:

$$a \cdot q \cdot A \to p \cdot B_1 \ldots B_k$$

Die

$$\omega \cdot q \cdot W \in \mathcal{A}^*_\lambda \cdot Q \cdot \mathcal{K}^*$$

heißen *Konfigurationen* von M und die

$$\omega \cdot q_0 \cdot S \in \mathcal{A}_\lambda^* \cdot \{q_0 \cdot S\}$$

heißen *Starkonfigurationen* von M.
Zu jedem $\delta-$Übergang $a{\cdot}q{\cdot}A \to p{\cdot}B_1 \dots B_k$ gibt es die Konfigurationsübergänge

$$a\omega \cdot q \cdot AW \Rightarrow \omega \cdot p \cdot B_1 \dots B_k W$$

Akzeptierte Sprache

$\mathcal{L}(M) := \{\omega \in \mathcal{A}^* \mid \omega \cdot q_0 \cdot S \stackrel{*}{\Rightarrow} \lambda \cdot q \cdot \lambda, q \in Q\}$

Satz

M ist ein (allgemeiner) Kellerautomat $\Rightarrow$ $\mathcal{L}(M)$ ist kontextfrei.

Beweis

$$\begin{aligned}
\mathcal{V} &:= \{{}^pA^q \mid A \in \mathcal{K},\, p,q \in Q\} \cup \{S\} \\
\Pi &:= \{{}^qA^{p_{k+1}} \to a\,{}^pB_1^{p_1}\,{}^{p_1}B_2^{p_2} \dots {}^{p_{k-1}}B_k^{p_k} \mid \\
&\quad aqA \stackrel{\delta}{\to} pB_1 \dots B_k \mid k \geq 1,\, p_1, \dots p_k \in Q\} \\
&\quad \cup \{{}^qA^p \to a \mid aqA \to p \cdot \lambda\} \\
&\quad \cup \{S \to {}^{q_0}S^q \mid q \in Q\}
\end{aligned}$$

Die Grammatik $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ ist kontextfrei und jede aus S linksabgeleitete Satzform hat die Gestalt $a_1 \dots a_m\,{}^qA_1^{q+1}\,{}^{q_1}A_2^{q_2} \dots {}^{q_{n-1}}A_n^{q_n}$. Diese Ableitung entspricht einer aus einer Startkonfiguration abgeleitete Konfiguration

$$a_1 \dots a_m\, \alpha \cdot q_0 \cdot S \stackrel{*}{\Rightarrow} \alpha \cdot q \cdot A_1 \dots A_n$$

des Automaten M mit $\alpha \in \mathcal{A}^*$. Umgekehrt entspricht einem Konfigurationsübergang aus einer Startkonfiguration eine Linksableitung der Grammatik G. Diese Entsprechung ist surjektiv. Also ist $\mathcal{L}(G) = \mathcal{L}(M)$.

Folgerung

$\mathcal{L}(\{\text{allgemeiner PDA}\}) = \mathcal{L}(\{\text{einfacher PDA}\})$

Bemerkung

Man kann zusätzlich Endzustände einführen und definiert dann

$$\mathcal{L}(M) := \{\omega \in \mathcal{A}^* \mid \omega \cdot q_0 \cdot S \Rightarrow \lambda q \gamma,\, q \in \mathcal{F}, \gamma \in \mathcal{K}^*\}.$$

Diese Automaten haben dieselbe Sprachmächtigkeit.

Aber

Deterministische Kellerautomaten mit Endzuständen erkennen nicht alle kontextfreien Sprachen.

3.6 Chomsky Normalform

Eine kontextfreie Grammatik hat *Chomsky Normalform*, falls ihre Produktionen von folgender Form sind:

- eventuell $S \to \lambda$, sofern S auf keiner rechten Seite einer Produktion
- $A \to BC$
- $A \to a$

wobei A, B, $C \in \mathcal{V}$ und $a \in \mathcal{A}$.

Satz

Jede kontextfreie Grammatik kann ohne Änderung der erzeugten Sprache in Chomsky Normalform überführt werden.

Beweis

1. Mache G λ-frei, siehe 1.2
2. Ersetze alle $a \in \mathcal{A}$ durch A_a und einem A_a zu $\mathcal{V}$ und füge $A_a \to a$ zu Π hinzu.
3. Für alle Zyklen $A_1 \to \ldots \to A_k \to A_1$ ersetze $A_2, \ldots, A_k$ durch A_1 (Zyklen können algorithmisch gefunden werden, siehe unten)
4. Entferne alle $A \to A \in \Pi$
5. Gibt es noch eine Produktion $A \to B$, entferne sie und nimm die Produktionen $A \to W$, wobei $B \to W \in \Pi$, hinzu
6. Wiederhole 5., bis sich Π nicht mehr ändert. Dies terminiert, da $A \to B$, wenn einmal entfernt, nicht neu hinzukommt, denn sonst gäbe es Zyklen $B \to C \to \ldots \to Z \to B$.
7. Ersetze alle Produktionen $A \to B_1 \ldots B_k$ durch die Produktionen

$$\begin{aligned} A &\to B_1 C_1 \\ C_1 &\to B_2 C_2 \\ &\vdots \\ C_{k-2} &\to B_{k-1} B_k \end{aligned}$$

   und nimm $C_1, \ldots, C_{k-2}$ zu $\mathcal{V}$ hinzu.

Finden von Zyklen: $A \to A_1 \to \ldots \to A_n \to A$

$\mathcal{M} := \emptyset$

$\mathcal{M}' := \{B \mid A \to B \in \Pi\}$

`while` $\mathcal{M} \neq \mathcal{M}'$ `do`

$\mathcal{M} := \mathcal{M}'$

$\mathcal{M}' := \mathcal{M} \cup \{C \mid B \to C \in \Pi, B \in \mathcal{M}\}$

`if`

$A \in \mathcal{M}$

`then`

Zyklus existiert

`else`

Zyklus existiert nicht

Beispiel

$$\begin{aligned} S &\to AB \\ A &\to a \mid aA \\ B &\to bBc \mid bc \end{aligned}$$

erzeugt $\mathcal{L} = \{a^m b^n c^n \mid m \geq 1, n \geq 1\}$. Umformung auf Chomsky Normalform ergibt:

$$\begin{aligned} S &\to AB \\ A' &\to a \\ B' &\to b \\ C' &\to c \\ A &\to A'A \mid a \\ B &\to B'C' \mid B'D \\ D &\to BC' \end{aligned}$$

3.7 Cocke-Kasami-Younger-Algorithmus

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ kontextfrei in Chomsky Normalform.

Frage Ist $\omega = a_1 \ldots a_n \in \mathcal{L}(G)$?

Der *Cocke-Kasami-Younger-Algorithmus* löst dieses Problem und füllt die Zellen (i, j) einer Pyramide (siehe Abbildung 3.6):
Für $i = 1, \ldots, n$
Für $j = 1, \ldots, n - i + 1$

Schreibe alle $A \in \mathcal{V}$ in Zelle (i, j) mit $A \stackrel{*}{\Rightarrow} a_j \ldots a_{j+i-1}$
(siehe Abbildung 3.7)
Falls $i \geq 2$, heißt das für $m = 1, \ldots, i - 1$:
Schreibe alle A in Zelle (i, j)
mit $A \to BC \in \Pi$, $B \in$ Zelle (m, j) und $C \in$ Zelle $(i - m, j + m)$.

Falls S in Zelle $(n, 1)$, dann ist $\omega \in \mathcal{L}(G)$.
Falls $S \notin$ Zelle $(n, 1)$, dann ist $\omega \notin \mathcal{L}(G)$.

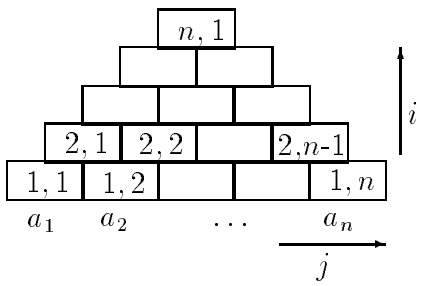


Abbildung 3.6: Cocke-Kasami-Younger-Algorithmus: Pyramide

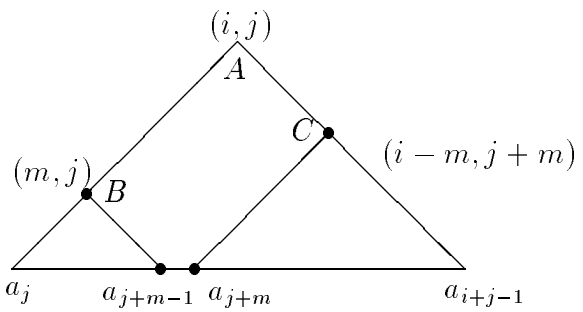


Abbildung 3.7: Dreieck mit A, B, C

Bemerkung 1

Der Cocke-Kasami-Younger-Algorithmus verfolgt eine „bottom-up"-Analyse. Kellerautomaten machen eine „top-down"-Analyse. Sie raten eine Ableitung von S ausgehend.

Beispiel

$$\begin{aligned} S &\rightarrow AB \mid BB \\ A &\rightarrow AB \mid CC \mid a \\ B &\rightarrow CA \mid BB \mid b \\ C &\rightarrow BA \mid AA \mid b \end{aligned}$$

Also gehört *aabb* zu $\mathcal{L}(G)$ (siehe Abbildung 3.8).

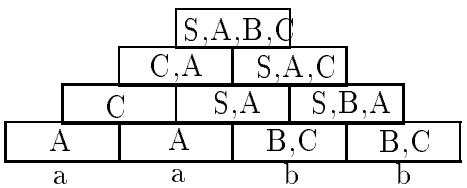


Abbildung 3.8: Ermitteln der Sprachzugehörigkeit von *aabb*

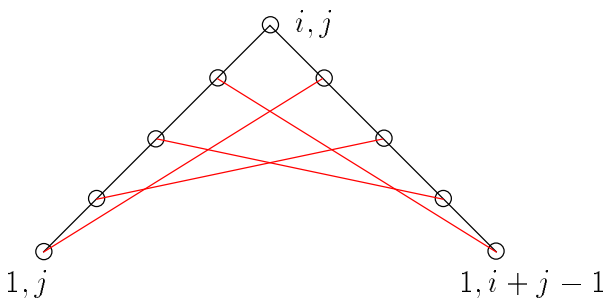


Abbildung 3.9: Zellpaarungen

Bemerkung 2

Um Zelle (i, j) zu füllen, werden $i-1$ Zellenpaare untersucht (siehe Abbildungen 3.7 und 3.9).

Aufwand

Wie oft wird gefragt: „Gibt es Produktionen mit rechter Seite α?“ Sei $k = |V|$. Dann gilt:

Zeile	# Zellen	# untersuchter Zellenpaare	# möglicher rechter Seiten	$\sum$Fragen
1	n	1	1	n
2	$n-1$	1	k^2	$(n-1)\cdot 1\cdot k^2$
3	$n-2$	2	k^2	$(n-2)\cdot 2\cdot k^2$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	1	$n-1$	k^2	$1\cdot(n-1)\cdot k^2$

und somit

$$\begin{aligned} \text{Anzahl aller Fragen} &= n + k^2 \sum_{i=1}^{n} i(n-i) \\ &= n + k^2 (n \sum_{i=1}^{n+1} i - \sum_{i=1}^{n+1} i^2) = O(n^3). \end{aligned}$$

3.8 Abschlußeigenschaften

Zur Erinnerung

Die Menge der regulären Sprachen ist abgeschlossen bezüglich Schnitt und Komplement.

Satz

Die Menge der kontextfreien Sprachen ist *nicht* abgeschlossen bezüglich Schnitt und Komplement.

Beweis (durch Gegenbeispiel)

„Schnitt“ $\mathcal{L}_1 := \{a^m b^n c^n \mid m, n \in \mathbb{N}\}$ ist kontextfrei, denn

$$\begin{aligned} S &\to AB \\ A &\to a \mid aA \\ B &\to bc \mid bBc \end{aligned}$$

erzeugt $\mathcal{L}_1$ (siehe Abschnitt 3.6).

Ebenso ist $\mathcal{L}_2 := \{a^n b^n c^m \mid m, n \in \mathbb{N}\}$ kontextfrei. Aber $\mathcal{L}_1 \cap \mathcal{L}_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$ ist nicht kontextfrei (siehe 3.2).

„Komplement“ (nach de Morgan) :

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}}$$

Also können die kontextfreien Sprachen nicht bezüglich des Komplements abgeschlossen sein.

3.9 Entscheidbare Probleme

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine Grammatik in Chomsky Normalform. Dann sind folgende Probleme in polynomialer Zeit entscheidbar:

1. Wortproblem
2. Leerheit
3. Endlichkeit

Bemerkung

4. Das Schnittproblem ist nicht entscheidbar.
5. Ob die Gleichheit entscheidbar ist, ist unbekannt.

Beweise

1. Cocke-Kasami-Younger-Algorithmus.
2. Folgender Algorithmus bestimmt alle Variablen, aus denen sich Worte ableiten lassen:

$\mathcal{M} := \emptyset$
$\mathcal{M}' := \{A \mid \exists a \in \mathcal{A}_\lambda : A \to a \in \Pi\}$
`while` $\mathcal{M} \neq \mathcal{M}'$ `do`
    $\mathcal{M} := \mathcal{M}'$
    $\mathcal{M}' := \mathcal{M} \cup \{A \mid \exists B, C \in \mathcal{M} : A \to BC \in \Pi\}$
`if`
  $S \in \mathcal{M}$
`then`
  $\mathcal{L}(G) \neq \emptyset$
`else`
  $\mathcal{L}(G) = \emptyset$

Alle Variablen $A \in \mathcal{V} \setminus \mathcal{M}$ sind nutzlos und können entfernt werden.

3. Folgender Algorithmus bestimmt alle Variablen, aus denen sich nur endlich viele Worte ableiten lassen:

   Annahme (OBdA) : Es gibt keine nutzlosen Variablen.
   $\mathcal{M} := \emptyset$, $i = 0$
   $\mathcal{M}' := \{A \mid \exists a \in \mathcal{A} : A \to a \in \Pi\}$
   Aus $A \in \mathcal{M}'$ lassen sich nur Worte der Länge $\leq 2^0$ ableiten.
   `while` $\mathcal{M} \neq \mathcal{M}'$ `do`
   $\mathcal{M} := \mathcal{M}'$, $i := i + 1$
   $\mathcal{M}' := \mathcal{M} \cup \{A \mid \forall B, C \in \mathcal{V} : (A \to BC \in \Pi \Rightarrow B, C \in \mathcal{M})\}$
   Aus $A \subset \mathcal{M}'$ lassen sich nur Worte der Länge $\leq 2^i$ ableiten.
   `if`
   $S \in \mathcal{M}$
   `then`
   $|\mathcal{L}(G)| < \infty$
   `else`
   $|\mathcal{L}(G)| = \infty$

   Ist $S \notin \mathcal{M}$, dann existieren $B, C \in \mathcal{V}$, so daß

   $$S \quad \to \quad BC \in \Pi \text{ und } B \in \mathcal{M} \text{ oder } C \notin \mathcal{M}.$$

   Wenn $S \stackrel{*}{\Rightarrow} \alpha A \beta \in \mathcal{V}^*(\mathcal{V} \setminus \mathcal{M})\mathcal{V}^*$, dann gibt es auch BC mit

   $$S \stackrel{*}{\Rightarrow} \alpha BC\beta \in \mathcal{V}^*(\mathcal{V} \setminus \mathcal{M})\mathcal{V}^*.$$

   Also lassen sich aus S beliebig lange Satzformen und Worte (es gibt keine nutzlosen Variablen) ableiten. Also ist $\mathcal{L}(G)$ unendlich.

Beispiel

$$\begin{aligned} S &\to AD \mid BC \\ B &\to BC \\ D &\to FE \mid b \\ F &\to AD \\ A &\to a \\ E &\to b \mid c \end{aligned}$$

Algorithmus 1

Durchlauf	$\mathcal{M}$
0	$\emptyset$
1	$\{A, D, E\}$
2	$\{A, D, E, F, S\}$
3	$\{A, D, E, F, S\}$

B, C sind nutzlos. D.h. $\mathcal{L}(G) \neq \emptyset$. Die Grammatik ohne Produktionen mit nutzlosen Variablen ist

$$\begin{aligned} S &\to AD \\ D &\to FE \mid b \\ F &\to AD \\ A &\to a \\ E &\to b \mid c \end{aligned}$$

Algorithmus 2

Durchlauf	$\mathcal{M}$
0	$\emptyset$
1	$\{A, E\}$
2	$\{A, E\}$

D.h. $\mathcal{L}(G)$ ist unendlich.

Kapitel 4

Sprachen vom Typ 0 und 1

4.1 Kontextsensitive Grammatiken

Satz

Zu jeder erweiternden Grammatik G gibt es eine sprachäquivalente Grammatik G', deren Produktionen die Form $\alpha A \beta \to \alpha \gamma \beta$ haben, wobei

$$\begin{aligned} \alpha, \beta, \gamma &\in (\mathcal{A} \cup \mathcal{V})^*, \\ \gamma &\neq \lambda, \\ A &\in \mathcal{V}. \end{aligned}$$

Die Produktion $S \to \lambda$ darf ebenso vorkommen, aber nur, falls S auf keiner rechten Seite einer Produktion vorkommt.

Beweis

- Ersetze alle $a \in \mathcal{A}$ durch eine neue Variable A_a und erweitere Π um $A_a \to a$.
- Ersetze alle Produktionen $A_1 \dots A_m \to B_1 \dots B_n$, $B_1, \dots, B_n \in \mathcal{V}$ durch folgende Produktionen:

$$\begin{aligned} A_1 A_2 \dots A_m &\to C_1 A_2 \dots A_m \\ C_1 A_2 \dots A_m &\to C_1 C_2 A_3 \dots A_m \\ &\vdots \\ C_1 \dots C_{m+1} A_n &\to C_1 \dots C_{m-1} B_m \dots B_n \\ C_1 \dots C_{m-1} B_m \dots B_n &\to C_1 \dots C_{m-2} B_{m-1} \dots B_n \\ &\vdots \\ C_1 B_2 \dots B_n &\to B_1 \dots B_n \end{aligned}$$

  Dann ist die Grammatik in der gewünschten Form und erzeugt dieselbe Sprache.

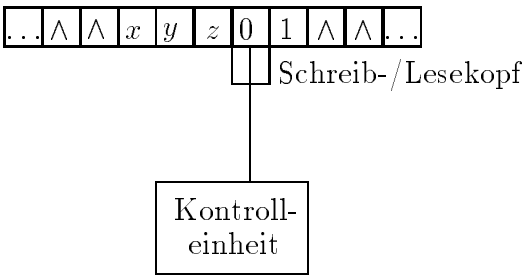


Abbildung 4.1: Turingmaschine

Beispiel (siehe Abschnitt 3.2)

$$\begin{aligned} S &\rightarrow aSBC \mid abc \\ CB &\rightarrow BC \\ aB &\rightarrow ab \\ bB &\rightarrow bb \\ C &\rightarrow c \end{aligned}$$

ist erweiternd und erzeugt die Sprache $\mathcal{L} = \{a^m b^n c^n \mid m, n \in \mathbb{N}\}$. Umformung wie im Beweis ergibt:

$$\begin{aligned} S &\rightarrow A'SBC \mid A'B'C' \\ A' &\rightarrow a \\ B' &\rightarrow b \\ C' &\rightarrow c \\ CB &\rightarrow DB \\ DB &\rightarrow DC \\ DC &\rightarrow BC \\ A'B &\rightarrow A'B' \\ B'B &\rightarrow B'B' \\ C &\rightarrow C' \end{aligned}$$

4.2 Turingmaschinen

Turingmaschinen gehen zurück auf Alan M. Turing (1912-1954), einem englischer Mathematiker und Begründer der Berechenbarkeitstheorie.
Typ-0-Sprachen werden von Turingmaschinen erkannt, Typ-1-Sprachen werden von linear beschränkten Turingmaschinen erkannt.

Definition einer Turingmaschine (siehe Abbildung 4.1)

Zu Beginn enthält das Eingabeband ein endliches Eingabewort. Die restlichen Bandfelder sind mit dem Bandzeichen $\square$ gefüllt und der Schreib-/Lesekopf steht auf dem ersten Eingabezeichen.

In jedem Schritt passiert folgendes:

- Einlesen des aktuellen Zeichens und Ersetzen
- Verschieben des Kopfs um eine oder keine Position nach links oder rechts
- Änderung des Zustands

Eine *Turingmaschine* (TM) ist ein Tupel $M = (Q, \mathcal{F}, q_0, \mathcal{A}, B, \square, \delta)$ bestehend aus

Q	Zustandsmenge
$\mathcal{F} \subset Q$	Endzustandsmenge
$q_0 \in Q$	Anfangszustand
$\mathcal{A} \subset B$	Eingabealphabet
B	Bandalphabet
$\square \in B$	Bandzeichen
$\delta : Q \times B \to Q \times B \times \{\mathtt{L}, \mathtt{N}, \mathtt{R}\}$	Übergangsfunktion (im determ. Fall)
$\delta : Q \times B \to 2^{Q \times B \times \{\mathtt{L},\mathtt{N},\mathtt{R}\}}$	Übergangsfunktion (im nichtdeterm. Fall)

Eine *Konfiguration* von M ist ein

$$\alpha q \beta \in B^* Q B^*,$$

wobei $\alpha\beta$ das momentane Bandwort enthält und q der momentane Zustand ist. Dabei steht der Schreib-/Lesekopf auf dem ersten Zeichen von β.

Achtung

$\alpha q \beta$ und $\square \alpha q \beta \square$ sind äquivalente Konfigurationen von M.

Notation

Wir schreiben $K \Rightarrow K'$, wenn M die Konfiguration K in die Konfiguration K' überführen kann und $K_1 \overset{*}{\Rightarrow} K_1$ für $K_1 \Rightarrow \ldots \Rightarrow K_n$.

Akzeptierte Sprache

$\mathcal{L}(M) := \{\omega \in \mathcal{A}^* \mid q_0 \omega \overset{*}{\Rightarrow} \alpha q \beta,\ q \in \mathcal{F}\}$

4.3 Ein Beispiel

$$\begin{array}{lcl} Q & = & \{q_0, \ldots, q_5\} \\ \mathcal{F} & := & \{q_5\} \\ \mathcal{A} & := & \{a, b, c\} \\ B & := & \{a, b, c, d, \square, \#, \not\in\} \\ \delta & & \text{so wie unten angegeben, d.h. so, daß} \\ & & \mathcal{L}(M) = \{a^n b^n c^n \mid n \in \mathbb{N}\} \end{array}$$

Abkürzung

$qa \quad pbM$ oder $qa \to pbM$ für $\delta(q,a) \ni (p,b,M)$, $M \in \{\mathtt{L},\mathtt{N},\mathtt{R}\}$

δ-Übergänge

$q_0 a$	$q_1 \# \mathtt{R}$	Wandele erstes a in # um (# = akzeptiert)
$q_1 a$	$q_1 a \mathtt{R}$	Überlies weitere a's
$q_1 b$	$q_2 ¢ \mathtt{R}$	Wandele erstes b in ein ¢ um
$q_1 ¢$	$q_1 ¢ \mathtt{R}$	Überlies ¢'s
$q_1 \#$	$q_1 \# \mathtt{R}$	Überlies #'s
$q_2 b$	$q_2 b \mathtt{R}$	Überlies weitere b's
$q_2 \#$	$q_2 \# \mathtt{R}$	Überlies weitere #'s
$q_2 c$	$q_3 \# \mathtt{L}$	Wandele erstes c um in # und ändere Laufrichtung
$q_3 \#$	$q_3 \# \mathtt{L}$	Überlies #'s
$q_3 b$	$q_3 b \mathtt{L}$	Überlies b's
$q_3 ¢$	$q_3 ¢ \mathtt{L}$	Überlies ¢'s
$q_3 a$	$q_1 \# \mathtt{R}$	Wandele das letzte a um in ein # und ändere Laufrichtung
$q_3 \square$	$q_4 \square \mathtt{R}$	Falls kein a mehr gefunden, ändere Laufrichtung
$q_4 \square$	$q_4 \# \mathtt{R}$	Prüfe, ob kein b
$q_4 ¢$	$q_4 ¢ \mathtt{R}$	und kein c mehr da
$q_4 \square$	$q_5 \square \mathtt{N}$	Falls der Fall, dann akzeptiere

4.4 Linear beschränkte Turing-Maschine

Eine linear beschränkte Turingmaschine (LBA) ist eine TM, die Bandzeichen unverändert läßt und nur auf dem Teil des Bandes operiert, auf dem die Eingabe mit den beiden sie einschließenden Bandzeichen steht.

Satz

Jede kontextsensitive Sprache wird durch einen LBA akzeptiert.

Beweis

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine erweiternde Grammatik. Konstruiere einen LBA M wie folgt:
M wählt nichtdeterministisch eine Produktion $\alpha \to \beta \in \Pi$ und sucht auf dem Band ein β und ersetzt β durch α.
Falls $|\alpha| < |\beta|$, dann schiebt M die Eingabe zusammen. Steht nur noch S auf dem Band oder ist das Band leer und $\lambda \in \mathcal{L}$, geht die Maschine M in einen akzeptierenden Zustand über.

Folgerung

Jede Typ-0-Sprache wird durch eine TM akzeptiert.

Umkehr des Satzes

Ein LBA akzeptiert eine kontextsensitive Sprache.

Beweis

Sei M ein LBA. Dann konstruieren wir eine kontextsensitive Grammatik $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$, die $\mathcal{L}(M)$ akzeptiert wie folgt.

$$\begin{aligned}\mathcal{V} \;:=\; & (B^2) \cup (QB^2) \cup (Q\square\,\square\,B^2) \cup (\square\,\square\,QB^2) \cup \\ & (QB^2\square\,\square\,) \cup (B^2Q\square\,\square\,) \cup \{S, A\}\end{aligned}$$

- Die Produktionen

$$\begin{aligned} S &\to A(aa\square\,\square\,), a \in \mathcal{A} \\ A &\to (q_0aa) \mid A(aa), a \in \mathcal{A}\end{aligned}$$

  erzeugen die Satzformen

$$(\square\,\square\,q_0a_1a_1)(a_2a_2)\ldots(a_na_n)(aa\square\,\square\,),\ n \in \mathbb{N}.$$

  Diese stellen alle möglichen Anfangskonfigurationen dar. Die Verdopplungen dienen dazu, sich die Eingabe zu merken.

- Die δ-Übergänge $q \times py\mathtt{L}$ werden simuliert durch die Produktionen

$$\begin{aligned}(\square\,\square\,qxa) &\to (p\square\,\square\,qa) \\ (\square\,\square\,zb)(qxa) &\to (\square\,\square\,pzb)(ya) \\ (zb)(qxa) &\to (pzb)(ya) \\ (zb)(qxa\square\,\square\,) &\to (pzb)(ya\square\,\square\,) \\ (xaq\square\,\square\,) &\to (pya\square\,\square\,)\end{aligned}$$

- Die δ-Übergänge $q \times py\mathtt{R}$ werden durch analoge Produktionen simuliert.

- Die δ-Übergänge $q \times py\mathtt{N}$ werden durch die Produktionen

$$\begin{aligned}(q\square\,\square\,zb) &\to (p\square\,\square\,zb) \\ (\square\,\square\,qxa) &\to (\square\,\square\,pya) \\ (qxa) &\to (pya) \\ (qxa\square\,\square\,) &\to (pya\square\,\square\,) \\ (zbq\square\,\square\,) &\to (zbp\square\,\square\,)\end{aligned}$$

  simuliert.

- Die Produktionen

$$\begin{aligned}(q\square\,\square\,xa) &\to a,\ q \in \mathcal{F} \\ (\square\,\square\,qxa) &\to a \\ (qxa) &\to a \\ (qxa\square\,\square\,) & \\ (xaq\square\,\square\,) &\to a\end{aligned}$$

  für alle $q \in \mathcal{F}$, $x \in B$, $a \in \mathcal{A}$ erzeugen die von M akzeptierten Worte.

- Die Produktionen $S \to a$, $a \in A_\lambda \cap \mathcal{L}(M)$, erzeugen die Worte $\omega \in \mathcal{L}(M)$ mit $|\omega| \leq 1$.

Alle oben aufgeführten Produkte ergeben eine kontextsensitive Grammatik G, die $\mathcal{L}(M)$ erzeugt.

Folgerung

Ist M eine TM, dann ist $\mathcal{L}(M)$ vom Typ 0.

Beweis

ähnlich wie oben, aber mit den zusätzlichen Produktionen $S \to (\square\,\square)S(\square\,\square)$, um beliebig große Anfangskonfigurationen darstellen zu können (dem Bandbedarf der TM entsprechend) und den Produktionen

$$\begin{aligned}(q\square\,\square) &\to \lambda, q \in \mathcal{F}\\ (\square\,\square) &\to \lambda\end{aligned}$$

aber ohne die Produktionen, in denen $\square\,\square$ vorkommen.

Bemerkung

Aus dem obenstehenden Beweis folgt auch der Satz in Abschnitt 4.1.

4.5 Deterministische Turingmaschinen

Satz

Die deterministischen Turingmaschinen erkennen dieselbe Sprachfamilie wie die nichtdeterministischen Turingmaschinen.

Beweis

Sei M eine nichtdeterministische Turingmaschine. Eine äquivalente Turingmaschine M' kann so konstruiert werden:
M' erzeugt zu einer Startkonfiguration von M alle möglichen Folgekonfigurationen und schreibt sie nebeneinander auf das Band.
Mit jeder Folgekonfiguration verfährt M' genauso. Taucht ein Endzustand in einer Folgekonfiguration auf, dann geht M' in einen Endzustand über.

Bemerkung

Es ist noch offen, ob deterministische LBAs und nichtdeterministische LBAs äquivalent sind.

4.6 Abschlusseigenschaften

Die folgende Tabelle zeigt, unter welchen Operationen die Sprachen vom Typ 0, 1, 2, 3 abgeschlossen sind:

	1.4 Produkt	1.4 Stern	1.4 $\cup$	$\cap$	Komplement
Typ 3	ja 1.4	ja 1.4	ja 1.4	ja 2.7	ja 2.7
Typ 2	ja 1.4	ja 1.4	ja 1.4	nein 3.8	nein 3.8
Typ 1	ja 1.4	ja 1.4	ja 1.4	ja[1]	ja[2]
Typ 0	ja 1.4	ja 1.4	ja 1.4	ja[1]	nein[3]

1. Beweis durch Produktautomat
2. ohne Beweis
3. später (Berechenbarkeitstheorie)

4.7 Das Halteproblem

Im folgenden werden nur deterministische Turingmaschinen betrachtet. Ohne Einschränkung der Allgemeinheit nehmen wir an, daß q_0 kein Endzustand ist. Dann kann eine Turingmaschine durch eine Folge von Quintupeln $pxqyM$ dargestellt werden, wobei

$$\begin{aligned} p, q &\in \overbrace{\{q_0, q_1, \dots\}}^{\text{Zustände}} \uplus \overbrace{\{f_0, f_1, \dots\}}^{\text{Endzustände}}, \\ x, y &\in \{a_0, a_1, \dots\} = \text{Bandalphabet mit } a_0 = \square, \\ M &\in \{\mathtt{L}, \mathtt{R}, \mathtt{N}\}. \end{aligned}$$

Beispiel

$$q_7 a_6 q_{19} a_1 \mathtt{N}$$
$$q_0 a_1 q_7 a_6 \mathtt{L}$$

Gödel-Nummern

Also ist jede Turingmaschine darstellbar durch eine Zeichenkette über $\{q, a, 0, 1, \dots, 9, \mathtt{L}, \mathtt{R}, \mathtt{N}\}$. Wird diese als natürlich Zahl in Hexadezimaldarstellung aufgefaßt, bekommt man eine *Gödel-Nummer* der Turingmaschine.
Im allgemeinen hat eine Turingmaschine mehrere Gödel-Nummern, weil die Quintupel permutiert werden können. M_i bezeichnet die Turingmaschine mit Gödel-Nummer i.
Kodiert $i \in \mathbb{N}$ keine Turingmaschine, definieren wir M_i als $q_0 a_0 q_0 a_0 \mathtt{N}$.

Die Funktion $\varphi_i : \mathbb{N}_0 \to \mathbb{N}_0$ ist wie folgt definiert:

Man gibt $\overbrace{a_1 \dots a_1}^{n}$ für M_i ein
Hält M_i, dann ist $\varphi_i(n)$ die Zahl der a_1 auf dem Band.
Hält M_i nicht, dann ist $\varphi_i(n)$ undefiniert, kurz $\varphi_i(n) = \perp$.

Im allgemeinen sind die φ_i *partielle,* d.h. nicht auf ganz $\mathbb{N}_0$ definierte Funktionen.

Beispiel

φ_0 ist überall undefiniert.

Definition

Die Funktionen φ_i heißen *berechenbar*, alle anderen Funktionen $\mathbb{N}_0 \to \mathbb{N}_0$ heißen *nicht berechenbar.*

Satz

Die Funktion

$$\varphi(i) = \begin{cases} \perp & \text{falls } \varphi_i(i) \text{ definiert} \\ 1 & \text{sonst} \end{cases}$$

ist nicht berechenbar.

Beweis

Falls $\varphi = \varphi_j$, dann wäre

$$\varphi_j(j) = \begin{cases} \perp & \text{falls } \varphi_j(j) \text{ hält} \\ 0 & \text{falls } \varphi_j(j) = \perp \end{cases}$$

Dies ist ein Widerspruch zur Annahme.

Folgerung

Die Halte-Funktion

$$\texttt{halt}(n) = \begin{cases} 1 & \text{falls } \varphi_n(n) \neq \perp \\ 0 & \text{sonst} \end{cases}$$

ist nicht berechenbar.

Beweis

Wäre `halt` berechenbar, dann offenbar auch φ.

4.8 Das Wortproblem

These (Church)

Jeder intuitive Algorithmus ist durch eine Turingmaschine berechenbar.

Satz

Das Wortproblem für Typ-0-Sprachen ist unentscheidbar.

Beweis

Wäre das Wortproblem lösbar, dann wäre auch `halt` berechenbar. Nach der These von Church führt dies zu einem Widerspruch.

Das Halteproblem ist ein spezielles Wortproblem. Das Halteproblem kann auf das Wortproblem *reduziert* werden, kurz:

$$\text{Halteproblem} \quad \leq \quad \text{Wortproblem}$$

Satz

Das Wortproblem ist für kontextsensitive Sprachen entscheidbar.

Beweis

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine erweiternde Grammatik und $\omega \in \mathcal{A}^*$. Alle Worte in $\mathcal{L}(G)$, die nicht länger als ω sind, werden von folgendem Algorithmus gefunden:

$M := \emptyset, \quad M' := \{S\}$
`while` $M \neq M'$ `do`
    $M := M'$
    $M' := M \cup \{\beta \mid \alpha \Rightarrow \beta, \alpha \in M, |\beta| \leq |\omega|\}$

Es gilt also $\omega \in \mathcal{L}(G) \Leftrightarrow \omega \in M$.

Folgerung

Die Typ-0-Sprachen bilden eine *echte* Obermenge der Typ-1-Sprachen.

4.9 Das Postsche Korrespondenzproblem

Ein *Post-System* über einem endlichen Alphabet $\mathcal{A}$ ist eine Folge

$$\begin{pmatrix} a_1 \\ b_1 \end{pmatrix}, \ldots, \begin{pmatrix} a_n \\ b_n \end{pmatrix}$$

in $\mathcal{A}^* \times \mathcal{A}^*$. Das *Postsche Korrespondenzproblem* (PCP) besteht darin, $i_1, \ldots, i_k \in \mathbb{N}$ zu finden, so daß

$$a_{i_1} \ldots a_{i_k} = b_{i_1} \ldots b_{i_k}.$$

Die Indexfolge $i_1, \ldots, i_k$ heißt *Lösung* des PCPs.

Beispiel 1

$\begin{pmatrix} 11 \\ 10 \end{pmatrix}, \begin{pmatrix} \lambda \\ 110 \end{pmatrix}, \begin{pmatrix} 111 \\ \lambda \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ hat die Lösung 2,1,4,4,3:

$$\begin{aligned} a_2 a_1 a_4 a_4 a_3 &= 1100111 \\ &= b_2 b_1 b_4 b_4 b_3 \end{aligned}$$

Beispiel 2

Die kürzeste Lösung für $\begin{pmatrix} 001 \\ 0 \end{pmatrix}, \begin{pmatrix} 01 \\ 011 \end{pmatrix}, \begin{pmatrix} 01 \\ 101 \end{pmatrix}, \begin{pmatrix} 10 \\ 001 \end{pmatrix}$ hat 66 Indizes, siehe Schöning S. 132.

Satz

Für jede Typ-0-Grammatik $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ und für alle $\omega \in \mathcal{A}^*$ existiert ein konstruktiv bestimmbares Post-System K, das genau dann lösbar ist, wenn $\omega \in \mathcal{L}(G)$.

Beweis

Konstruiere ein Postsystem K über $\mathcal{X} := \mathcal{V} \cup \mathcal{V}' \cup \mathcal{A} \cup \mathcal{A}' \cup \{*, [,]\}$, wobei $'$ Kopien kennzeichnet.
Das Post-System K bestehe aus

$$\left(\begin{array}{c} [S* \; S* \\ [\end{array} \right), \left(\begin{array}{c}] \\ *\omega] \end{array} \right), \left(\begin{array}{c}] \\ *\omega'] \end{array} \right), \left(\begin{array}{c} \beta' \\ \alpha \end{array} \right), \left(\begin{array}{c} \beta \\ \alpha' \end{array} \right), \left(\begin{array}{c} * \\ * \end{array} \right),$$

wobei $\alpha \to \beta \in \Pi$. Dann gilt: $i_1, \ldots, i_k$ ist eine Lösung von K
$\Leftrightarrow \left(\begin{array}{ccc} a_{i_1} & \cdots & a_{i_k} \\ b_{i_1} & \cdots & b_{i_k} \end{array} \right) =$

$$\left(\begin{array}{c|c|c|c|c|c|c|} [S* & \omega_1' & * & \omega_2 & * & \omega_3 & \ldots \\ [& S & * & \omega_1' & * & \omega_2 & \end{array} \right. \left\{ \begin{array}{l} \left(\begin{array}{c|c} \omega & \\ \omega_1' & *w \end{array} \right) \text{ falls } n \text{ ungerade} \\ \left(\begin{array}{c|c} \omega' & \\ \omega_n & *\omega' \end{array} \right) \text{ falls } n \text{ gerade} \end{array} \right.$$

$\Leftrightarrow S \Rightarrow \omega_1 \Rightarrow \omega_2 \Rightarrow \ldots \Rightarrow \omega_k \Rightarrow \omega$
$\Leftrightarrow \omega \in \mathcal{L}(G)$.

4.10 (Un)entscheidbare Probleme

1. Satz

Das Schnittproblem ist für kontextfreie Sprachen nicht entscheidbar (und deshalb auch nicht für kontextsensitive und Typ-0-Sprachen).

Beweis (durch Reduktion des PCP)

Sei $\left(\begin{array}{c} a_1 \\ b_1 \end{array} \right), \ldots, \left(\begin{array}{c} a_n \\ b_n \end{array} \right)$ ein Post-System. Die Produktionen

$$A \to a_1 A_1 \mid \ldots \mid a_n A_n \mid a_1 1 \mid \ldots \mid a_1 n$$

erzeugen dann die kontextfreie Sprache $\mathcal{L}_a := \{a_{i_1} \ldots a_{i_k} i_k \ldots i_1 \mid i_1, \ldots, i_k \in \{1, \ldots, n\}\}$, wobei $1, \ldots, n$ als n verschiedene Zeichen zu verstehen sind.
Die Produktionen

$$B \to b_1 B 1 \mid \ldots \mid b_n B_n \mid b_1 1 \mid \ldots \mid b_n n$$

erzeugen die kontextfreie Sprache $\mathcal{L}_b := \{b_{i_1} \ldots b_{i_k} i_k \ldots i_1 \mid i_1, \ldots, i_k \in \{1, \ldots, n\}\}$.
Offenbar repräsentiert $\mathcal{L}_a \cap \mathcal{L}_b$ alle Lösungen des Post-Systems.

2. Folgerung

Das Schnittproblem ist auch für kontextfreie Sprachen mit kontextfreiem Komplement nicht entscheidbar.

Beweis

$\mathcal{L}_a$ und $\mathcal{L}_b$ haben kontextfreie Komplemente (PDA als Aufgabe).

3. Folgerung

Das Schnittproblem ist nicht entscheidbar für:

1. kontextfreie Sprachen
2. kontextfreie Sprachen mit kontextfreiem Komplement
3. Typ-0- und Typ-1-Sprachen

4. Satz

Das Leerheitsproblem ist für kontextsensitive Sprachen nicht entscheidbar und deshalb auch nicht für Typ-0-Sprachen.

Beweis (Reduktion des Schnittproblems)

Kontextsensitive und Typ-0-Sprachen sind bezüglich des Schnitts abgeschlossen (siehe Abschnitt 4.6). Wäre das Leerheitsproblem entscheidbar, dann auch das Schnittproblem.

5. Satz

Das Gleichheitsproblem ist für kontextfreie Sprachen nicht entscheidbar.

Beweis (durch Reduktion des Schnittproblems)

Seien $\mathcal{L}_1$ und $\mathcal{L}_2$ kontextfreie Sprachen mit kontextfreiem Komplement. Weil

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \emptyset \Leftrightarrow \mathcal{L}_1 \cup \overline{\mathcal{L}_2} = \overline{\mathcal{L}_2},$$

wäre das Schnittproblem entscheidbar, wenn die Gleichheit der kontextfreien Sprachen $\mathcal{L}_1 \cup \overline{\mathcal{L}_2}$ und $\overline{\mathcal{L}_2}$ entscheidbar wäre. Das Schnittproblem ist aber nicht entscheidbar, womit die Behauptung folgt.

6. Zusammenfassung

entscheidbar	Wortproblem	Schnitt	Leerheit	Gleichheit
Typ 3	ja 2.8	ja 2.8	ja 2.8	ja 2.8
Typ 2	ja 3.9	nein 1.	ja 3.9	nein 5.
Typ 1	ja 4.8	nein 3.	nein 4.	nein 5.
Typ 0	nein 4.8	nein 3.	nein 4.	nein 5.

7. Satz

Es kann nicht entschieden werden, ob kontextfreie Grammatiken eindeutig sind.

Beweis (durch Reduktion des PCP)

Betrachte das Post-System aus 1. Sei G die kontextfreie Grammatik mit den Produktionen

$$\begin{array}{ll} S & A \mid B \\ A & a, A1 \mid \ldots \mid a_n A_n \mid a_n n \mid \ldots \mid a_1, 1 \\ B & b, B1 \mid \ldots \mid b_n A_n \mid b_n n \mid \ldots \mid b_{1,} 1 \end{array}$$

Offenbar stellen die Lösungen des PCP die Worte mit 2 oder mehr Ableitungen in $\mathcal{L}(G)$ dar.
Da das Postsche Korrespondenzproblem nicht entscheidbar ist, kann auch die Eindeutigkeit nicht entscheidbar sein.

Kapitel 5

Fixpunkttheorie

5.1 Halbordnungen

Definition

$\leq$ ist eine Halbordnung auf einer Menge $\mathcal{M}$, falls $\forall x, y, z \in \mathcal{M}$:

- $x \leq x$ (Reflexivität),
- $x \leq y \wedge y \leq z \Rightarrow x \leq z$ (Transitivität),
- $x \leq y \wedge y \leq x \Rightarrow y = x$ (Antisymmetrie).

Bemerkung 1

$\leq$ ist eine *totale Ordnung*, falls zusätzlich gilt: $x \leq y \vee y \leq x$.

Bemerkung 2

Ist $\leq$ eine Halbordnung, dann auch die Relation $\geq$, wobei $x \geq y :\Leftrightarrow y \leq x$.

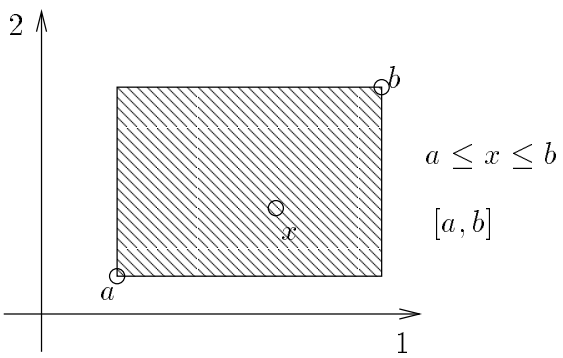


Abbildung 5.1: Halbordnung

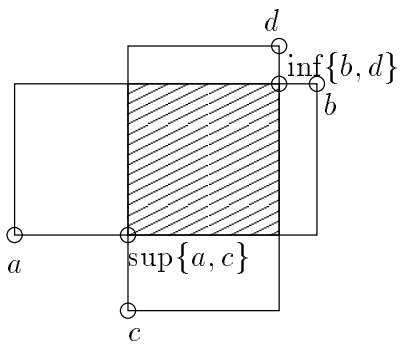


Abbildung 5.2: $[a,b] \wedge [c,d] = [\sup\{a,c\}, \inf\{b,d\}]$

Beispiel 1

$\mathcal{M} := \mathbb{R}^2$, $\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \leq \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} :\Leftrightarrow a_1 \leq b_1 \wedge a_2 \leq b_2$. Dies ist eine Halbordnung (siehe Abbildung 5.1), aber keine totale Ordnung, denn

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix} \not\leq \begin{bmatrix} 1 \\ 0 \end{bmatrix} \not\leq \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Beispiel 2

$\mathcal{M} = 2^{\mathcal{A}}$; die Inklusion $\subseteq$ ist eine Halbordnung.

Beispiel 3

$\mathcal{M} := 2^{\mathcal{A}} \times \cdots \times 2^{\mathcal{A}} - (2^{\mathcal{A}})^n$ die Menge aller k-Tupel von Teilmengen von $\mathcal{A}$. Die Halbordnung $\leq$ sei definiert durch:

$$(\mathcal{U}_1, \ldots, \mathcal{U}_n) \leq (\mathcal{V}_1, \ldots, \mathcal{V}_n) \Leftrightarrow \forall i = 1, \ldots, n : \mathcal{U}_i \subset \mathcal{V}_i$$

Definition

$\leq$ sei eine Halbordnung auf $\mathcal{M}$.

- $m \in \mathcal{M}$ heißt das *kleinste Element* von $\mathcal{M}$, falls $\forall x \in \mathcal{M} : m \leq x$. Es muß nicht existieren. Wenn es existiert, bezeichnen wir es mit $\min \mathcal{M}$.
- s heißt *obere Schranke* von $\mathcal{U} \subset \mathcal{M}$, falls $\forall x \in \mathcal{U} : x \leq s$.
- $\sup \mathcal{U}$ ist die *kleinste obere Schranke* von $\mathcal{U}$.
- Eine *Kette* in $\mathcal{M}$ ist eine Folge $a_1, a_2, \ldots$ mit $a_1 \leq a_2 \leq \ldots$

In Beispiel 1

gibt es kein kleinstes Element. $\mathbb{R}^2$ hat kein Supremum. In Abbildung 5.3 und 5.2 sind Beispiele für Suprema zu sehen.

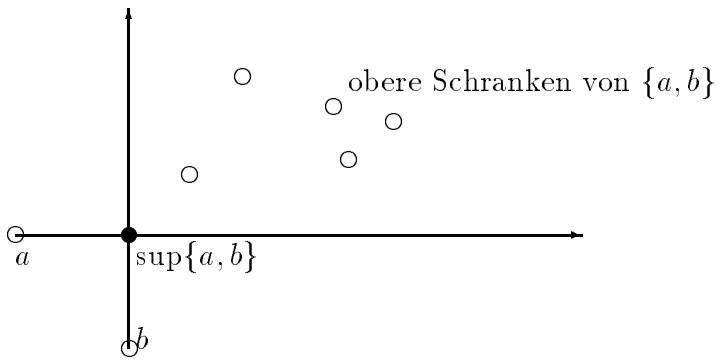


Abbildung 5.3: Beispiel 1

In Beispiel 2

ist $\emptyset = \min \mathcal{M}$ und $\sup \mathcal{M} = \mathcal{A}$.

In Beispiel 3

ist $\min \mathcal{M} = (\emptyset, \ldots, \emptyset)$ und $\sup \mathcal{M} = (\mathcal{A}, \ldots, \mathcal{A})$.

5.2 Ein Fixpunkttheorem

Definition

$(\mathcal{M}, \leq)$ ist eine *vollständig halbgeordnete Menge*, falls

- $\min \mathcal{M}$ existiert.
- $\forall$Ketten k in $\mathcal{M}$ existiert $\sup k$ in $\mathcal{M}$.

Beispiel

In Abschnitt 5.1, Beispiel 2 und 3 ist die Halbordnung vollständig.

Definition

$(\mathcal{A}, \leq_{\mathcal{A}})$ und $(\mathcal{B}, \leq_{\mathcal{B}})$ seien vollständig halbgeordnete Mengen. $f : \mathcal{A} \to \mathcal{B}$ heißt *stetig*, falls $\forall$Ketten $(x_1, x_2, \ldots) \subset \mathcal{A}$:

$$\sup_{k=1}^{\infty} f(x_k) = f(\sup_{k=1}^{\infty} x_k)$$

Lemma

Ist f stetig, dann ist f auch monoton.

Beweis

Sei $a \leq_{\mathcal{A}} b$ und $(x_1, x_2, \ldots) = (a, b, b, \ldots)$

$$\begin{aligned} \Rightarrow f(a) \quad &\leq_{\mathcal{B}} \quad \sup_{k=1}^{\infty} f(x_k) \\ &\overset{\text{stetig}}{=} \quad f(\sup_{k=1}^{\infty} x_k) = f(b) \end{aligned}$$

Bemerkung

Nach dieser Definition sind reelle Funktionen $f : [a, b] \to [c, d]$ genau dann stetig, wenn f monoton und linksseitig stetig ist im Sinne der Analysis.

Definition

Sei $(\mathcal{A}, \leq)$ eine vollständig halbgeordnete Menge und $f : \mathcal{A} \to \mathcal{A}$ eine Abbildung. Ein Fixpunkt von f ist ein $a \in \mathcal{A}$ mit $f(a) = a$.

Satz (Kleene)

$f : \mathcal{A} \to \mathcal{A}$ sei eine stetige Abbildung auf einer vollständig halbgeordneten Menge $(\mathcal{A}, \leq)$. Für $m := \min \mathcal{A}$ gilt:

$$\sup_{k=0}^{\infty} f^k(m) \text{ ist der kleinste Fixpunkt von } f$$

(das heißt auch, daß f einen Fixpunkt hat).

Beweis

- $\sup f^k(m)$ existiert:
  Zeige, daß $(f^k(m))_{k=0}^{\infty}$ eine Kette ist.

$$\begin{aligned} & m = \min \mathcal{A} \leq f(m) \\ \overset{\text{Monotonie}}{\Rightarrow} \quad & f(m) \leq f^2(m) \\ \Rightarrow \cdots \Rightarrow \quad & f^k(m) \leq f^{k+1}(m). \end{aligned}$$

- $\sup f^k(m)$ ist ein Fixpunkt:

$$\begin{aligned} f(\sup_{k=0}^{\infty} f^k(m)) \quad &\overset{\text{stetig}}{=} \quad \sup_{k=0}^{\infty} (f^{k+1}(m)) \\ &= \quad \sup_{k=0}^{\infty} f^k(m), \text{ da } m = \min \mathcal{A} \leq \sup f^{k+1}(m). \end{aligned}$$

- $\sup f^k(m)$ ist der kleinste Fixpunkt:
  c sei ein Fixpunkt von f:

$$\begin{aligned} m \quad &= \quad \min \mathcal{A} \leq c \\ &\overset{\text{Monotonie}}{\Rightarrow} \quad f(m) \leq f(c) = c \\ &\Rightarrow \ldots \Rightarrow \quad \forall k : f^k(m) \leq c \\ &\Rightarrow \quad \sup f^k(m) \leq c \end{aligned}$$

  Anschaulich siehe Abbildung 5.4, 5.5.

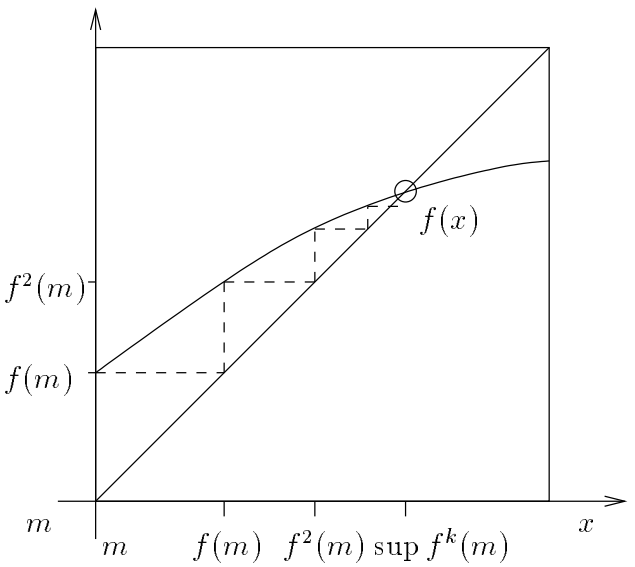


Abbildung 5.4: monoton steigende Funktion

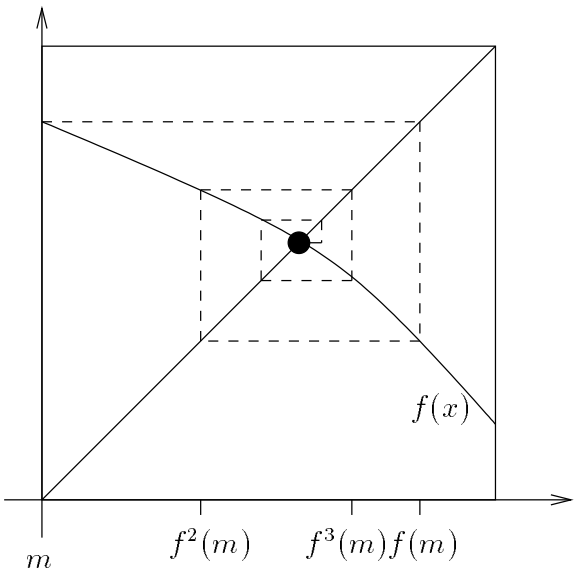


Abbildung 5.5: **Bemerkung** auch für stetig monoton fallende Funktionen

5.3 Kontextfreie Sprachen als Fixpunkte

$G = (\mathcal{V}, \mathcal{A}, A_1, \Pi)$ sei eine kontextfreie Grammatik mit $\mathcal{V} := \{A_1, \ldots, A_n\}$ und den Produktionen

$$A_1 \to \alpha_{i1} \mid \ldots \mid \alpha_{ik_i}, \quad i = 1, \ldots, n, \alpha_{ij} \in (\mathcal{A} \cup \mathcal{V})^*.$$

Zu den Produktionen $A_i \to \ldots$ definieren wir die Abbildung

$$\begin{aligned} g_i : 2^{\mathcal{A}^*} \times \cdots \times 2^{\mathcal{A}^*} &\to 2^{\mathcal{A}^*} \\ (\mathcal{M}_1, \ldots, \mathcal{M}_n) &\mapsto \overline{\alpha}_{i1} \cup \cdots \cup \overline{\alpha}_{ik_i}, \end{aligned}$$

wobei $\overline{\alpha}_{ij}$ aus a_{ij} entsteht, indem

$$\begin{aligned} &A_1 \text{ durch } \mathcal{M}_1 \text{ ersetzt wird,} \\ &\vdots \\ &A_n \text{ durch } \mathcal{M}_n \text{ ersetzt wird.} \end{aligned}$$

Beispiel

$$\begin{aligned} A_i &\to aA_2c \mid bA_1c \mid cA_1a \\ \mathcal{M}_1 &:= \{a, c\}, \quad \mathcal{M}_2 = \{b\} \\ g_i(\mathcal{M}_1, \mathcal{M}_2) &= a\mathcal{M}_2c \cup b\mathcal{M}_1c \cup c\mathcal{M}_1a \\ &= \{abc, bac, bcc, caa, cca\} \end{aligned}$$

Definition

Die *Funktion der Grammatik* G ist die Abbildung

$$\begin{aligned} g = (g_1, \ldots, g_n) \quad : \quad & 2^{\mathcal{A}^*} \times \cdots \times 2^{\mathcal{A}^*} \to 2^{\mathcal{A}^*} \times \cdots \times 2^{\mathcal{A}^*} \\ & (\mathcal{M}_1, \ldots, \mathcal{M}_2) \mapsto (g_1(\mathcal{M}_1, \ldots, \mathcal{M}_n), \ldots, \\ & g_n(\mathcal{M}_1, \ldots, \mathcal{M}_n)). \end{aligned}$$

Beispiel

$S \to ab \mid ScS \mid ab$
Dann ist $g(\mathcal{M}) = a\mathcal{M}b \cup \mathcal{M}c\mathcal{M} \cup \{ab\}$.
Insbesondere gilt für $\mathcal{M} = \mathcal{L}(G)$

$$a\mathcal{M}b = \mathcal{M}c\mathcal{M} = \mathcal{M} \text{ und } ab \in \mathcal{M}$$

Also $g(\mathcal{M}) = \mathcal{M}$.

Lemma

Die Funktion g einer kontextfreien Grammatik ist stetig.

Beweis

Sei $\mathcal{M}_k = (\mathcal{M}_{k1}, \ldots, \mathcal{M}_{kn})$. Es ist zu zeigen, daß

$$g(\sup \mathcal{M}_k) = (\ldots g_i(\sup \mathcal{M}_k), \ldots) = \sup g(\mathcal{M}_k) = (\ldots, \sup g_i(\mathcal{M}_k), \ldots),$$

zum Beispiel für die erste Koordinate

$$\begin{aligned}
\omega \in g_1(\sup(\mathcal{M}_k)) &= g_1(\sup(\mathcal{M}_{k1}), \ldots, \sup(\mathcal{M}_{kn})) \\
&= \alpha_{11} \cup \cdots \cup \alpha_{1k_1}, \text{ wobei die } A_j \text{ durch } \sup \mathcal{M}_{kj} \text{ ersetzt sind} \\
&\Leftrightarrow \exists i : \omega \in \alpha_{1i}, \text{ wobei die } A_j \text{ durch } \sup \mathcal{M}_{kj} \text{ ersetzt sind} \\
&\Leftrightarrow \exists i \exists k : \omega \in \alpha_{ij}, \text{ wobei die } A_j \text{ ersetzt sind durch } \mathcal{M}_{kj} \\
&\Leftrightarrow \exists k : \omega \in \alpha_{11} \cup \cdots \cup \alpha_{1k_1}, \text{ wobei } A_j \text{ durch } \mathcal{M}_{kj} \text{ ersetzt wird} \\
&= g_1(\mathcal{M}_{k1}, \ldots, \mathcal{M}_{kn}) \\
&\Leftrightarrow \omega \in \sup g_1(\mathcal{M}_{k1}, \ldots, \mathcal{M}_{kn}) \\
&= \sup g_1(\mathcal{M}_k)
\end{aligned}$$

Satz

Sei $\mathcal{L}_i := \{\omega \in \mathcal{A}^* \mid A_i \overset{*}{\Rightarrow} \omega\}$. Dann ist $\mathcal{L}_1 = \mathcal{L}(G)$ und $(\mathcal{L}_1, \ldots, \mathcal{L}_n)$ der kleinste Fixpunkt von g.

Beweis

Sei $\mathcal{L}_{ik} := \{\omega \in \mathcal{A}^* \mid \omega \text{ ist in } \leq k \text{ Schritten aus } A_i \text{ ableitbar}\}$. Dann ist

$$\begin{aligned}
g(\emptyset, \ldots, \emptyset) &= (\mathcal{L}_{11}, \ldots, \mathcal{L}_{n1}), \\
g^k(\emptyset, \ldots, \emptyset) &= (\mathcal{L}_{1k}, \ldots, \mathcal{L}_{nk})
\end{aligned}$$

und

$$\sup g^k(\emptyset, \ldots, \emptyset) = (\mathcal{L}_1, \ldots, \mathcal{L}_n).$$

D.h. die Behauptung folgt aus dem Satz von Kleene (Abschnitt 5.2).

5.4 Gleichungssysteme

Beispiel

G sei gegeben durch

$$\begin{aligned}
A_1 &\rightarrow aA_2 \mid A_3 \\
A_2 &\rightarrow bA_1c \mid a \\
A_3 &\rightarrow aA_2b \mid ba
\end{aligned}$$

Dann ist in Vektorschreibweise

$$g(\mathcal{M}) = \begin{pmatrix} \mathcal{M}_1 \\ \mathcal{M}_2 \\ \mathcal{M}_3 \end{pmatrix} = \begin{pmatrix} a\mathcal{M}_2 + \mathcal{M}_3 \\ b\mathcal{M}_1c + a \\ a\mathcal{M}_2b + ba \end{pmatrix},$$

wobei + für $\cup$ und a für $\{a\}$ etc. steht.
Der Fixpunkt $(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3)$ von g ist die kleinste Lösung des folgenden Gleichungsproblems:

$$\begin{aligned} \mathcal{L}_1 &= a\mathcal{L}_2 + \mathcal{L}_3 \\ \mathcal{L}_2 &= b\mathcal{L}_1 c + a \\ \mathcal{L}_3 &= a\mathcal{L}_2 b + ba \end{aligned}$$

Wie kann das lineare Gleichungssystem gelöst werden?

Satz

Die kleinste Lösung für $\mathcal{L} = \mathcal{A}\mathcal{L} + B$ ist $\mathcal{L} = \mathcal{A}^*\mathcal{B}$.

Beweis

Die Funktion $g(\mathcal{L}) = \mathcal{A}\mathcal{L} + B$ der Gleichung ist stetig, siehe Lemma in Abschnitt 5.3. Nach Abschnitt 5.2 ist also $\sup g^k(\emptyset)$ die kleinste Lösung, d.h.

$$\mathcal{L} = \sup g^k(\emptyset) = \emptyset \cup B \cup \mathcal{A}B \cup \mathcal{A}^2 B \cup \cdots = \mathcal{A}^* B$$

Beispiel

Gegeben sei

$$\begin{aligned} \mathcal{A} &= a\mathcal{A} + bB + c \\ B &= d\mathcal{A} + eB + f, \end{aligned}$$

wobei $\mathcal{A}, B$ unbekannte Sprachen über $\{a, b, \ldots, f\}$ seien.
Anwendung des Satzes:

$$\mathcal{A} = a^*(bB + c).$$

Einsetzen in Gleichung II:

$$B = da^*(bB + c) + eB + f = (da^*b + e)B + da^*c + f.$$

Anwendungs des Satzes:

$$B = (da^*b + e)^*(da^*c + f).$$

Einsetzen:

$$\mathcal{A} = a^*(b \cdot (da^*b + e)^*(da^*c + f) + a^*c).$$

Kapitel 6

Syntaxanalyse

Syntaxanalyse: $\omega \in \mathcal{L}(G)$?

Ziel: kontextfreie Grammatiken, die sich für Syntaxanalyse eignen.

6.1 Einführende Beispiele

Man ist an Grammatiken interessiert, bei denen man bei einer top-down-Analyse möglichst wenig Ableitungen untersuchen muß.

Beispiel 1

$$\begin{aligned} S &\rightarrow aAb \mid bBc \mid c \\ A &\rightarrow aA \mid bBB \\ B &\rightarrow bSS \mid c \end{aligned}$$

Analysiere $\omega = aabbcccb$

$$\begin{aligned} S &\Rightarrow aAb \Rightarrow aaAb \Rightarrow aabBBb \\ &\Rightarrow aabbSSBb \\ &\Rightarrow aabbcSBb \\ &\Rightarrow aabbcccb \end{aligned}$$

Die Linksableitung ist eindeutig bestimmbar, weil alle möglichen Ersetzungen einer Variable mit unterschiedlichen Terminalzeichen beginnen.

Beispiel 2

$$\begin{aligned} S &\rightarrow aA \mid SBA \\ A &\rightarrow cB \mid dA \\ B &\rightarrow SA \mid AS \end{aligned}$$

Hier ist unklar, wie S zu ersetzen wäre. Man schaut eine Ableitung weiter:

$$B \rightarrow SA \mid AS \Rightarrow aAA \mid bBAA \mid cBS \mid dAS$$

Beispiel 3

$$\begin{aligned} S &\rightarrow abc \mid cAB \\ A &\rightarrow BS \mid aB \\ B &\rightarrow abA \mid c \end{aligned}$$

Probleme wie oben: also eine Ableitung weiter schauen.

$$A \rightarrow BS \mid aB \Rightarrow abAS \mid cS \mid aB$$

Problem noch nicht gelöst. Leite weiter ab, bis die ersten k Terminalzeichen verschieden.

$$A \stackrel{*}{\Rightarrow} abAS \mid cS \mid aabA \mid ac$$

6.2 SLL(k)-Grammatiken

Sei $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine kontextfreie Grammatik und $\omega = (\mathcal{A} \cup \mathcal{V})^*$ eine Satzform.

Definitionen

- $\text{Kopf}_k(\omega) := \left\{ \begin{array}{l} \omega \\ \alpha \end{array} \right.$ falls $\begin{array}{c} |\omega| \leq k \\ \omega = \alpha\beta \text{ und } |\alpha| = k \end{array}$
- $\text{Anf}_k(A \rightarrow \alpha) := \{\text{Kopf}_k(\omega) \mid S \stackrel{*}{\Rightarrow} \beta A \gamma \Rightarrow \beta\alpha\gamma \stackrel{*}{\Rightarrow} \beta\omega \in \mathcal{A}^*\}$
- G ist eine SLL(k) – Grammatik, falls $\forall A \rightarrow \alpha_1 \mid \alpha_2 \in \Pi$ mit $\alpha_1 \neq \alpha_2$: $\text{Anf}_k(A \rightarrow \alpha_1) \cap \text{Anf}_k(A \rightarrow \alpha_2) = \emptyset$

Bemerkung

S	$\cong$	stark
L	$\cong$	Eingabe wird von links nach rechts gelesen
L	$\cong$	Linksableitung
k	$\cong$	die nächsten k Eingabezeichen sind jeweils bekannt

Folgerung

1. SLL(0)$\Rightarrow$SLL(1)$\Rightarrow$SLL(2)$\Rightarrow \ldots$
2. SLL(k) $\Rightarrow$ eindeutige Grammatik
3. SLL(k) $\Rightarrow$ zu jeder Variable gibt es maximal eine Ersetzung $\Rightarrow |\mathcal{L}(G)| \leq 1$
4. Die Palindrome über $\{0, 1\}$ bilden eine kontextfreie Sprache, die für alle k nicht SLL(k) ist (ohne Beweis).
5. Eine Grammatik mit Linksrekursionen

   $$A \rightarrow A\alpha, \quad \alpha \in (\mathcal{A} \cup \mathcal{V})^*$$

   ist nicht SLL(k), falls

- $\alpha \stackrel{*}{\Rightarrow} \omega_\alpha \in \mathcal{A}^*$
- $\exists A \to \beta : \beta \neq A\alpha$ und $\beta \stackrel{*}{\Rightarrow} \omega\beta \in \mathcal{A}^*$
- $S \stackrel{*}{\Rightarrow} \gamma A\delta \stackrel{*}{\Rightarrow} \omega \in \mathcal{A}^*$.

Denn

$$\mathrm{Kopf}_k(\omega_\beta\omega_\alpha\delta) \in \mathrm{Anf}_k(A \to A_\alpha) \cap \mathrm{Anf}_k(A \to \beta)$$

Beispiel

$A \to A + T \mid T$	(arithmetischer Ausdruck)
$T \to T * F \mid F$	(Term)
$F \to (A) \mid a$	(Faktor)

Diese Grammatik hat Linksrekursionen mit den obigen Eigenschaften, ist also nicht SLL(k).
Umformung auf Rechtsrekursionen

$$\begin{aligned} A &\to T + A \mid T \\ T &\to F * T \mid F \\ F &\to (A) \mid a \end{aligned}$$

Pech: diese Grammatik ist auch nicht SLL(k) für alle k, denn

$$\mathrm{Kopf}_k(\overbrace{a * \cdots * a}^{\geq k}) \in \mathrm{Anf}_k(A \to T + A) \cap \mathrm{Anf}_k(A \to T)$$

Forme Ersetzungen mit gleichem Anfang um:

$$\begin{aligned} A &\to TA' \\ A' &\to +A \mid \lambda \\ T &\to FT' \\ T' &\to *T \mid \lambda \\ F &\to (A) \mid a \end{aligned}$$

Diese Grammatik ist SLL(1), denn

$$\begin{aligned} \mathrm{Anf}_1(A' \to +A) &= \{+\} \\ \mathrm{Anf}_1(A' \to \lambda) &= \{), \lambda\} \end{aligned}$$

und

$$\begin{aligned} \mathrm{Anf}_1(T' \to *T) &= \{*\} \\ \mathrm{Anf}_1(T' \to \lambda) &= \{), \lambda\} \end{aligned}$$

sind jeweils disjunkt.

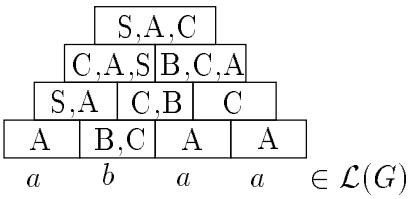


Abbildung 6.1: Pyramide für Beispiel 1

6.3 LR(k)-Grammatiken

Beispiel 1 (Zur Erinnerung: CKY-Algorithmus)

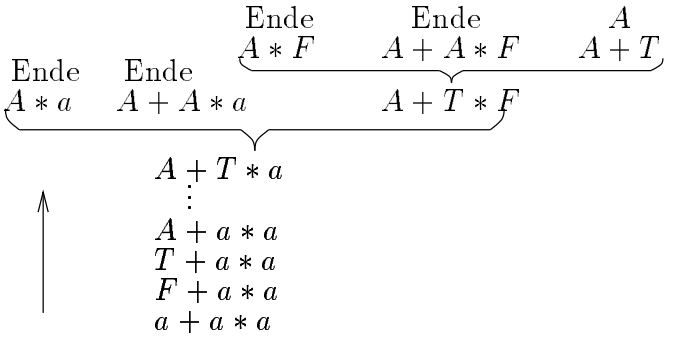
$$\begin{aligned} S &\to AB \mid BB \\ A &\to AB \mid CC \mid a \\ B &\to CA \mid BB \mid b \\ C &\to BA \mid AA \mid b \end{aligned}$$

Alle Ableitungsbäume sind aus der Pyramide (Abbildung 6.1) ablesbar. Der Aufwand beträgt $O(n^3)$.

Beispiel 2

$$\begin{aligned} A &\to A+T \mid T \\ T &\to T*F \mid F \\ F &\to (A) \mid a \end{aligned}$$

Versuche, Rechtsableitungen für $a+a*a$ zu konstruieren:



Ziel: Grammatiken, die eine deterministische Analyse ermöglichen.

$G := (\mathcal{V}, \mathcal{A}, S, \Pi)$ sei kontextfreie Grammatik.

Definition

Sei $S \underset{R}{\overset{*}{\Rightarrow}} \alpha B \gamma \underset{R}{\Rightarrow} \alpha\beta\gamma = \omega$.
Dann heißt ω eine *rechte Satzform* und β heißt *Griff* von ω.

Folgerung

Der Griff von $\alpha B\gamma$ ist entweder in γ enthalten oder enthält selber B, aber er ist nicht ganz in α. Denn

$$S \overset{*}{\underset{R}{\Rightarrow}} \alpha_1 A \alpha_3 B\gamma \underset{R}{\Rightarrow} \alpha_1\alpha_2\alpha_3 B\gamma = \alpha B\gamma$$

wäre keine Rechtsableitung.

Definition

- $\alpha\beta\gamma$ heißt *k-Kontext* der Produktion $A \to \beta$, falls

$$S \overset{*}{\underset{R}{\Rightarrow}} \alpha A\gamma\delta \underset{R}{\Rightarrow} \alpha\beta\gamma\delta \quad \text{und} \quad \gamma = \text{Kopf}_k(\gamma\delta).$$

- $\langle A \to \beta \rangle_k :=$ *Menge der k-Kontexte* von $A \to \beta$.
  Diese Menge heißt *k-Kontextmenge*.

- Die Grammatik G ist $LR(k)$, wenn kein k-Kontext Anfang eines anderen k-Kontextes ist und es keinen Zyklus $S \overset{+}{\Rightarrow} S$ gibt.

Folgerung

LR(k)-Grammatiken sind eindeutig.

Beweis (der Kontraposition)

Gibt es eine rechte Satzform $\omega \neq s$ mit zwei echten Ableitungen

$$S \overset{*}{\underset{R}{\Rightarrow}} \alpha A\gamma\delta \underset{R}{\Rightarrow} \alpha\beta\gamma\delta = \omega,$$
$$S \overset{*}{\underset{R}{\Rightarrow}} \alpha' A'\gamma'\delta' \underset{R}{\Rightarrow} \alpha'\beta'\gamma'\delta' = \omega,$$

wobei $\gamma = \text{Kopf}_k(\gamma\delta)$ und $\gamma' = \text{Kopf}_k(\gamma'\delta')$, dann ist der k-Kontext $\alpha\beta\gamma$ Anfang des k-Kontextes $\alpha'\beta'\gamma'$ oder umgekehrt. Gibt es zwei Ableitungen für $\omega = s$, gibt es einen Zyklus $S \overset{+}{\Rightarrow} S$. Also sind mehrdeutige Grammatiken nicht LR(k) bzw. LR(k)-Grammatiken eindeutig.

6.4 LR(k)-Analyse

Gegeben sei eine Grammatik $G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ mit der Eigenschaft LR(k). Weiter sei $\omega = a_1 \ldots a_n \in \mathcal{A}^*$ und $a_{n+1} = \ldots = a_{n+k} = \lambda$.
Mit Hilfe des folgenden Algorithmus kann überprüft werden, ob ω in $\mathcal{L}(G)$ liegt:

$\alpha := \lambda, \quad \beta := \lambda, \quad \gamma := a_1 \dots a_k$
while $\alpha\beta\gamma \neq S$ **do**
begin
  if $\alpha\beta\gamma \in \langle A \to \beta \rangle_k$ für ein $A \to \beta \in \Pi$
  then $\beta := A$
  elseif $\alpha = b_1 \dots b_l \neq \lambda$
    and $|\beta| < m := \max\{|\delta| \mid A \Rightarrow \delta \in \Pi\}$
  then $\alpha := b_1 \dots b_{l-1}$
    $\beta := b_l\beta$
  elseif $\gamma = a_{l+1} \dots a_{l+k} \neq \lambda$
  then $\gamma := a_{l+2} \dots a_{l+k+1}$
    $\alpha := \alpha\beta a_{l+1}$
    $\beta := \lambda$
  else Ausgabe : $\omega \notin \mathcal{L}(G)$; **stop**
end
Ausgabe : $\omega \in \mathcal{L}(G)$

Problem: Die k-Kontextmengen können unendlich groß sein.

6.5 Linke Kontexte

Sei $G := (\mathcal{V}, \mathcal{A}, S, \Pi)$ eine kontextfreie Grammatik.
Der *linke Kontext* von $A \in \mathcal{V}$ ist

$$\langle A \rangle := \{\alpha \mid S \underset{R}{\overset{*}{\Rightarrow}} \alpha A\gamma, \gamma \in \mathcal{A}^*\}$$

und der linke Kontext einer Produktion $A \to \beta$ ist

$$\langle A \to \beta \rangle = \langle A \rangle \, \beta = \langle A \to \beta \rangle_0 .$$

Beispiel

Für $S \to aAb$, $A \to aAa \mid ab$ ist $\langle A \rangle = a^+$ und $\langle A \to aAa \rangle = aa^+Aa$.

Satz

Für alle Produktionen $A \to \alpha \in \Pi$ sind $\langle A \rangle$ und $\langle A \to \alpha \rangle$ reguläre Sprachen.

Beweis

Sei $\mathcal{V} = \{A_1, \dots, A_n\}$ und seien

$$B_{ij} \to \alpha_{ij} A_i \beta_{ij}, \quad j = 1, \dots, k_i$$

alle Produktionen, in denen A_i auf der rechten Seite vorkommt.

Dann ist $\langle A_i \rangle = \epsilon_i + \langle B_{i1} \rangle \alpha_{i1} + \cdots + \langle B_{ik_i} \rangle \alpha_{ik_i}$,
wobei $\epsilon_i := \begin{cases} \lambda & \text{falls } A_i = S \\ \emptyset & \text{sonst} \end{cases}$.
Diese Gleichungen beschreiben eine linkslineare Grammatik. Die reguläre Sprache, die sich aus einer Variablen „$\langle A_i \rangle$“ ableiten läßt, ist der linke Kontext von A_i.

Beispiel

$$\begin{aligned} S &\to R \mid [S] \\ R &\to A = A \\ A &\to [A \wedge A] \mid a \mid b \end{aligned}$$

Die zugehörigen Kontextgleichungen sind

$$\begin{aligned} \langle S \rangle &= \langle S \rangle\, [+ \lambda \\ \langle R \rangle &= \langle S \rangle \\ \langle A \rangle &= \overbrace{\langle R \rangle\, (\lambda A =)}^{U} + \langle A \rangle\, \overbrace{([+ [A \wedge)}^{V} \\ &= U + \langle A \rangle\, V \end{aligned}$$

Lösung

$$\begin{aligned} \langle S \rangle &= [^* \\ \langle R \rangle &= [^* \\ \langle A \rangle &= UV^* \end{aligned}$$

Also

$$\begin{aligned} \langle S \to R \rangle &= [^* R \\ \langle S \to [S] \rangle &= [^+ S] \\ \langle R \to A = A \rangle &= [^* A = A \\ \langle A \to [A \wedge A] \rangle &= UV^* [A \wedge A] \\ \langle A \to a \rangle &= UV^* a \\ \langle A \to b \rangle &= UV^* b \end{aligned}$$

Offensichtlich ist kein 0-Kontext Anfang eines anderen 0-Kontextes. D.h. die Grammatik ist LR(0). Außerdem sind die Kontextmengen disjunkt.

Bemerkung

Sei g die Ersetzungsfunktion zu den Kontextgleichungen. Dann gilt nach 5.3

$$\sup g^k (\emptyset, \ldots, \emptyset) = (\langle A_1 \rangle, \ldots, \langle A_n \rangle).$$

6.6 Kontextmengen

$G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ sei kontextfreie Grammatik.

Definition

- Falls

$$S \underset{R}{\overset{*}{\Rightarrow}} \alpha B \gamma \underset{R}{\Rightarrow} \alpha \beta \gamma,$$

  dann heißt $\alpha\beta\text{Kopf}_k(\gamma)$ der r,k-Kontext von $B \to \beta \in \Pi$, wobei $r = |\text{Kopf}_k(\gamma)|$.

- $[B \to \beta]_{r,k} = \{r,k - \text{Kontexte von } B \to \beta\}$

Bemerkung

Die verschiedenen r,k-Kontexte einer Produktion $B \to \beta$ bilden eine Zerlegung des k-Kontextes von $B \to \beta$.
Zur Erinnerung:
LR(k) genau dann, wenn kein r,k-Kontext Anfang eines anderen s,k-Kontextes ist.

Satz

Jede r,k-Kontextmenge $[B \to \beta]_{r,k}$ ist regulär.

Beweis

Jeder Ableitung in G

$$S \underset{R}{\overset{*}{\Rightarrow}} \alpha_1 A \gamma_1 \Rightarrow \alpha_1\alpha_2 B \gamma_2\gamma_1 \underset{R}{\overset{*}{\Rightarrow}} \alpha_1\alpha_2 B\gamma \underset{R}{\Rightarrow} \alpha_1\alpha_2\beta\gamma$$

ordnen wir eine rechtslineare Ableitung zu:

$$S_\lambda \overset{*}{\Rightarrow} \alpha_1 A_{\overline{\gamma}_1} \Rightarrow \alpha_2 B_{\overline{\gamma}} \Rightarrow \alpha_2\beta\overline{\gamma},$$

wobei $\overline{\omega} = \text{Kopf}_k(\omega)$.
Diese rechtslineare Ableitungen definieren eine Grammatik $G' = (\mathcal{V}', \mathcal{A}', S_\lambda, \Pi')$ mit

$$\begin{aligned} \mathcal{V}' &:= \{A_{\overline{\gamma}} \mid A \in \mathcal{A}, \gamma \in \mathcal{A}^*\} \\ \mathcal{A}' &:= \mathcal{A} \cup \mathcal{V} \end{aligned}$$

und den Produktionen

$$A_{\overline{\gamma}} \to \alpha C_{\overline{\omega}} \quad \text{für} \quad A \to \alpha C \delta \text{ und } \delta\overline{\gamma} \overset{*}{\Rightarrow} \omega \in \mathcal{A}^*$$

sowie den terminalen Produktionen

$$B_{\overline{\gamma}} \to \beta\overline{\gamma}, \quad |\overline{\gamma}| = r \text{ und } \overline{\gamma} \in \mathcal{A}^*.$$

Dann ist $\mathcal{L}(G') = [B \to \beta]_{r,k}$.

6.7 Kellerautomaten für LR(k)-Grammatiken

$G = (\mathcal{V}, \mathcal{A}, S, \Pi)$ sei eine LR(k)-Grammatik. Die Produktionen seien

$$A_i \to \alpha_i, \quad i = 1, \ldots, n$$

(Dabei taucht jedes $A \in \mathcal{V}$ unter den A_i so oft auf, wie es Produktionen mit A auf linker Seite gibt).
Sei $M_{i,r} = (Q_{i,r}, \mathcal{A}, \delta_{i,r}, q_{i,r}, \mathcal{F}_{i,r})$ ein DFA, der $[A_i \to \alpha_i]_{r,k}$ erkennt, und sei

$$M = (Q, \mathcal{A}, \varphi, q_0, \mathcal{F})$$

der Produktautomat mit

$$\begin{aligned} Q &= Q_{1,0} \times \cdots \times Q_{1,k} \times Q_{2,0} \times \cdots \times Q_{n,k}, \\ q_0 &= (q_{1,0}, \ldots, q_{n,k}), \\ \mathcal{F} &= \mathcal{F}_{1,0} \times \cdots \times \mathcal{F}_{n,k}, \\ \varphi((p_{1,0}, \ldots, p_{n,k}), x) &= (\varphi_{1,0}(p_{1,0}, x), \ldots, \varphi_{n,k}(p_{n,k}, x)). \end{aligned}$$

Bemerkung

- Da G eine LR(k)-Grammatik ist, sind die r,k-Kontextmengen disjunkt. Deshalb ist $\mathcal{L}(M) = \emptyset$. M wird nicht zum Akzeptieren benutzt.
- Die Zustände $q \in Q$ entsprechen den Äquivalenzklassen auf $(\mathcal{A} \cup \mathcal{V})^*$.

$$\begin{aligned} q &\stackrel{\wedge}{=} \{\omega \in (\mathcal{A} \cup \mathcal{V})^* \mid \varphi^*(q_0, \omega) = q\} \\ &=: [\omega]. \end{aligned}$$

- M ist endlich und deterministisch.

Satz

Ist G eine LR(k)-Grammatik, kann $\mathcal{L}(G)$ durch einen deterministischen Kellerautomaten akzeptiert werden.

Beweis

Sei ω Eingabe für den in Abbildung 6.2 gezeigten Kellerautomaten. Für die Kellerzeichen (b_i, p_i) soll gelten

$$\begin{aligned} b_i &\in \mathcal{A} \cup \mathcal{V} \\ p_i &= (p_i^{1,0}, \ldots, p_i^{n,k}) = \varphi(p_{i-1}, b_i) \\ &= [b_0 \ldots b_i]. \end{aligned}$$

Wenn ein Zeichen gekellert wird, kann p_i aus p_{i-1} berechnet werden (Nachschauen in einer endlichen Matrix).

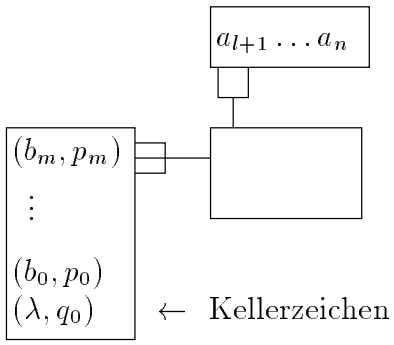


Abbildung 6.2: Kellerautomat

Aktionen des Kellerautomaten:

- Ist $p_m^{i,r} \notin \mathcal{F}_{i,r}$ für alle i, r, dann lies und kellere a_{l+1}.
- Ist $p_m^{i,r} \in \mathcal{F}_{i,r}$, dann ist $b_0 \ldots b_m \in [A_i \to \alpha_i]_{r,k}$, d.h. $\alpha_i = b_s \ldots b_{m-r}$ für ein gewisses s. Ersetze $b_s \ldots b_m$ durch $A_i b_{m+1-r} \ldots b_m$, und berechne dabei die entsprechenden Zustände $p_i \in Q$.

Der Automat erkennt alle $\omega \in \mathcal{A}^*$ mit

$$(\lambda, q_o) \text{ Anfangszustand } \omega \stackrel{*}{\Rightarrow} (\lambda\, q_0)(S, \varphi(q_0, S)) \text{ Zustand } \lambda.$$

Das sind die $\omega \in \mathcal{L}(G)$.

Folgerung

Die Sprachen mit LR(k)-Grammatiken sind unter Komplementbildung abgeschlossen.

Beweisidee

Der deterministische Kellerautomat akzeptiert oder verwirft in endlich vielen Schritten. D.h. es gibt auch einen deterministischen Kellerautomaten für die komplementäre Sprache.

Folgerung

Es gibt kontextfreie Sprachen, die nicht LR(k) sind.

Beweis

Kontextfreie Sprachen sind nicht unter dem Komplement abgeschlossen.

Teil II

Berechenbarkeitstheorie

Kapitel 7

Berechenbarkeitstheorie

7.1 Intuitiver Berechenbarkeitsbegriff

Intuitive Definition

Eine totale oder partielle Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ heißt intuitiv berechenbar, falls es einen Algorithmus gibt, der f berechnet. Bei Eingabe eines $\mathbf{a} \in \mathbb{N}_0^k$ soll der Algorithmus nach endlich vielen Schritten $f(\mathbf{a})$ ausgeben, oder, falls $f(\mathbf{a}) = \perp$, soll der Algorithmus nicht terminieren.

Die Schritte des Algorithmus sollen deterministisch, mechanisch und in endlicher Zeit ausführbar sein.

Aber: Speicher und Rechenzeit sind unbegrenzt.

These von Church

Jede intuitiv berechenbare Funktion ist durch eine Turingmaschine berechenbar.

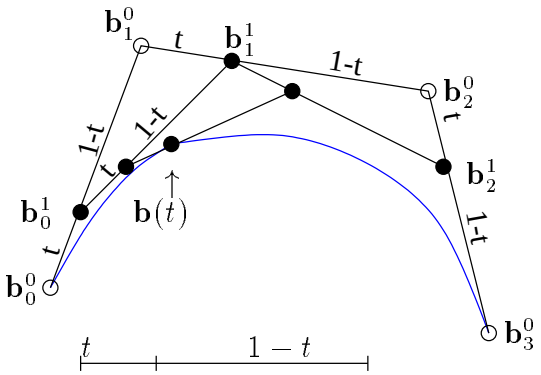


Abbildung 7.1: DE CASTELJAU-Algorithmus

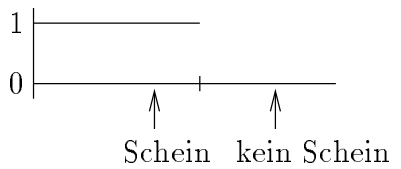


Abbildung 7.2: Informatiker in der Vorlesung

Beispiel (de Casteljau-Algorithmus)

Gegeben seien $\mathbf{b}^0, \ldots, \mathbf{b}^0_n \in \mathbb{R}^d, \quad t \in \mathbb{R}$ (siehe auch Abbildung 7.1).

$$\begin{array}{l} \texttt{for } i = 1, \ldots, n \\ \quad \texttt{for } j = 0, \ldots, n-i \\ \quad\quad \mathbf{b}^i_j := (1-t)\,\mathbf{b}^{i-1}_j + t\,\mathbf{b}^{i-1}_{j+1} \\ \texttt{Ausgabe } \mathbf{b}(t) = \mathbf{b}^n_0 \end{array}$$

Dieser Algorithmus berechnet die polynomiale Kurve

$$\mathbf{b}(t) = \sum_{i=0}^{n} \mathbf{b}^0_i \left(\begin{array}{c} n \\ i \end{array} \right) t^i (1-t)^{n-i}\,.$$

.

Beispiel

$f(n) = \begin{cases} 1 \text{ falls } n \text{ Leute die Info} - \text{III} - \text{Klausur bestehen} \\ 0 \text{ sonst} \end{cases}$
ist berechenbar (siehe Abbildung 7.2).

Beispiel

$\texttt{halt}(n)$ ist nicht berechenbar.

Beispiel

$f(n) = \begin{cases} 1 \text{ falls LBAs und deterministische LBAs gleichmächtig sind} \\ 0 \text{ sonst} \end{cases}$
ist berechenbar.

7.2 Turing-Berechenbarkeit

Definition

Eine Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ ist Turing-berechenbar, falls es eine Turingmaschine gibt mit:

$f(x_1, \ldots, x_k) = (y_1, \ldots, y_n)$ genau dann, wenn

$q_0 1^{x_0} \square 1^{x_2} \square \ldots \square 1^{x_k} \square \quad \overset{*}{\Rightarrow} \quad$ Endzustand $1^{y_1} \square \ldots \square 1^{y_m} \square$

Turingmaschine

Beispiele

für Turing-berechenbare Funktionen:

- $f(n) = n + 1$
- $f(n) = \perp$
- $f(n) = \begin{cases} 1 \text{ falls } 1^n \in \mathcal{L} \\ \perp \text{ sonst} \end{cases}$, wobei $\mathcal{L}$ eine Typ-0-Sprache ist
- $f \circ g$, wenn f und g Turing-berechenbar
- $f(n_1, \ldots, n_k) = (n_1, \ldots, n_k) + (0, \ldots, 0, 1, 0, \ldots, 0)$
- $f(n_1, \ldots, n_k) = (n_1, \ldots, n_k) - (0, \ldots, 0, 1, 0, \ldots, 0)$, wobei $0 - 1 = 0$
- $f(n_1, \ldots, n_k) := (n_1, \ldots, n_{i-1}, n_j, n_{j+1}, \ldots, n_k)$, d.h. : $(x_i := x_j)$
- `while` $n_i \neq 0$ `do` $(n_1, \ldots, n_k) := f(n_1, \ldots, n_k)$ `end`, wobei f Turing-berechenbar

7.3 loop- und while-Berechenbarkeit

`while`-Programme bestehen aus

Variablen	:	$x_0, x_1, \ldots$
Konstanten	:	$0, 1, 2, \ldots$
Zeichen	:	$; := + -$
Schlüsselwörtern	:	`while`, `do`, `end`

und sind von folgender rekursiv definierter Form:

- $x_i := x_j + c$
- $x_i := x_j - c$
- $P_1; P_2$
- `while` $x_i \neq 0$ `do` P_1 `end`

wobei P_1 und P_2 `while`-Programme sind und c eine Konstante.

Semantik

`while`-Programme haben die übliche Bedeutung mit der Ausnahme:

$$x_i := x_j - c = 0, \text{ falls } c > x_j.$$

Eine Funktion $f : \mathbb{N}_0^k \to \mathbb{N}_0^m$ heißt `while`-berechenbar, falls es ein `while`-Programm gibt, das gestartet mit $x_1 = n_1, \ldots, x_k = n_k$ und 0 in den übrigen Variablen genau dann stoppt, wenn $f(n_1, \ldots, n_k)$ definiert ist und dann $(x_1, \ldots, x_m) = f(n_1, \ldots, n_k)$ gilt.

Beispiel

$f(x_1, x_2) = (x_1, x_1)$ wird durch das `while`-Programm $x_2 := x_1 + 0$ realisiert.

Beispiel/Definition

`loop` x `do` P `end`, wobei P ein `while`-Programm ist, wird berechnet durch das `while`-Programm

$$y := x$$
$$\texttt{while } y \neq 0 \texttt{ do } P;\ y = y - 1 \texttt{ end}$$

wobei y nicht in P vorkommt.

Beispiel für eine nicht terminierende `while`-Schleife

$$x := 1$$
$$\texttt{while } x \neq 0 \texttt{ do } x = x + 1 \texttt{ end}$$

Definition

Ein `loop`-Programm ist ein `while`-Programm, dessen `while`-Schleifen `loop`-Schleifen sind.

Folgerung

`loop`-berechenbar $\Rightarrow$ total und `while`-berechenbar.
Die Umkehr gilt nicht, denn `loop`-Programme terminieren immer, `while`-Programme hingegen nicht.

7.4 Makros

Die Programme
$x := y$ und `loop` x `do` P `end`
sind keine `while`-Programme, aber durch `while`-Programme berechenbar. Sie heißen *Makros*.

Weitere Makros

$$\begin{aligned}
z &:= x - y \qquad (\text{d.h. } z = 0, \text{ falls } y > x)\\
z &:= x \cdot y\\
z &:= x + y\\
z &:= x^y\\
z &:= x \bmod y \qquad (\text{Rest von } x/y)\\
z &:= x \operatorname{div} y \qquad \lfloor x/y \rfloor\\
z &:= \log_2 x \qquad = \lfloor \log_2 x \rfloor\\
z &:= n, \quad \text{wobei } n \in \mathbb{N}_0\\
z &:= (x \cdot y) \cdot n\\
z &:= m \cdot n\\
&\text{usw.}
\end{aligned}$$

Aufgabe

Gib `loop`-Programme für diese Makros an.

Testmakros (rekursive Definitionen)

1. $\alpha < \beta$ ist ein Testmakro, wobei $\alpha, \beta \in \mathbb{N}_0$ oder Variable
2. Sind S und T Testmakros, dann auch

$$(S \wedge T), (S \vee T), \neg S$$

Beispiel

$$\begin{array}{lcl} \alpha \neq \beta & : & (\alpha < \beta) \vee (\beta < \alpha) \\ \alpha = \beta & : & \neg(\alpha \neq \beta) \\ \alpha > \beta & : & \beta < \alpha \\ \alpha \leq \beta & : & (\alpha < \beta) \vee (\alpha = \beta) \end{array}$$

Semantik

Testmakros haben die übliche Bedeutung.

Lemma

Zu jedem Testmakro T gibt es einen arithmetischen Ausdruck A_T mit

$$A_T = \left\{ \begin{array}{l} 0 \\ 1 \end{array} \right. \text{ falls } \begin{array}{l} T \text{ falsch} \\ T \text{ wahr} \end{array}$$

Beweis (durch rekursive Konstuktion)

1. $A_{\alpha<\beta} = (\beta - \alpha) - ((\beta - \alpha) - 1)$
2. $A_{(S \wedge T)} = A_S + A_T - 1 = A_S \cdot A_T$
3. $A_{(S \vee T)} = A_S + A_T - A_S \cdot A_T$
4. $A_{\neg S} = 1 - A_S$

Anwendung

Seien P, Q `while`-Programme und T ein Testmakro, dann werden

1. `while` T `do` P `end`
2. `if` T `then` P `end`
3. `if` T `then` P `else` Q `end`
4. `repeat` P `until` T `end`

durch `while`-Programme berechnet:

1. $x := A_T$
   `while` $x \neq 0$ `do` P; $x := A_T$ `end`
2. $x := 1$
   `while` $T \wedge (x = 1)$ `do` P; $x = 0$ `end`
3. $x := 0$
   `if` T `then` P; $x = 1$ `end`;
   `if` $x = 0$ `then` Q `end`
4. P; `while` $\neg T$ `do` P `end`

7.5 Vergleich von `while`- und Turing-Berechenbarkeit

Satz

`while`-Programme sind durch eine Turingmaschine simulierbar.

Beweis

siehe Abschnitt 7.2.

Umkehr

Turingmaschinen sind durch `while`-Programme simulierbar.

Beweis

$B = \{b_0, \ldots, b_{p-1}\}$ sei das Bandalphabet und $Q = \{q_0, \ldots, q_r\}$ sei die Zustandsmenge einer Turingmaschine. Ein Wort $\omega = b_{i_1} \ldots b_{i_m} \in B^*$ entspricht eineindeutig einer Zahl

$$\overline{\omega} = \sum_{k=1}^{m} i_k \cdot p^{k-1} \in \mathbb{N}.$$

Eine Konfiguration $\alpha q_i \beta$ wird kodiert mit den 3 Zahlen

$$\overline{\alpha}, i, \overline{\beta}.$$

Die Konfigurationsübergänge können mit einem `while`-Programm berechnet werden, weil folgende Funktionen `while`-berechenbar sind:

$$\begin{aligned} f(\overline{\omega}) &= i_k, \text{ für ein beliebiges festes } k, \\ f(\overline{\omega}) &= \overline{\omega} \text{ div } p \text{ (Streichen von } b_{i_1}), \\ f(\overline{\omega}) &= \overline{b_{i_1} \ldots b_{i_{m-1}}} \text{ (Streichen des letzten Zeichens).} \end{aligned}$$

Folgerung

Turing-berechenbar $\Leftrightarrow$ `while`-berechenbar

Beweis

Auch

$$f(x_1,\ldots,x_k) = \overline{\underbrace{1\ldots1}_{x_1} \square \underbrace{1\ldots1}_{x_2} \square \ldots \underbrace{1\ldots1}_{x_k}}$$

und die Umkehrfunktion sind `while`-berechenbar.

7.6 goto-Programme

`goto`-Programme sind Folgen (teilweise) numerierter Anweisungen

$$1.\, A_1, \ldots, n.\, A_n$$

Mögliche Anweisungen sind
Zuweisungen

$$\begin{aligned} x_i &:= x_j + c, \\ x_i &:= x_j - c, \end{aligned}$$

unbedingte Sprünge

$$\texttt{goto}\ k,$$

bedingte Sprünge

$$\texttt{if}\ x_i = c\ \texttt{then}\ k,$$

Stopanweisung

$$\texttt{stop}$$

Satz

Jedes `while`-Programm kann durch ein `goto`-Programm simuliert werden.

Beweis

Eine `while`-Schleife

$$\texttt{while}\ x \neq 0\ \texttt{do}\ P\ \texttt{end}$$

wird berechnet durch

$$\begin{array}{ll} i. & \texttt{if}\ x = 0\ \texttt{then}\ j \\ & P \\ & \texttt{goto}\ i \\ j. & \ldots \end{array}$$

Umkehr

`goto`-Programme können durch `while`-Programme berechnet werden.

Beweis

Ein `goto`-Programm

$$1.\, A_1, \ldots, n.\, A_n$$

wird berechnet durch

```
k := 1
while k ≠ 0 do
  if k = 1 then B_1 end
    ⋮
  if k = n then B_n end
end
```

wobei

$$B_i = \begin{cases} A_i;\, k := k+1 & \text{falls } A_i \text{ eine Zuweisung} \\ k := m & \text{falls } A_i = \texttt{goto } m \\ \texttt{if } x_i = c \texttt{ then } k := m \texttt{ else } k := k+1 & \text{falls } A_i = \texttt{if } x_i = c \texttt{ then goto } m \\ k := 0 & \text{falls } A_i = \texttt{stop} \end{cases}$$

Folgerung

Jedes `while`-Programm kann durch ein `while`-Programm mit nur einer `while`-Schleife und mehreren `if-then`-Makros simuliert werden.

7.7 Zusammenfassung

Turing-berechenbar $\Leftrightarrow$ `while`-berechenbar $\Leftrightarrow$ `goto`-berechenbar

$\Uparrow$

`loop`-berechenbar

7.8 Die Ackermannfunktion

Die Ackermann-Funktion ist total und `while`-berechenbar, aber nicht `loop`-berechenbar.

Definition

$$\begin{aligned} a(0,y) &:= 3y+1 \\ a(x,0) &:= 1 \\ a(x+1,y) &:= \overbrace{a(x,a(x,\ldots a(x,y)\ldots))}^{y-\text{mal}} \end{aligned}$$

Dies ist die Funktion von Ackermann (1928) in modifizierter Form (siehe Tabelle 7.1).

x \ y	0	1	2	3	4	5	6	7
0	1	4	7	10	13	16	19	22
1	1	4	22					
2	1	4						
3	1	4						

Tabelle 7.1: $a(x, y)$

Bemerkung

$a(x, y)$ ist mit Hilfe eines rekursiven `while`-Programms berechenbar. Daraus folgt mit Hilfe des Rekursionssatzes in 10.4, daß $a(x, y)$ `while`-berechenbar ist.

Lemma

$a(x, y)$ ist monoton in beiden Variablen.

Beweis als Übung.

Sei P ein `loop`-Programm mit genau k Variablen und $\varphi(\mathbf{x}) : \mathbb{N}_0^k \to \mathbb{N}_0^k$ die von P berechnete Funktion.

Definition

$$f_p(n) := \max\{\|\varphi(\mathbf{x})\|_\infty \mid \|\mathbf{x}\|_\infty \leq n\}$$

Bemerkung

$f_p(n)$ ist monoton.

Satz

Es gibt eine Konstante $q \in \mathbb{N}_0 : \forall n \geq q : f_p(n) < a(q, n)$.

Beweis (durch rekursive Induktion)

Falls $P = x_i := x_j + c$, gilt

$$\begin{aligned} f_p(n) &\leq 2n + c \\ &< 3n + 1 = a(1, n) \end{aligned}$$

für $n \geq c$. Wähle $q = c$.

Falls $P = P_1; P_2$, dann existiert nach Induktionsannahme ein $c \in \mathbb{N}_0$ mit $f_{P_i}(n) < a(c,n)$ für alle $n \geq c$ und $i = 1, 2$. Also folgt für $n \geq c$ unter Ausnutzung der Monotonie von $a(x,y)$

$$\begin{aligned} f_P(n) &\leq f_{P_2}(f_{P_1}(n)) \\ &\leq a(c, f_{P_1}(n)) \\ &\leq a(c, a(c,n)) \end{aligned}$$

und weiter, da $n \leq a(c,n)$

$$\begin{aligned} f_P(n) &\leq a(c, a(c, a(c,n))) \\ &\vdots \\ &\leq \overbrace{a(c, \ldots a(c,n)}^{n-\text{mal}} \ldots) \\ &= a(c+1, n) \end{aligned}$$

Wähle $q = c + 1$.
Falls $P =$ `loop` x; `do` Q `end`, dann existiert ein $c \in \mathbb{N}_0$ mit $f_Q(n) < a(c,n)$ für alle $n \geq c$. Also folgt für $n \geq c$

$$\begin{aligned} f_P(n) &\leq \overbrace{f_Q(f_Q(\ldots f_Q(n)}^{x-\text{mal}} \ldots)) \\ &\leq \overbrace{f_Q(f_Q(\ldots f_Q}^{n-\text{mal}}(n) \ldots)) \\ &< \overbrace{a(c, \ldots, a(c,n)}^{n-\text{mal}} \ldots) \\ &= a(c+1, n) \end{aligned}$$

Wähle $q = c + 1$.

Folgerung

$a(x,y)$ ist nicht `loop`-berechenbar.

Kapitel 8

Universelle Funktionen

Es existiert eine algorithmische Aufzählung der `while`-Programme $P_0, P_1, \ldots$ und der berechenbaren Funktionen $\varphi_0, \varphi_1, \ldots$ Dazu existiert eine berechenbare Funktion φ mit

$$\varphi(i, x_1, \ldots, x_n) = \varphi_i(x_1, \ldots, x_n).$$

8.1 Aufzählung der `while`-Programme

Jedes `while`-Programm kann anhand folgender Tabelle als Zahlenfolge beziehungsweise natürliche Zahl interpretiert werden.

0	100	`while`	110	;	116
1	101	`do`	111	x	117
$\vdots$	$\vdots$	`end`	112	$\neq$	118
9	109	$+$	113		
		$-$	114		
		$:=$	115		

Diese Zahl heißt *Index* (oder *Gödel-Nummer*) des `while`-Programms. Den Index eines `while`-Programms P bezeichnen wir kurz mit "P".

Definition

$$P := \begin{cases} x_1 := x_1 + 0 & \text{falls } n \text{ kein Index eines } \texttt{while} - \text{Programms} \\ P & \text{falls } n = \text{"}P\text{"}. \end{cases}$$

Beispiel

1. $n = 111110$ ist die Kodierung für `do while`. Also ist $P_n = \ x_1 := x_1 + 0$.
2. $n = 13$ kodiert kein `while`-Programm. Also ist $P_n = \ x_1 := x_1 + 0$

Definition

$P_0, P_1, \ldots$ heißt *algorithmische* (*effektive* oder *rekursive*) Aufzählung der `while`-Programme, weil sie schrittweise und mechanisch ausgegeben werden kann.

Bemerkung

Es gibt überabzählbar viele Aufzählungen der `while`-Programme, aber nur abzählbar viele sind algorithmisch.

Definition

$\varphi_i^{k,m}$ bezeichnet die Funktion $\mathbb{N}_0^k \to \mathbb{N}_0^m$, die von P_i berechnet wird, d.h. ist $\varphi_i^{k,m}(a_1 \dots a_k) = (b_1 \dots b_m)$ und wird P_i mit den Variablenbelegungen

$$x_i := \begin{cases} a_i & \text{falls } 1 \leq i \leq k \text{ und } x_i \text{ Variable von } P_i \text{ ist} \\ 0 & \text{sonst,} \end{cases}$$

gestartet, dann terminiert P_i mit den Variablenbelegungen

$$x_i := \begin{cases} b_i & \text{falls } 1 \leq i \leq m \\ 0 & \text{sonst} \end{cases} .$$

Ist $\varphi_i^{k,m}(a_1, \dots, a_k) = \perp$, dann terminiert P_i bei entsprechender Eingabe nicht. Die Folge $\varphi_0^{k,m}, \varphi_1^{k,m}, \dots$ ist die *Standardaufzählung der berechenbaren Funktionen* $\mathbb{N}_0^k \to \mathbb{N}_0^m$.

Abkürzungen

$$\begin{aligned} \varphi_i^k &:= \varphi_i^{k,1}, \\ \varphi_i &:= \varphi_i^{1,1}, \\ \varphi_i(x_1, \dots, x_k) &:= \varphi_i^{k,1}(x_1, \dots, x_k) \end{aligned}$$

8.2 Indexüberprüfung

Satz

$$\texttt{korrekt}(n) = \begin{cases} 1 & \text{falls } n \text{ Gödel} - \text{Nummer} \\ 0 & \text{sonst} \end{cases} \text{ ist } \texttt{while}\text{-berechenbar.}$$

Beweisskizze

Folgendes Programm berechnet `korrekt`. Dabei enthält x zuerst die Ein- und zum Schluß die Ausgabe. i, j, c stehen für Ziffernfolgen bestehend aus den

Blöcken $100, \ldots, 109$.

$y := 1$
`while` $y \neq 0$ `do`
  `if` "$x =$ $x_i := x_j + c$; P" `or`
    $x =$ "$x_i := x_j - c$; P"
  `then` $x :=$ "P"
  `elseif` $x =$ "`while` $x_i \neq 0$ `do` P `end` Q"
    `and` `while` $-$ `end` $-$ Klammerung korrekt
  `then` $x :=$ "$P; Q$"
  `else` $y := 0$
`end`
`if` $x =$ "$x_i := x_j \pm c$"
`then` $x := 1$
`else` $x := 0$

Aufgabe

Zeige, daß das skizzierte Programm durch ein `while`-Programm realisierbar ist.

8.3 Kompakte `while`-Programme

Lemma

Die Funktionen

$$
\begin{aligned}
f_m(x_1, \ldots, x_m) &= \overbrace{1\ldots1}^{x_m}\,0\,\overbrace{1\ldots1}^{x_{m-1}}\,0\ldots0\,\overbrace{1\ldots1}^{x_1}, \\
g(x, i) &= \begin{cases} x_i & \text{falls } x = f_m(x_1, \ldots, x_m) \\ \bot & \text{sonst} \end{cases}, \\
h_+(x, i, j, c) &:= \begin{cases} \bot \text{ falls } x \neq f_m(x_1, \ldots x_m) \\ \overbrace{1\ldots1}^{x_m}\,0\ldots0\,\overbrace{1\ldots1}^{x_{i+1}}\,0\,\overbrace{1\ldots1}^{x_j+c}\,0\,\overbrace{1\ldots1}^{x_{i-1}}\,0\ldots0\,\overbrace{1\ldots1}^{x_1} \\ \quad \text{falls } x = f_m(x_1, \ldots x_m) \end{cases}, \\
h_-(x, i, j, c) &\quad \text{analog}
\end{aligned}
$$

sind berechenbar.

Beweis

als Aufgabe.

Satz

Es gibt eine Konstante K, so daß für alle i, k, m ein `while`-Programm P_{ikm} mit $K + \max\{k, m\}$ Variablen existiert, das $\varphi_i^{k,m}$ berechnet.

Beweis

P_{ikm} ist gegeben durch:

$$\begin{aligned} & x := f(x_1, \ldots x_k); \\ & \overline{P}_i; \\ & x_1 := g(x, 1); \\ & \vdots \\ & x_m := g(x, m) \end{aligned}$$

wobei $\overline{P}_i$ aus P_i entsteht, indem alle Anweisungen

$$x_i := x_j \pm c$$

ersetzt werden durch

$$h_{\pm}(x, i, j, c)$$

und $x_i \neq 0$ durch $g(x, i) \neq 0$.

Folgerung

$\texttt{kompakt}_{km}(n)$ = Index von P_{nkm} ist berechenbar.

Beweis

als Übung.

8.4 Der Interpreter

In einem `while`-Programm denken wir uns alle Anweisungen

$$\begin{aligned} & x_i := x_j + c \\ & x_i := x_j - c \\ & x_i \neq 0 \\ & \texttt{end} \end{aligned}$$

durchnumeriert.
Damit sind folgende Funktionen, die nur für Indexe x von `while`-Programmen definiert sind, berechenbar:

$$\texttt{anw}(x, i) := \begin{cases} \text{“}i - \text{te Anweisung von } P_x\text{”}, \\ \quad \text{falls } P_x \text{ mindestens } i \text{ Anweisungen hat} \\ 0 \text{ sonst} \end{cases}$$

$$\texttt{end}(x, i) := \begin{cases} \text{die Nummer des zugehörigen } \texttt{ends}, \\ \quad \text{falls } \texttt{anw}(x, i) \text{ eine Abfrage } x_i \neq 0 \text{ kodiert} \\ \perp \text{ sonst} \end{cases}$$

$$\texttt{while}(x, i) := \begin{cases} \text{die Nummer der zugehörigen Abfrage } x_i \neq 0, \\ \quad \text{falls } \texttt{anw}(x, i) = \text{“}\texttt{end}\text{”} \\ \perp \text{ sonst} \end{cases}$$

Satz

$\Phi^{k\,m}(x, x_1, \ldots, x_k) := \varphi_x^{k,m}(x_1, \ldots, x_k)$ ist berechenbar.

Beweis

Folgendes Programm berechnet $\Phi^{k\,m}$. Dabei enthalten die Variablen $x, x_1, \ldots, x_k$ die Eingabe und $x_1, \ldots, x_m$ die Ausgabe.

```
if korrekt(x) = 0
then berechne x_1 := x_1 + 0
else i := 1
   y := f(x_1, ..., x_k)   {Zur Definition von f siehe 8.3}
while i ≠ 0 do
   if anw(x, i) = "x_i := x_j ± c"
   then h_±(x, i, j, c); i := i + 1
   if anw(x, i) = "x_i ≠ 0" and g(y, i) ≠ 0
   then i := i + 1
   else i := end(x, i) + 1
   if anw(x, i) = "end"
   then i := while(x, i)
   if anw(x, i) = 0
   then i := 0
end;
x_1 := g(y, 1);
⋮
x_m := g(y, m)
```

Bezeichnung

$\Phi^{k,m}$ heißt *universelle Funktion*, das Programm für $\Phi^{k,m}$ ein *Interpreter*.

Abkürzungen

$$\Phi := \Phi^{1,1},$$
$$\Phi^k := \Phi^{k,1},$$
$$\Phi(i, x_1, \ldots, x_k) := \Phi^{k,1}(i, x_1, \ldots, x_k)$$

Kapitel 9

Elementare Techniken

9.1 Beispiele

Beispiel 1

$\varphi(x,y) := \begin{cases} y \text{ falls } \varphi_x(x) \text{ terminiert } (\downarrow) \\ \bot \text{ sonst} \end{cases}$
ist berechenbar, durch $\Phi(x_1, x_1)$; $x_1 := x_2$.

Beispiel 2

$\varphi(x,y) = \begin{cases} y \text{ falls } \varphi_x(x) \ \downarrow \\ 0 \text{ sonst} \end{cases}$
ist unberechenbar, weil $\varphi(x,1) = \texttt{halt}(x)$.

Beispiel 3

Sei $\varphi_i \leq \varphi_j$, d.h.

$$\varphi_i(x) \downarrow \quad \Rightarrow \quad \varphi_i(x) = \varphi_j(x).$$

Dann ist

$$\varphi(x,y) = \begin{cases} \varphi_j(y) & \text{falls } \varphi_x(x) \downarrow \\ \varphi_i(j) & \text{sonst} \end{cases}$$

berechenbar:

Berechne $\varphi_i(y)$ und $\varphi_x(x)$ gleichzeitig
Terminiert $\varphi_x(x)$, dann höre auf und berechne $\varphi_j(y)$
Terminiert $\varphi_i(y)$, dann höre auf.

Beispiel 4

$\varphi(x,y,z) = \begin{cases} 1 \text{ falls } P_x \text{ bei Eingabe von } y \text{ in } z \text{ Schritten terminiert} \\ 0 \text{ sonst} \end{cases}$
ist berechenbar.
Erweitere den Interpreter um einen Schrittzähler. Dann kann damit φ berechnet werden.

9.2 Das s-n-m-Theroem

Satz

Für alle $m, n \in \mathbb{N}$ gibt es eine berechenbare, totale Funktion $s : \mathbb{N}_0^{m+1} \to \mathbb{N}_0$, so daß

$$\varphi_i(a_1, \ldots, a_m, x_1, \ldots, x_n) = \varphi_{s(i,a_1,\ldots,a_m)}(x_1, \ldots, x_n)$$

für alle konstante Werte $i, a_1, \ldots, a_m \in \mathbb{N}_0$.

Beweis (Exemplarisch für $m = n = 1$)

$\varphi_{s(i,a)}(x_1)$ wird berechnet durch $x_2 := x_1$; $x_1 := a$; P_i. Weil der Index

$$s(i, a) = \text{“}x_2 := x_1;\ x_1 := a;\ P_i\text{”}$$

berechenbar ist, folgt die Behauptung.

Folgerung

Für alle $a_1, \ldots, a_m$ ist $\varphi(x_1, \ldots, x_n) = \varphi_i(a_1, \ldots, a_m, x_1, \ldots, x_n)$ berechenbar.

Beispiel 1

Ist $g(i, j) =$ ein Index von $\varphi_i \circ \varphi_j(x)$ berechenbar?
Ja, denn $\varphi(i, j, x) = \varphi_i \circ \varphi_j(x)$ wird durch das `while`-Programm

$$x_1 := \Phi(j, x_3);\ \Phi(i, x_1)$$

berechnet. Also existiert ein $k \in \mathbb{N}_0$, so daß $\varphi_k = \varphi$. Nach dem s-m-n-Theorem gibt es eine berechenbare Funktion s, so daß $\varphi_k(i, j, x) = \varphi_{s(k,i,j)}(x)$ $(= (\varphi_i \circ \varphi_j)(x))$. Wegen der Folgerung ist $g(i, j) = s(k, i, j)$ berechenbar, wie zu beweisen war.

Einfacher

Offenbar ist

$$g(i, j) = \text{“}P_j;\ P_i\text{”}$$

berechenbar.

Beispiel 2

Finde eine algorithmische Aufzählung der Funktionen

$$\varphi(x) = 1, \varphi(x) = x, \varphi(x) = x^2, \ldots$$

d.h. finde eine berechenbare, totale Funktion g mit

$$\varphi_{g(n)}(x) = x^n.$$

Es gibt ein $k \in \mathbb{N}_0$, so daß $\varphi_k(y, x) = x^y$. Nach dem s-m-n-Theorem gibt es eine berechenbare Funktion s mit $\varphi_k(y, x) = \varphi_{s(k,y)}(x)$.
Mit $g(y) = s(k, y)$ finden wir eine algorithmische Aufzählung.

9.3 Unentscheidbare Probleme

Satz 1

Es gibt totale Funktionen, die nicht berechenbar sind.

Beweis

Es gibt überabzählbar viele totale Funktionen, aber nur abzählbar viele berechenbare Funktionen.

Definition

$\vartheta_0, \vartheta_1, \vartheta_2, \ldots$ ist eine *effektive* (*algorithmische* oder *rekursive*) Aufzählung einer Teilmenge der berechenbaren Funktionen, falls es eine totale berechenbare Funktion g gibt mit $\vartheta_n = \varphi_{g(n)}$.

Satz 2

Die totalen, berechenbaren Funktionen sind nicht effektiv aufzählbar.

Beweis (Diagonalverfahren)

Sei $f_0, f_1, \ldots$ eine effektive Aufzählung von totalen berechenbaren Funktionen. Dann ist $f(i) = f_i(i) + 1$ eine berechenbare totale Funktion. Aber f kommt in der Aufzählung nicht vor, denn $\forall j : f_j(j) \neq f_j(j) + 1 = f(j)$.

Satz 3

$$\texttt{total}(i) := \begin{cases} 1 \text{ falls } \varphi_i \text{ total} \\ 0 \text{ sonst} \end{cases} \quad \text{ist nicht berechenbar.}$$

Beweis (indirekt)

Wäre `total` berechenbar, dann lieferte

$$g(i) = \begin{cases} i \text{ falls } \texttt{total}(i) = 1 \\ j \text{ sonst} \end{cases}, \text{ wobei } \varphi_j \text{ total},$$

eine effektive Aufzählung der totalen Funktionen (Widerspruch zu Satz 2).

Satz 4 (allgemeines Halteproblem)

$$h(i,j) = \begin{cases} 1 \text{ falls } \varphi_i(j) \downarrow \\ 0 \text{ sonst} \end{cases}$$

ist nicht berechenbar (da allgemeiner als das Halteproblem).

Beweis

$h(i,i) = \texttt{halt}(i)$

Satz 5

$$e(i) = \begin{cases} 1 \text{ falls } \varphi_i = \text{id} \\ 0 \text{ sonst} \end{cases} \quad \text{ist nicht berechenbar.}$$

Beweis

Konstruiere eine berechenbare Funktion g mit $e \circ g = \texttt{total}$, d.h. es soll sein:

$$\varphi_{g(i)}(x) = \mathrm{id}(x) \Leftrightarrow \varphi_i \text{ total.}$$

Weil

$$\varphi(i,x) = \varphi_{g(i)}(x) = \begin{cases} x & \text{falls } \varphi_i(x) \downarrow \\ \bot & \text{sonst} \end{cases}$$

berechenbar ist, gibt es ein k mit $\varphi_k = \varphi$ und nach dem s-m-n-Theorem eine berechenbare Funktion s, so daß $\varphi(i,x) = \varphi_{s(k,i)}(x)$. Die Restriktion $g(i) = s(k,i)$ gibt das Gewünschte. Also kann e nicht berechenbar sein.

Kapitel 10

Rekursive `while`-Programme

`while`-Programme mit rekursiven Aufrufen sind `while`-berechenbar.

10.1 Beispiel

```
procedure Fakultät
if x_1 = 0 then x_1 := 1
else x_2 := x_1; x_1 := x_1 − 1;
  Fakultät;
  x_1 := x_2 * x_1
end
```

Bedeutung

x_1 ist Ein- und Ausgabevariable der Prozedur. Ein Aufruf der Prozedur läßt alle Variablen außer x_1 unverändert.

Ziel

Ein `while`-Programm (ohne Rekursionen) für Fakultät.

Definition

Definiere eine *Programmumschreibefunktion* (PUF), $f : \mathbb{N}_0 \to \mathbb{N}_0$, durch

$f(k)$ = "`if` $x_1 = 0$ `then` $x_1 := 1$
`else` $x_2 := x_1;\ x_1 := x_1 - 1;$
P_k, wobei die Variablen x_i, $i \geq 2$, umbenannt werden in x_{i+1}
$x_1 := x_1 * x_2$
`end`"

Bemerkung

f ist berechenbar und total und für $x < m$ gilt $\varphi_{f^m(k)}(x) = x!$.

Folgerung

Fakultät wird durch folgendes `while`-Programm berechnet:

$$\begin{aligned} P \quad = \quad & x_2 := f^{x_1+1}(0); \\ & x_1 := \Phi(x_2, x_1) \end{aligned}$$

Bemerkung

Ist $P_k = P$, dann gilt

$$\varphi_k = \varphi_{f(k)},$$

aber $k \neq f(k)$.

10.2 Fixpunkte von Programmumschreibefunktionen

Sei P ein rekursives `while`-Programm und $P_{g(n)}$ das Programm, das aus P entsteht, wenn jeder rekursive Aufruf von P durch P_n ersetzt wird mit Umbenennung der Variablen, so daß keine Seiteneffekte auftreten.
g heißt *Programmumschreibefunktion* (PUF) zu P.
g ist berechenbar und total.
Zu g gehört eine Transformation

$$T \quad : \quad \varphi_n \mapsto \varphi_{g(n)}$$

Definition

$$\begin{aligned} \varphi_P(x) \quad := \quad & \begin{cases} (T^m(\bot))(x) & \text{falls } P \text{ bei einer Rekursionstiefe } < m \\ & \text{terminiert} \\ \bot & \text{sonst} \end{cases} \\ = \quad & \sup T^m(\bot) \end{aligned}$$

Folgerung

φ_P ist berechenbar, nämlich durch den folgenden Algorithmus

1. Führe 1. Schritt von $T(\bot)(x)$ durch.
2. Führe 1. und 2. Schritt von $T^2(\bot)$ durch.
3. Führe 1. und 2. und 3. Schritt von $T^3(\bot)$ durch.

   $\vdots$

Wenn ein $T^m(\bot)$ terminiert, stoppe und gib das Ergebnis aus.

Offenbar gilt: $T(\varphi_P) = \varphi_P$, d.h. φ_P ist Fixpunkt von T. Unser nächstes Ziel ist es, zu zeigen, daß φ_P der kleinste Fixpunkt von T ist.

Mit der Relation

$$\varphi \leq \zeta, \text{ falls } \varphi(i) = \zeta(i) \text{ oder } \varphi(i) \uparrow$$

ist $\mathcal{M} = \{\varphi \mid \varphi : \mathbb{N}_0 \to \mathbb{N}_0\}$ eine vollständig halbgeordnete Menge.

Lemma

T ist stetig.

Beweis

- T ist monoton $\quad\surd$
- Sei $\vartheta_1, \vartheta_2, \ldots$ eine Kette in $\mathcal{M}$. Zu zeigen ist dann:

$$T(\sup_i \vartheta_i) = \sup_i (\vartheta_i)$$

" $\geq$ ": Aus der Monotonie folgt $T(\sup_i \vartheta_i) \geq T(\vartheta_i)$ und daraus $T(\sup_i \vartheta_i) \geq \sup_i T(\vartheta_i)$.

" $\leq$ ": Offenbar existiert für alle $k \in \mathbb{N}_0$ ein j:

$$T(\sup_i \vartheta_i)(k) = T(\vartheta_j)(k) \leq \sup_i T(\vartheta_i)(k)$$

Folgerung

$\varphi_P = \sup T^m(\bot)$ ist der kleinste Fixpunkt von T.

Beweis

Fixpunktsatz von Kleene (siehe Abschnitt 5.2) und Lemma.

10.3 Beispiele

Beispiel

T sei die Transformation zum rekursiven Programm Fakultät aus Abschnitt 10.1. Dann gilt nach 10.2

$$\sup T^m(\bot) = f \text{ mit } f(x) = x!$$

Das heißt $f(x)$ ist der kleinste Fixpunkt und der einzige, weil f total ist.

Beispiel

```
procedure gerade
if x1 = 0 then x1 = 1
elseif x1 = 1 then x1 := 3; gerade;
  else x1 := x1 − 2; gerade;
```

Also ist
$\varphi_{\text{gerade}}(x) = \begin{cases} 1 & \text{falls } x \text{ gerade} \\ \perp & \text{sonst} \end{cases}$.
Sei T die zugehörige Transformation. Dann gilt

$$T(\varphi)(x) = \begin{cases} 1 & \text{falls } x = 0 \\ \varphi(3) & \text{falls } x = 1 \\ \varphi(x-2) & \text{sonst} \end{cases}$$

Das heißt φ ist Fixpunkt von T, falls $\varphi(x) = \begin{cases} 1 & \text{falls } x \text{ gerade} \\ \varphi(3) & \text{falls } x \text{ ungerade} \end{cases}$. Also gibt es unendlich viele Fixpunkte.

Beispiel

`procedure` alle
alle

Also ist $\varphi_{\text{alle}} = \perp$. Sei T die zugehörige Transformation. Dann gilt

$$T(\varphi_i) = \varphi_i,$$

d.h. alle φ_i sind Fixpunkte von T.

Beispiel

`procedure` leer
leer; $x_1 := x_1 + 1$

Also ist $\varphi_{\text{leer}} = \perp$. Sei T die zugehörige Transformation. Dann gilt

$$T(\varphi)(x) = \begin{cases} \varphi(x)+1 & \text{falls } \varphi(x) \downarrow \\ \perp & \text{sonst} \end{cases},$$

d.h. kein $\varphi(x)$ außer $\perp$ kann Fixpunkt von T sein.

10.4 Der Rekursionssatz

Rekursionssatz (Kleene, 1935)

Für alle berechenbaren totalen Funktionen f gibt es ein n mit

$$\varphi_n = \varphi_{f(n)}.$$

Beweis

Die Funktion $\varphi_{\varphi_m(m)}(x) = \Phi(\Phi(m,m),x)$ ist eine berechenbare Funktion φ_n. Nach dem s-m-n-Theorem gibt es eine berechenbare totale Funktion g mit $\varphi_{g(m)}(x) = \varphi_n(m,x)$. Insbesondere für m mit

$$\varphi_m = f \circ g$$

folgt

$$\varphi_{g(m)}(x) = \varphi_{f(g(m))}(x).$$

D.h. die Behauptung folgt für $n = g(m)$.

Beispiel 1

Sei f die Programmumschreibefunktion, die + in − umwandelt. Dann gibt es ein n mit

$$\varphi_n = \varphi_{f(n)}.$$

Beispiel 2

Es existiert ein $n \in \mathbb{N}_0 : \varphi_n(x) = n$, denn für

$$f(n) = \text{“}x_1 := n\text{”}$$

folgt

$$\varphi_{f(n)}(x) = n$$

$$\text{Rekursions–satz} \Rightarrow \quad \exists n : \varphi_n(x) = \varphi_{f(n)}(x) = n$$

Beispiel 3

Es existiert ein n für alle $i : \varphi_n(i) = \varphi_i(n)$, denn für

$$f(n) = \text{“}x_1 := \Phi(x_1, n)\text{”}$$

gilt

$$\varphi_{f(n)}(x) = \varphi_x(n)$$

woraus mit Hilfe des Rekursionssatzes folgt, daß es ein n gibt mit

$$\varphi_n(x) = \varphi_{f(n)}(x) = \varphi_x(n).$$

Beispiel 4

$\exists n : \varphi_n(x) \downarrow \quad \Leftrightarrow \quad x = n$, denn für

$$f(n) = \text{“}\texttt{while}\, x_1 \neq n\, \texttt{do}\, x_1 := x_1\, \texttt{end}\text{”}$$

gilt

$$\varphi_{f(n)}(x) = \begin{cases} n & \text{falls } x = n \\ \perp & \text{sonst} \end{cases}$$

woraus mit Hilfe des Rekursionssatzes folgt, daß es ein n gibt mit

$$\varphi_n(x) = \varphi_{f(n)}(x) = \begin{cases} n & \text{falls } x = n \\ \perp & \text{sonst} \end{cases}$$

Satz

Für alle berechenbaren totalen Funktionen f gibt es unendlich viele n mit

$$\varphi_n = \varphi_{f(n)}.$$

Beweis

Sei $k \in \mathbb{N}_0$ beliebig und sei

$$\varphi_k \neq \varphi_0, \ldots, \varphi_k.$$

Sei

$$g(n) = \begin{cases} f(n) & \text{falls } n > k \\ l & \text{falls } n \leq k \end{cases}.$$

Dies ist eine berechenbare totale Funktion. Zum Beispiel wird $g(n)$ berechnet durch

$$\begin{array}{l} \texttt{if } x_1 \leq k \texttt{ then } x_1 := l \\ \quad \texttt{else } x_1 := f(x_1) \end{array}$$

Nach dem Rekursionssatz gibt es ein n mit

$$\varphi_n = \varphi_{g(n)}.$$

Weil für alle $m \leq k$ gilt: $g(m) = l$, ist also

$$\varphi_{g(m)} = \varphi_l \neq \varphi_0, \ldots, \varphi_k.$$

Also ist $n > k$. Daraus folgt

$$\varphi_n = \varphi_{g(n)} = \varphi_{f(n)},$$

d.h. es gibt beliebig viele/große n:

$$\varphi_n = \varphi_{f(n)}.$$

10.5 Zusammenfassung

$\mathcal{P} := \{P_i \mid i \in \mathbb{N}_0\}$	$\mathcal{F} := \{\varphi_i \mid i \in \mathbb{N}_0\}$
$R = PRQ$ ein rekursives `while`-Programm	φ_R
$G: \mathcal{P} \to \mathcal{P}$ $P_n \mapsto PP_nQ$ zugehörige PUF $g: \mathbb{N}_0 \to \mathbb{N}_0$ $n \mapsto$ "PP_nQ"	$T: \mathcal{F} \to \mathcal{F}$ $\varphi_n \mapsto \varphi_{g(n)}$
$G^2 : P_n \mapsto PPP_nQQ$ $\vdots$ $G^\infty : P_n \mapsto \underbrace{PP\ldots\ldots QQ}_{R \notin \mathcal{P}}$	$T^2 : \varphi_n \mapsto \varphi_{g^2(n)}$ $\vdots$ $T^\infty : \varphi_n \to \varphi_R = \sup_k T^n(\varphi_n)$ $\varphi_R \in \mathcal{F}$
G^∞ ist konstant	T^∞ ist konstant

Ohne Entsprechung in $\mathcal{P}$

$\varphi_i \leq \varphi_j \Rightarrow T(\varphi_i) \leq T(\varphi_j)$.
Insbesondere $\perp \leq T(\perp) \leq T^2(\perp) \leq \ldots$
Also

$$\varphi_R = \sup_k T^k(\varphi_n) = \sup_k T^k(\perp).$$

Aus dem Fixpunktsatz folgt

$$\varphi_R = \sup_k T^k(\perp) \text{ ist kleinster Fixpunkt.}$$

Rekursionssatz

f beliebig und total $\Rightarrow \exists n : \varphi_n = \varphi_{f(n)}$

Kapitel 11

Entscheid- und Aufzählbarkeit

entscheidbar $\Rightarrow$ aufzählbar $\Rightarrow$ abzählbar unendlich

Anwendungen

- Gödels Unvollständigkeitssatz
- Hilberts 10. Problem (David Hilbert, Göttinger Mathematiker. 10. Problem: „Sind polynomiale Gleichungen über $\mathbb{Z}$ (diophantische Gleichungen) in $\mathbb{Z}$ lösbar?“)

11.1 Rekursive und rekursiv aufzählbare Mengen

Sei $\mathcal{A}$ eine Menge. Dann ist die *charakteristische Funktion* von $\mathcal{A}$ gegeben durch

$$\chi_{\mathcal{A}}(x) = \begin{cases} 1 & \text{falls } x \in \mathcal{A} \\ 0 & \text{sonst.} \end{cases}$$

Definition

$\mathcal{A} \subset \mathbb{N}_0^k$ heißt *berechenbar*, *lösbar*, *entscheidbar* oder *rekursiv*, falls $\chi_{\mathcal{A}}$ berechenbar ist.

Beispiele

1. endliche Mengen sind rekursiv.
2. $\{p \mid p \text{ prim}\}$ ist rekursiv.
3. $\{i \mid \varphi_i \text{ total}\}$ ist nicht rekursiv, siehe Abschnitt 9.3.
4. $\{(i,j) \mid \varphi_i(j) \downarrow\}$ ist nicht rekursiv, siehe Abschnitt 9.3.
5. Jedes Wort über einem Alphabet $\mathcal{A} = \{a_2, \ldots a_q\}$ ist q-näre Darstellung einer natürlichen Zahl. Daraus folgt nach 4.8, daß Typ-1-Sprachen über $\mathcal{A}$ rekursiv sind.

6. Jedes $n \in \mathbb{N}_0$ repräsentiert eine Turingmaschine bzw. eine Typ-0-Sprache $\mathcal{L}_n$. Daraus folgt nach 4.10, daß $\{(i,j) \mid q\text{-näre Darstellung von } i \in \mathcal{L}_j\}$ nicht rekursiv ist.

Definition

Eine Menge $\mathcal{A} \subset \mathbb{N}_0^k$ heißt *rekursiv aufzählbar*, kurz *r.a.*, falls

- $\mathcal{A} = \emptyset$ oder
- $\mathcal{A} = \text{Bild}(f)$, f berechenbar und total.

f heißt *Aufzählfunktion* und $f(0), f(1), \ldots$ *effektive Aufzählung* von $\mathcal{A}$.

Beispiele

1. $\{p \mid p \text{ prim}\}$ ist rekursiv aufzählbar.
2. $\{i \mid \varphi_i \text{ ist total}\}$ ist nicht rekursiv aufzählbar.

Satz 1

Jede rekursive Menge ist auch rekursiv aufzählbar.

Beweis

Die leere Menge ist nach Definition rekursiv aufzählbar. Sei also $\mathcal{A}$ eine rekursive Menge und a ein Element von $\mathcal{A}$. Dann wird $\mathcal{A}$ durch folgendes `while`-Programm aufgezählt:

- $a \in \mathcal{A}$:

$y := x_1;$
`if` $\chi_{\mathcal{A}}(x_1) = 1$ `then` $x_1 := y$ `else` $x_1 := a$

Satz 2

$\mathcal{K} := \{i \mid \varphi_i(i) \downarrow\}$ ist rekursiv aufzählbar.

Bemerkung

Also gilt die Umkehr von Satz 1 nicht.

Beweis

Sei $f : \mathbb{N}_0 \to \mathbb{N}_0^2$ eine berechenbare Bijektion. Dann wird $\mathcal{K}$ aufgezählt durch

$(i,j) := f(x_1)$
`while` $\Phi(i,i)$ terminiert nicht in j Schritten
`do` $x_1 := x_1 + 1;\ (i,j) := f(x_1)$ `end`;
$x_1 := i$

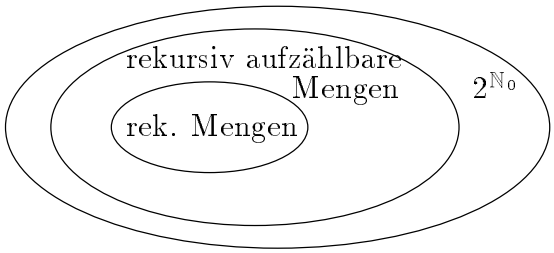


Abbildung 11.1: Mengenschaubild

Zusammenfassung

siehe Abbildung 11.1.
fdsfd dsfj sdfsd fsdfsdlf sdf sd fls sdf dsfjsdfl sfksdf sdjf sdf slfjds lsd fsd lfdj fds fjsldfj sdj fds flsd fsdj fsdjfdsf.d sd klfsdlfsdfj fj lsd. sd fdlfjd djf sdf djflsd s fsldfjsdfls f fjslfjsdfjs fs.

11.2 Zusammenhänge zwischen rekursiven und rekursiv aufzählbaren Mengen

Satz

$\mathcal{A}$ rekursiv $\Leftrightarrow$ $\mathcal{A}$ und $\overline{\mathcal{A}}$ rekursiv aufzählbar.

Beweis

„$\Rightarrow$“

$$\begin{aligned} \mathcal{A} \text{ rekursiv} &\Leftrightarrow \chi_{\mathcal{A}} \text{ berechenbar} \\ &\Leftrightarrow \chi_{\overline{\mathcal{A}}} = 1 - \chi_{\mathcal{A}} \text{ berechenbar} \\ &\Leftrightarrow \overline{\mathcal{A}} \text{ rekursiv} \\ &\Rightarrow \overline{\mathcal{A}} \text{ rekursiv aufzählbar} \end{aligned}$$

„$\Leftarrow$“

- $\mathcal{A} = \emptyset$ oder $\overline{\mathcal{A}} = \emptyset$ $\quad\surd$
- Seien f und $\overline{f}$ Aufzählfunktionen von $\mathcal{A}$ und $\overline{\mathcal{A}}$. Dann wird $\chi_{\mathcal{A}}$ berechnet durch

$y := 0;$
while $x_1 \neq f(y)$ **and** $x_1 \neq \overline{f}(y)$
do $y := y + 1$ **end**;
if $x_1 = f(y)$ **then** $x_1 := 1$
else $x_1 := 0$

Beispiel (vgl. Abschnitt 11.1)

$\overline{\mathcal{K}} := \{i \mid \varphi_i(i) \uparrow\}$ ist nicht rekursiv aufzählbar, weil $\mathcal{K}$ nicht rekursiv, aber rekursiv aufzählbar ist.

Satz

Jede unendliche rekursiv aufzählbare Menge hat unendlich große rekursive Teilmengen.

Beweis

Sei f berechenbar und total und $f(\mathbb{N}_0)$ unendlich. Dann ist

$$\mathcal{M} := \{f(i) \mid f(i) > f(0), \ldots, f(i-1) \text{ und } i \in \mathbb{N}_0\}$$

eine unendliche Teilmenge von $f(\mathbb{N}_0)$.
$\mathcal{M}$ ist rekursiv, weil $\chi_{\mathcal{M}}$ berechenbar ist, zum Beispiel durch

$y := 0$
`while` $f(y) < x_1$ `do` $y := y + 1$ `end`
`if` $f(y) = x_1$ `then` $x_1 := 1$
`else` $x_1 := 0$

11.3 Andere Definitionen von rekursiv aufzählbaren Mengen

Definition

Sei $\varphi : \mathbb{N}_0^k \to \mathbb{N}_0^m$. Dann ist

$$D(\varphi) = \{x \in \mathbb{N}_0^k \mid \varphi(x) \downarrow\} = \varphi^-(\mathbb{N}_0^m)$$

der *Definitionsbereich* und

$$\text{Bild}(\varphi) = \varphi(\mathbb{N}_0^k)$$

der *Bildbereich* von φ.

Satz

Folgende Aussagen sind äquivalent

a) $\mathcal{A}$ ist rekursiv aufzählbar

b) $\mathcal{A} = D(\varphi)$, wobei φ berechenbar

c) $\mathcal{A} = \text{Bild}(\psi)$, wobei ψ berechenbar

Bemerkung

φ, ψ müssen nicht total sein.

Beweis

a)⇒b)

- Falls $\mathcal{A} = \emptyset$, gilt $\mathcal{A} = D(\bot)$.
- Falls $\mathcal{A} = f(\mathbb{N}_0^k)$, wobei f berechenbar und total, setze

$$\varphi(x) := \begin{cases} x & \text{falls } x \in \mathcal{A} \\ \bot & \text{falls } x \notin \mathcal{A} \end{cases} .$$

  Dann gilt $\mathcal{A} = D(\varphi)$ und φ ist berechenbar durch

$$y := 0;$$
$$\texttt{while} \quad x_1 \neq f(y) \quad \texttt{do} \quad y := y + 1 \quad \texttt{end}$$

b)⇒c)
Sei

$$\psi(x) = \begin{cases} x & \text{falls } x \in \mathcal{A} \\ \bot & \text{sonst.} \end{cases}$$

Dann gilt $\mathcal{A} = \text{Bild}(\psi)$ und ψ wird berechnet durch

$$y := x_1;\ \varphi(x_1);\ x_1 := y$$

c)⇒a)

- Falls $\mathcal{A} = \emptyset$, ist $\mathcal{A}$ nach Definition rekursiv aufzählbar.
- Falls es ein $a \in \mathcal{A}$ gibt, sei $f : \mathbb{N}_0 \to \mathbb{N}_0^2$ eine Bijektion und

$$g(x) := \begin{cases} \psi(i) & \text{falls } \psi(i) \text{ in } j \text{ Schritten berechenbar,} \\ & \text{wobei } (i,j) = f(x_1) \\ a & \text{sonst.} \end{cases}$$

  Dann ist g berechenbar und total und

$$\text{Bild}(g) = \mathcal{A}.$$

11.4 Reduktionen

Definition

$\mathcal{A}$ heißt *auf* $\mathcal{B}$ *reduzierbar*, kurz $\mathcal{A} \leq \mathcal{B}$, falls es eine berechenbare totale Funktion f gibt mit

$$f(\mathcal{A}) \subset \mathcal{B} \text{ und } f(\overline{\mathcal{A}}) \subset \overline{\mathcal{B}}$$

bzw.

$$\mathcal{A} = f^{-}(\mathcal{B}).$$

Lemma

Falls $\mathcal{A} \leq \mathcal{B}$, gilt:

a) $\mathcal{B}$ rekursiv $\Rightarrow$ $\mathcal{A}$ rekursiv

b) $\mathcal{A}$ nicht rekursiv $\Rightarrow$ $\mathcal{B}$ nicht rekursiv

c) $\mathcal{B}$ rekursiv aufzählbar $\Rightarrow$ $\mathcal{A}$ rekursiv aufzählbar

d) $\mathcal{A}$ nicht rekursiv aufzählbar $\Rightarrow$ $\mathcal{B}$ nicht rekursiv aufzählbar

Beweis

a) $\chi_{\mathcal{A}} = \chi_{\mathcal{B}} \circ f$

c) $\mathcal{B} = D(\varphi) \;\Rightarrow\; \mathcal{A} = D(\varphi \circ f)$

b) und d) sind die Kontrapositionen von a) und d)

Folgerung

$\overline{\mathcal{K}} \leq \mathcal{A} \Rightarrow \mathcal{A}$ ist nicht rekursiv aufzählbar.

Satz 1

$\mathcal{A}$ rekursiv aufzählbar $\Rightarrow \mathcal{A} \leq \mathcal{K} := \{i \mid \varphi_i(i)\downarrow\}$.

Beweis

Sei f so, daß

$$\varphi_{f(i)} = \begin{cases} 1 & \text{falls } i \in \mathcal{A} \\ \perp & \text{sonst.} \end{cases}$$

Dann gilt

$$i \in \mathcal{A} \Leftrightarrow f(i) \in \mathcal{K}.$$

Ist g die Aufzählfunktion von $\mathcal{A}$, setze z.B.

$$\begin{aligned} f(i) = \quad & \text{“}y := 0; \quad \texttt{while } i \neq g(y) \\ & \qquad\quad \texttt{do } y := y + 1 \\ & \qquad\quad \texttt{end;} \\ & \qquad\quad x_1 := 1\text{”} \end{aligned}$$

Also ist $\mathcal{A} \leq \mathcal{K}$.

Bemerkung

- Wegen dieser Eigenschaft heißt $\mathcal{K}$ eine *vollständige* rekursiv aufzählbare Menge.
- Es gibt rekursiv aufzählbare Mengen, die nicht vollständig, aber auch nicht rekursiv sind.

11.5 Der Satz von Rice

Ein universelles Hilfsmittel, um Unentscheidbarkeit zu zeigen.

Definition

$\mathcal{M} \subset \mathbb{N}_0$ *respektiert Funktionen*, falls

$$\forall i \in \mathcal{M} \quad \forall j \in \mathbb{N}_0 : \varphi_i = \varphi_j \Rightarrow j \in \mathcal{M}.$$

Satz (Rice)

$\emptyset$ und $\mathbb{N}_0$ sind die einzigen rekursiven Mengen, die Funktionen respektieren.

Beweis

- $\emptyset$ und $\mathbb{N}_0$ sind rekursiv und respektieren Funktionen.
- $\mathcal{M}$ respektiere Funktionen und $\mathcal{M} \neq \emptyset, \mathbb{N}_0$

  a) Sei $a \in \mathcal{M}$ und alle Indexe von $\perp$ in $\overline{\mathcal{M}}$
  Zeige: $\mathcal{M}$ ist nicht rekursiv bzw. $\mathcal{K} \leq \mathcal{M}$
  D.h. wir suchen eine berechenbare totale Funktion f mit

  $$\mathcal{K} = f^{-}\mathcal{M},$$

  z.B. f gegeben durch

  $$\varphi_{f(i)}(x) = \begin{cases} \varphi_a(x) & \text{falls } \varphi_i(i) \downarrow \\ \perp & \text{sonst} \end{cases}$$

  bzw.

  $$f(i) = \quad \text{“}x_1 := \Phi(i,i);\ x_1 := \Phi(a,x)\text{”}$$

  Also ist $\mathcal{M}$ nicht rekursiv.

  b) Sind die Indexe von $\perp$ in $\mathcal{M}$, zeigt man wie in a), daß $\overline{\mathcal{M}}$ nicht rekursiv ist. Daraus folgt, daß $\mathcal{M}$ ebenfalls nicht rekursiv ist.

Beispiele

für Indexmengen, die Funktionen respektieren:

1. $\{i \mid \varphi_i \text{ ist total}\}$
2. $\{i \mid \varphi_i = \varphi_j\}$, $j \in \mathbb{N}_0$
3. $\{i \mid a \in D(\varphi_i)\}$, $a \in \mathbb{N}_0$
4. $\{i \mid D(\varphi_i) \text{ ist endlich}\}$
5. $\{i \mid a \in \text{Bild}(\varphi_i)\}$, $a \in \mathbb{N}_0$
6. $\{i \mid \text{Bild}(\varphi_i) \text{ ist unendlich}\}$

7. $\{i \mid \varphi_i$ ist injektiv$\}$
8. $\{i \mid \varphi_i$ hat ein bestimmtes Ein/Ausgabeverhalten$\}$

Daraus folgt mit Hilfe des Riceschen Satzes

1. total ist nicht berechenbar
2. Äquivalenzproblem unentscheidbar

   $\vdots$

8. Unentscheidbar, ob ein Programm ein bestimmtes Ein/Ausgabeverhalten hat.

Definition

$\mathcal{M} \subset \mathbb{N}_0^k$ *respektiert Funktionen*, falls

$$\forall \mathbf{i} = (i_1, \ldots, i_k) \in \mathcal{M}\, \forall \mathbf{j} = (j_1, \ldots, j_k) \in \mathbb{N}_0^k : \\ \varphi_{\mathbf{i}} := (\varphi_{i_1}, \ldots, \varphi_{i_k}) = \varphi_{\mathbf{j}} := (\varphi_{j_1}, \ldots, \varphi_{j_k}) \quad \Leftrightarrow \quad \mathbf{j} \in \mathcal{M}$$

Satz (Ricescher Satz, Vektorfassung)

$\emptyset$ und $\mathbb{N}_0^k$ sind die einzigen rekursiven Teilmengen von $\mathbb{N}_0^k$, die Funktionen respektieren.

Beweis

analog

Beispiele

für Mengen, die Funktionen respektieren:

1. $\{(i,j) \mid \varphi_i = \varphi_j\}$
2. $\{(i,j) \mid \varphi_i(a) < \varphi_j(a)$ falls definiert$\}$
3. $\{(i,j,k) \mid \varphi_i = \varphi_j \circ \varphi_k\}$

11.6 Erweiterungen des Riceschen Satzes

Satz 1

$\mathcal{M}$ respektiere Funktionen und $\exists a \in \mathcal{M}\, \exists b \in \overline{\mathcal{M}} : \varphi_a \leq \varphi_b$. Dann ist $\mathcal{M}$ nicht rekursiv aufzählbar.

Beweis

Nach 11.4 genügt es zu zeigen, daß $\overline{\mathcal{K}} \leq \mathcal{M}$. D.h. wir suchen eine berechenbare totale Funktion f mit

$$f(\overline{\mathcal{K}}) \subset \mathcal{M} \text{ und } f(\mathcal{K}) \subset \overline{\mathcal{M}},$$

z.B. f gegeben durch

$$\varphi_{f(i)}(x) = \begin{cases} \varphi_a(x) & \text{falls } \varphi_i(i) \uparrow \\ \varphi_b(x) & \text{falls } \varphi_i(i) \downarrow \end{cases}$$

bzw.

$$f(i) = \text{“Programm aus 9.1 Beispiel 3”}$$

Beispiele

auf die Satz 1 anwendbar ist

1. $\{i \mid \varphi_i \text{ nicht total}\}$. Setze z.B. $\varphi_2 = \perp, \varphi_b = 1$.
2. $\{i \mid \varphi_i \neq \varphi_j\}$, wobei $\varphi_j \neq \perp$. Setze z.B. $\varphi_a = \perp$, $\varphi_b = \varphi_j$.
3. $\{i \mid \varphi_i = \varphi_j\}$, wobei $\varphi_j < \varphi_k$ für ein $k \in \mathbb{N}_0$. Setze z.B. $\varphi_a = \varphi_j$, $\varphi_b = \varphi_k$.
4. $\{i \mid D(\varphi_i) \text{ ist endlich}\}$. Setze z.B. $\varphi_a = \perp$, $\varphi_b = 1$.
5. $\{i \mid \varphi_i \text{ ist injektiv und nicht total}\}$. Setze z.B. $\varphi_a = \perp$, $\varphi_b = x$.
6. $\{i \mid c \notin \text{Bild } \varphi_i\}$. Setze z.B. $\varphi_a = \perp$ und $\varphi_b = c$.

Allgemein: Die `while`-Programme mit einem bestimmten beliebigen Ein/Ausgabeverhalten, das von einem φ_i, aber nicht von einer Fortsetzung von φ_j erfüllt wird, sind nicht rekursiv aufzählbar.

Satz 2

$\mathcal{M}$ respektiere Funktionen und

$$\exists a \in \mathcal{M} \quad \forall i \in \mathbb{N}_0 : \varphi_i \leq \varphi_a \text{ und } D(\varphi_i) \text{ ist endlich } \Rightarrow i \in \overline{\mathcal{M}}$$

Dann ist $\mathcal{M}$ nicht rekursiv aufzählbar.

Beweis

Wie in Satz 1 suchen wir eine berechenbare totale Funktion f mit

$$f(\overline{\mathcal{K}}) \subset \mathcal{M} \text{ und } f(\mathcal{K}) \subset \overline{\mathcal{M}},$$

z.B. f gegeben durch

$$\varphi_{f(i)}(x) = \begin{cases} \varphi_a(x) & \text{falls } \Phi(i,i) \uparrow \text{ in } x \text{ Schritten} \\ \perp & \text{falls } \Phi(i,i) \downarrow \text{ in } x \text{ Schritten.} \end{cases}$$

Dann ist

$$\varphi_{f(i)} = \varphi_a \text{ falls } \varphi(i,i) \uparrow$$

oder

$$D(\varphi_{f(i)}) \subset \{0, 1, \ldots, n-1\} \text{ falls } \Phi(i,i) \text{ in } n \text{ Schritten terminiert}$$

und $\varphi_{f(i)} \leq \varphi_a$.

Beispiele

auf die Satz 2 anwendbar ist

1. $\{i \mid \varphi_i \text{ ist total}\}$
2. $\{i \mid \varphi_i = f\}$, wobei f berechenbar und total
3. $\{i \mid D(\varphi_i) \text{ ist unendlich}\}$
4. $\{i \mid \varphi_i(\mathbb{N}_0) \text{ ist unendlich}\}$

$\vdots$

Allgemein

Die `while`-Programme mit einem bestimmten beliebigen unendlichen Ein- oder Ausgabeverhalten sind nicht rekursiv aufzählbar.

Kapitel 12

Gödels Satz

Gödel (1931): Man kann nicht alle wahren arithmetischen Formeln beweisen.

12.1 Arithmetische Formeln

Terme

- Jede Zahl $n \in \mathbb{N}_0$ ist ein Term.
- Variable x_i, $i \in \mathbb{N}_0$, sind Terme.
- u, v Terme $\Rightarrow (u + v)$ und $(u \cdot v)$ Terme.

Formeln

- Sind u, v Terme, dann ist $(u = v)$ eine Formel.
- Sind F und G Formeln, dann auch $\neg F, (F \wedge G), (F \vee G)$.
- Ist x eine Variable und F eine Formel, dann sind auch $\exists x F$ und $\forall x F$ Formeln.

Beispiele

- $x < y$ entspricht $\exists z(((x + z) + 1) = y)$
- $x > y$, $x \leq y$, $x \geq y$ analog
- $\exists x < k\, F$ entspricht $\exists x((x < k) \wedge F)$

Die Variablen, die direkt hinter einem Quantor stehen, heißen *gebunden*, die anderen *frei*.
$F(x_1, \ldots, x_n)$ bezeichnet eine Formel, in der genau die Variablen $x_1, \ldots, x_n$ frei sind. Werden $x_1, \ldots, x_n$ durch $m_1, \ldots, m_n \in \mathbb{N}_0$ ersetzt, erhält man die Formel $f(m_1, \ldots, m_n)$.
Eine Formel ist wahr oder falsch wie gewohnt.

Definition

$\varphi : \mathbb{N}_0^k \to \mathbb{N}_0^l$ heißt *arithmetisch repräsentierbar*, falls es eine Formel $F(x_1, \ldots, x_k, z_1, \ldots, z_l)$ gibt, so daß $f(m_1, \ldots, m_k, n_1, \ldots, n_l)$ genau dann wahr ist, wenn

$$\varphi(m_1, \ldots, m_k) = (n_1, \ldots, n_l)$$

für alle $m_1, \ldots, m_k, n_1, \ldots, n_l \in \mathbb{N}_0$.

Beispiel

- $f(x, y) = x + y$ wird repräsentiert durch
$$(z = (x + y))$$
- $f(x, y) = x \bmod y$ wird repräsentiert durch
$$\exists k\, (x = ky + z \wedge z < y)$$
- $f(x, y) = x \operatorname{div} y$ wird repräsentiert durch
$$\exists r < y\, (x = zy + r)$$
- $\varphi(x) = \perp$ wird repräsentiert durch
$$(x = z \wedge 1 = 2)$$

12.2 Lemma

Lemma

$\forall (n_0, \ldots, n_k) \in \mathbb{N}_0^{k+1} \quad \exists a, b \in \mathbb{N}_0 \quad \forall i = 0, \ldots, k : n_i = a \bmod (1 + ib)$.

Beweis

Sei $s := \max\{k, n_0, \ldots, n_k\}$ und $b := s!$

1. Die $b_i = (1 + ib)$, $i = 0, \ldots, k$, sind teilerfremd, denn
$$\begin{aligned} & p \text{ prim, } p \mid b_i \text{ und } p \mid b_j \\ \Rightarrow\ & p \mid (b_j - b_i) \\ \Rightarrow\ & p \mid (j - i)b \\ \Rightarrow\ & p \mid b \text{ weil } j - i \leq k \text{ und } k! \mid b \\ \Rightarrow\ & p \mid ib \text{ und } p \mid (1 + ib) \\ \Rightarrow\ & p = 1 \end{aligned}$$
2. Sei $a < a'$ und $\forall i \leq k : a \bmod b_i = a' \bmod b_i$. Dann gilt
$$\begin{aligned} & \forall i \leq k : (a' - a) \bmod b_i = 0 \\ \Rightarrow\ & \forall i \leq k : b_i \mid (a' - a) \\ \Rightarrow\ & b_0 \ldots b_k \mid a' - a \\ \Rightarrow\ & a' - a \geq b_0 \ldots b_k \\ \Rightarrow\ & a' \geq b_0 \ldots b_k \end{aligned}$$

3. Also gilt

$$\{(a \bmod b_0, \ldots, a \bmod b_k) \mid 0 \leq a < b_0 \ldots b_k\}$$
$$= \{(m_0, \ldots, m_k) \mid m_i < b_i\}.$$

12.3 Satz

Satz

Sei P ein `while`-Programm mit den Variablen $x_1, \ldots, x_q$. Dann ist $\varphi_p^{q,q} : \mathbb{N}_0^q \to \mathbb{N}_0^q$ arithmetisch repräsentierbar.

Beweis

durch Induktion

- $x_i := x_j + c$ wird repräsentiert durch

$$F(x_1, \ldots, x_q, y_1, \ldots, y_q) = (y_i = x_j + c \wedge \forall l \neq i (y_l = x_l))$$

- $x_i := x_j - c$ analog
- $Q; R$ wird repräsentiert durch

$$\exists z_1, \ldots, z_q \quad (F_Q(x_1, \ldots, x_q, z_1, \ldots, z_q) \wedge F_R(z_1, \ldots, z_q, y_1, \ldots, y_q))$$

- `while` $x_i \neq x_j$ `do` Q `end` wird repräsentiert durch

$\exists k$
  {Anzahl der Schleifendurchgänge}
$\exists a_1 \exists b_1 \ldots \exists a_q \exists b_q$
  {Kodierungen für die Variablenbelegungen nach 11.2.
  D.h. nach l Schleifendurchgängen
  $x_s = n_s^l := a_s \mod (1 + l \cdot b_s)$}
$(x_1 = n_1^0 \wedge \cdots \wedge x_q = a_q^0 \wedge$ {Belegung vor der Schleife}
$y_1 = n_1^k \wedge \cdots \wedge y_q = n_q^k \wedge$ {Belegung nach der Schleife}
$\forall l < k : (F_Q(n_1^l, \ldots, n_q^l, n_1^{l+1}, \ldots, n_q^{l+1}) \wedge n_i^l \neq n_j^l)$
$\wedge n_i^k = n_j^k)$.

Folgerung

Jede berechenbare Funktion $\varphi_i : \mathbb{N}_0^k \to \mathbb{N}_0^m$ ist arithmetisch repräsentierbar.

12.4 Gödels Unvollständigkeitssatz

$W := \{\text{“}F\text{”} \mid F \text{ ist eine wahre Formel}\}$,
wobei “F” eine numerische Kodierung von F ist.

Satz

W ist nicht rekursiv.

Beweis

$$\varphi(x) := \begin{cases} 1 & \text{falls } \varphi_x(x) \downarrow \\ \bot & \text{sonst} \end{cases}$$

ist berechenbar. Also ist φ durch eine Formel $F(x, y)$ repräsentiert. D.h.

$$\varphi_x(x) \downarrow \quad \Leftrightarrow \quad \varphi(x) = 1 \quad \Leftrightarrow \quad F(x, 1) \in W.$$

Wäre W rekursiv, dann könnte halt(x) berechnet werden. Dies ist unmöglich.

Folgerung

W ist nicht rekursiv aufzählbar.

Beweis

Wäre W aufzählbar, dann ist $F \in W$ entscheidbar, indem W bis F oder $\neg F$ aufgezählt wird.

Definition

Eine *Beweissystem* für W besteht aus einer rekursiven Menge B von Beweisen (d.h. Formelfolgen) und einer berechenbaren totalen Funktion $f : B \to W$. Ein Beweissystem ist *vollständig*, falls f surjektiv, d.h. $f(B) = W$.

Gödels Unvollständigkeitssatz

Jedes Beweissystem B für W ist unvollständig.

Beweis

B rekursiv $\Rightarrow$ B rekursiv aufzählbar $\Rightarrow f(B)$ rekursiv aufzählbar. W ist aber nicht rekursiv aufzählbar. Also ist $f(B) \subsetneq W$.

Teil III

Komplexitätstheorie

Kapitel 13

Komplexitätstheorie

Komplexität eines Problems/Funktion = minimale Anzahl von Rechnungsschritten, die zur Lösung/Berechnung notwendig sind.

13.1 Komplexitätsklassen

M sei eine deterministische Turingmaschine mit Eingabealphabet $\mathcal{A}$.

Rechenzeit

$$t_M : \mathcal{A}^* \to \mathbb{N}_0$$

$$\alpha \mapsto \begin{cases} \text{minimale Zahl der Rechenschritte von } M \text{ bei Eingabe von } \alpha, \text{ um zu einem Endzustand zu kommen} & \text{falls } \alpha \in \mathcal{L}(M) \\ 0 & \text{sonst} \end{cases}$$

Komplexitätsklassen

$$\begin{aligned} \mathrm{DTIME}(t(n)) &:= \{\mathcal{L} \subset \mathcal{A} \mid \exists \text{ deterministische TM } M \text{ mit } \mathcal{L} = \mathcal{L}(M) \text{ und } t_M(\alpha) \leq \text{const. } f(|\alpha|)\} \\ \mathrm{NTIME}(f(n)) &:= \{\mathcal{L} \subset \mathcal{A}^* \mid \exists \text{ nichtdeterministische TM } M \text{ mit } \mathcal{L} = \mathcal{L}(M) \text{ und } t_M(\alpha) \leq \text{const. } f(|\alpha|)\} \\ \mathrm{P} &:= \bigcup_{k=1}^{\infty} \mathrm{DTIME}(n^k) \\ \mathrm{NP} &:= \bigcup_{k=1}^{\infty} \mathrm{NTIME}(n^k) \\ \mathrm{EXPTIME} &:= \bigcup_{k=1}^{\infty} \mathrm{DTIME}(2^{n^k}) \end{aligned}$$

Analog sind DSPACE, NSPACE, PSPACE, NPSPACE definiert.

Es gilt

$$\begin{array}{ccccccc} \mathrm{P} \subset \mathrm{NP} \subset \mathrm{PSPACE} & = & \mathrm{NPSPACE} & \subset & \mathrm{EXPTIME} \\ & \uparrow & & \uparrow & \\ & \text{ohne Beweis} & & \text{ohne Beweis} & \end{array}$$

$$\subset \texttt{loop} - \text{berechenbar} \subset \text{entscheidbar} \subset \text{rekursiv aufzählbar}$$

Es ist bis heute offen, ob P=NP.

13.2 NP-Vollständigkeit

Definition

$f : \mathcal{A}^* \to \mathcal{A}^*$ heißt *berechenbar*, falls es eine deterministische TM M gibt mit

$$\forall \alpha \in \mathcal{A}^* \quad \exists \text{Endzustand } p : q_0\alpha \overset{*}{\underset{M}{\Rightarrow}} p\, f(\alpha)$$

Definition

$\mathcal{L}_1, \mathcal{L}_2$ seine Sprachen über $\mathcal{A}$. Dann heißt $\mathcal{L}_1$ *polynomial* auf $\mathcal{L}_2$ *reduzierbar*, kurz

$$\mathcal{L}_1 \leq_p \mathcal{L}_2,$$

falls es eine mit polynomialer Laufzeit berechenbare Funktion $f : \mathcal{A}^* \to \mathcal{A}^*$ gibt mit

$$f(\mathcal{L}_1) \subset \mathcal{L}_2 \text{ und } f(\overline{\mathcal{L}_1}) \subset \overline{\mathcal{L}_2},$$

das heißt

$$f^-(\mathcal{L}_2) = \mathcal{L}_1.$$

Ist $\mathcal{S}$ eine Menge von Sprachen, die polynomial auf eine Sprache $\mathcal{L}$ reduziert werden können, schreiben wir kurz

$$\mathcal{S} \leq_p \mathcal{L}.$$

Lemma

$\mathcal{A} \leq_p \mathcal{B}$ und $\mathcal{B} \in \mathrm{P}$ (oder NP) $\Rightarrow$ $\mathcal{A} \in \mathrm{P}$ (bzw. NP).

Beweis

Seien F und M Turingmaschinen mit

$$\begin{aligned} t_F(\alpha) &= O(|\alpha|^k) \\ t_M(\alpha) &= O(|\alpha|^m). \end{aligned}$$

M akzeptiere $\mathcal{B}$ und F berechne eine Funktion mit

$$f^-\mathcal{B} = \mathcal{A}.$$

Dann gilt, daß $M \circ F$ die Sprache $\mathcal{A}$ akzeptiert und

$$t_{M\circ F}(\alpha) = O(|\alpha|^{km}),$$

wie zu beweisen war.

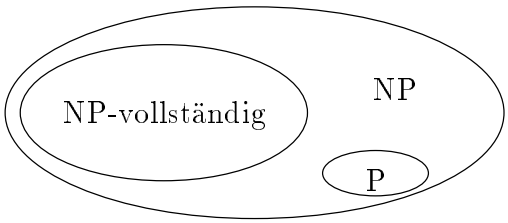


Abbildung 13.1: NP, P, NP-vollständig

Definition

Eine Sprache $\mathcal{A}$ heißt NP-*hart*, falls NP $\leq_p \mathcal{A}$ und sie heißt NP-*vollständig*, falls NP $\leq_p \mathcal{A} \in$ NP.

Satz

$\mathcal{A}$ sei NP-vollständig. Dann gilt

$$\mathcal{A} \in \mathrm{P} \quad \Leftrightarrow \quad \mathrm{P} = \mathrm{NP}$$

Beweis

$$\mathrm{NP} \quad \leq_p \quad \mathcal{A} \in \mathrm{P} \quad \overset{\Leftrightarrow}{\underset{\text{Lemma}}{}} \quad \mathrm{NP} = \mathrm{P}$$

Vermutung

Man vermutet, daß die NP-vollständigen Sprachen nicht in P lieigen, siehe Abbildung 13.1.

13.3 Das Erfüllbarkeitsproblem

Beispiel

$F = (x \vee y) \wedge (x \vee \overline{x} \vee z) \wedge (y \vee \overline{z})$ ist eine Formel der Aussagenlogik.
Für $x = 1$, $y = 1$, $z = 0$ ist $F = 1$, d.h. F ist eine *erfüllbare Formel*.

Bemerkung

Eine Formel kann als Wort über einem geeigneten Alphabet aufgefaßt werden.

Satz 1

SAT := $\{F \mid F$ ist eine erfüllbare Formel$\} \in$ NP

Beweis

Es gibt eine Turingmaschine M, die SAT akzeptiert:

- M belegt die Variablen einer Formel F nichtdeterministisch mit den Worten 0 oder 1

- berechnet dann deterministisch F
  - Gehe Formel von links an durch
  - Überschreibe aktuellen Term durch Wert der Teilformel von der letzten öffnenden Klammer bis zur aktuellen Position
  - Bei schließender Klammer ersetze Klammerausdruck durch einen Wert.
- Falls $F = 1$, akzeptiert M.

Es ist $t_M(|F|) = O(|F|^2)$ machbar.

Satz 2

SAT ist NP-vollständig.

Beweis

im nächsten Abschnitt

Bemerkung

SAT kann in 2^n Schritten deterministisch gelöst werden, d.h.

$$\text{SAT} \in \text{DTIME}(2^n).$$

Da

$$\text{NP} \leq_p \text{SAT}$$

folgt

$$\text{NP} \subset \bigcup_{k=1}^{\infty} \text{DTIME}(2^{n^k}) = \text{EXPTIME}$$

13.4 Beweis

In diesem Abschnitt wird gezeigt, daß SAT NP-hart ist.

Definition

$$G(x_1, \ldots, x_n) := \bigvee_{i=1}^{m} x_i \wedge \bigwedge_{i \neq j} \neg(x_i \wedge x_j)$$

Offenbar

- $G = 1$ genau dann, wenn nur ein $x_i = 1$.
- $|G| = O(m^2)$.

Satz

Sei $M = (Q, \mathcal{E}, q_0, \mathcal{A}, B, \Box, \delta)$ eine Turingmaschine und p ein Polynom mit

$$t_M(\alpha) \leq O(p(n)).$$

Weiter sei für alle Endzustände $q \in \mathcal{E}$ und alle $a \in B$

$$\delta(q, a) = (q, a, N).$$

Dann gibt es zu jedem Wert $\alpha = x_1 \ldots x_n \in \mathcal{A}^*$ eine Formel F, so daß

$$\alpha \in \mathcal{L}(M) \Leftrightarrow F \in \text{SAT}.$$

Beweis

Variablen von F	Bedeutung		
$z_{t,q}$	$z_{t,q} = 1$	gdw.	nach t Schritten ist M im Zustand q
$p_{t,i}$	$p_{t,i} = 1$	gdw.	nach t Schritten ist Kopf von M in Position i
$b_{t,i,a}$	$b_{t,i,a}$	gdw.	nach t Schritten ist auf Position i das Zeichen a

Dabei ist

$$\begin{aligned} t &= 0, 1, \ldots, p(n), \\ i &= -p(n), \ldots, p(n), \\ g &\in Q = \{q_0, \ldots, q_k\}, \\ a &\in B = \{a_1, \ldots, a_l\}. \end{aligned}$$

Die Formel F setzt sich wie folgt zusammen,

$$F = Z \wedge P \wedge B \wedge A \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E,$$

wobei

$$\begin{aligned} Z &= \bigwedge_t G(z_{t,q_0}, \ldots, z_{t,q_k}) \\ P &= \bigwedge_t G(P_{t,-p(n)}, \ldots, P_{t,p(n)}) \\ B &= \bigwedge_t \bigwedge_i G(b_{t,i,a_1}, \ldots, b_{t,i,a_l}). \end{aligned}$$

D.h. in jedem Zeitpunkt t befindet sich M in genau einem Zustand und in genau einer Position und auf jeder Position steht genau ein Zeichen. Weiter beschreiben

$$A = z_{0,q_0} \wedge \bigwedge_{i=1}^{n} b_{0,i,x_i} \wedge \bigwedge_{i \neq 1,\ldots,n} b_{0,i,\Box} \wedge p_{0,1}$$

die Anfangskonfiguration,

$$\begin{aligned} \ddot{U}_1 &= \bigwedge_{t,q,i,a} ((z_{t,q} \wedge p_{t,i} \wedge b_{t,i,a}) \\ &\Rightarrow \bigvee_{q',c} (z_{t+1,q'} \wedge p_{t+1,i+h} \wedge b_{t+1,i,c})), \\ &\quad \delta(q,a) \ni (q',c,h) \\ &\quad n \in \{-1,0,1\} \end{aligned}$$

den Übergang von Zeitpunkt t zu $t+1$, wobei

$$\ddot{U}_2 = \bigwedge_{t,i,a} ((\neg p_{t,i} \wedge b_{t,i,a}) \Rightarrow b_{t+1,i,a})$$

besagt, daß alle anderen Zeichen auf dem Band unverändert bleiben. Das Akzeptieren des Wortes α wird durch

$$E = \bigvee_{q \in \mathcal{E}} z_{p(n),q}$$

ausgedrückt. Offenbar gilt $\alpha \in \mathcal{L}(M) \Leftrightarrow F \subset \mathrm{SAT}$.

Folgerung

Die Formel F kann in polynomialer Zeit aus $\alpha = x_1, \ldots, x_n$ abgeleitet werden durch einfaches Hinschreiben. Die Laufzeit ist durch die Länge von F gegeben. Es gilt

$$\begin{aligned} |Z| &= O(p(n)) \\ |P| &= O(p^3(n)) \\ |B| &= O(p^2(n)) \\ |A| &= O(p(n)) \\ |\ddot{U}_1| &= O(p^2(n)) \\ |\ddot{U}_2| &= O(p^2(n)) \\ |E| &= O(1) \end{aligned}$$

gemessen in der Anzahl der Variablen. Die Kodierungslänge kommt als Faktor hinzu.

Damit ist bewiesen, daß SAT NP-vollständig ist.

13.5 3SAT

Definition

Eine Boolsche Formel F ist in *konjunktiver Normalform*, falls sie von folgender Form ist

$$F = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{m_i} x_{i,j},$$

d.h.

$$F = K_1 \wedge \cdots \wedge K_n,$$

wobei

$$K_i = x_{i,1} \vee \cdots \vee x_{i,m_i}.$$

Satz

3SAT:={erfüllbare Formeln in konjunktiver Normalform mit maximal 3 Variablen pro Klausel} ist NP-vollständig.

Beweis

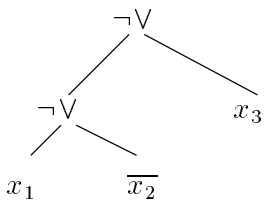
- Offensichtlich gilt 3SAT$\in$ NP.
- Nach 13.2 genügt es SAT$\leq_p$ 3SAT zu zeigen:
  Sei F eine beliebige Formel. Dazu konstruieren wir eine Formel F' mit

$$F \in \text{SAT} \Leftrightarrow F' \in \text{3SAT}.$$

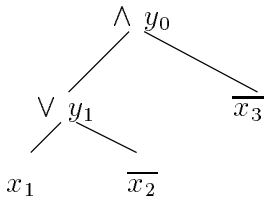
Erläuterung der Konstruktion am Beispiel

$$F = \neg(\neg(x_1 \vee \overline{x_2}) \vee x_3)$$

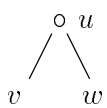
1. Fasse F als Baum auf.



2. Wende die Regel von de Morgan an, um Negationen zu den Variablen x_i zu bringen.



3. Ordne jedem inneren Knoten eine andere Variable y_j zu und außerdem jedem Teilbaum



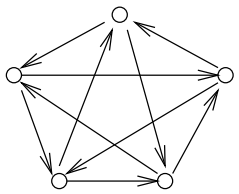


Abbildung 13.2: gerichteter Graph

die Teilformel $u \Leftrightarrow v \circ w$, zu, wobei $\circ \in \{\wedge, \vee\}$.

4. Verknüpfe y_0 und alle Teilformeln mit$\wedge$-Operatoren:

$$F_1 = y_0 \wedge (y_0 \Leftrightarrow y_1 \wedge x_3) \wedge (y_1 \Leftrightarrow x_1 \vee \overline{x_2}).$$

Dann gilt

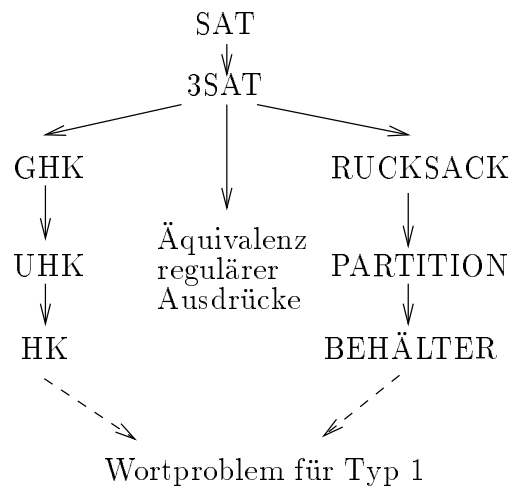
$$F_1 \in \text{SAT} \Leftrightarrow F \in \text{SAT}.$$

5. Forme F_1 in konjunktive Normalform um mit maximal 3 Variablen pro Klausel durch

$$\begin{array}{lllll} a \Leftrightarrow b \wedge c & = & & & (\overline{a} \vee b) \wedge (\overline{a} \vee c) \wedge (a \vee \overline{b} \vee \overline{c}) \\ a \Leftrightarrow b \vee c & = & \overline{a} \Leftrightarrow \overline{b} \wedge \overline{c} & = & (a \vee \overline{b}) \wedge (a \vee \overline{c}) \wedge (\overline{a} \vee b \vee c) \end{array}$$

Also kann F in linearer Zeit auf eine Formel F' in konjunktiver Normalform mit maximal 3 Variablen/Klauseln überführt werden. QED

Überblick über das Folgende (Die Pfeile stehen für $\leq_p$.)

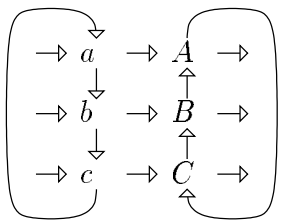


13.6 Gerichteter Hamiltonkreis

Sei $G = (V, E)$ ein gerichteter Graph (Abbildung 13.2 zeigt ein Beispiel). $V := \{v_1, \ldots, v_n\}$. Ein *Hamilton-Kreis* ist eine Permutation $(\pi_1, \ldots, \pi_n)$ von $(1, \ldots, n)$, so daß $(v_{\pi_1}, v_{\pi_2}), \ldots, (v_{\pi_{n-1}}, v_{\pi_n}), (v_{\pi_n}, v_{\pi_1}) \in E$.

Vorbemerkung

Ist K ein Teilgraph von G der Form



gibt es folgende einfache Wege von a aus K hinaus

$$
\begin{array}{ll}
* & a\,A \\
 & a\,A\,C \\
 & a\,A\,C\,B \\
 & a\,b\,B \\
* & a\,b\,B\,A \\
 & a\,b\,B\,A\,C \\
 & a\,b\,c\,C \\
 & a\,b\,c\,C\,B \\
* & a\,b\,c\,C\,B\,A
\end{array}
$$

und entsprechende Wege von b und c aus.
Nur die mit $*$ markierten Wege können Teil eines Hamilton-Kreises sein. Bei den anderen käme man nie zu A oder von c aus nicht weiter.

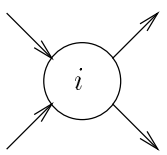
Satz

GHK $=$ {gerichtete Graphen mit Hamilton $-$ Kreis} ist NP-vollständig.

Beweis

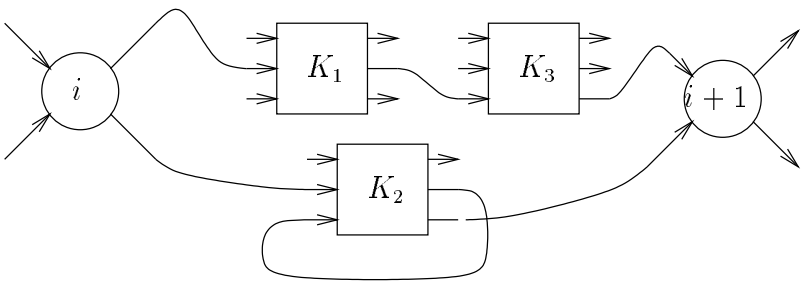
- Offenbar gilt GHK $\in$ NP.
- Nach 13.2 und 13.5 genügt es 3SAT $\leq_p$ GHK zu zeigen:

Sei F eine Formel in konjunktiver Normalform mit m Klauseln und maximal drei Variablen pro Klausel. $x_1, \ldots, x_n$ seien die Variablen von F.
Zu F konstruieren wir einen Graphen G bestehend aus den Teilgraphen



$i = 1, \ldots, n,$

und den Teilgraphen $K_1, \ldots, K_m$ mit der Form wie in der Vorbemerkung.
Die Verknüpfung der Teilgraphen wird am Beispiel erklärt:
Falls x_i in Klausel 1 an 2. Stelle, in Klausel 3 an 3. Stelle und $\overline{x_i}$ in Klausel 2 an 2. Stelle steht, dann führen folgende Wege vom Knoten i zum Konten $i+1$:



Der Weg verläßt ein K_j auf der selben Höhe, wie er in K_j eintritt.

Dann gilt $F \in 3SAT \Leftrightarrow G \in GHK$, denn:

" $\Rightarrow$ " Betrachte eine Belegung, die F erfüllt. Dazu konstruiere einen Hamilton-Kreis wie folgt. Ist $x_i = 1$, verlasse i nach oben und sonst nach unten und folge dem oben konstruierten Weg nach $i+1$. Da alle Klauseln erfüllt sind, werden alle K_j durchlaufen.
Der Weg kann so eingerichtet werden, daß in jedem K_j alle Knoten durchlaufen werden.

" $\Leftarrow$ " Betrachte einen Hamilton-Kreis und belege die x_i wie folgt

$$x_i = \begin{cases} 1 & \text{falls Hamilton} - \text{Kreis den Knoten } i \text{ nach oben verläßt} \\ 0 & \text{sonst} \end{cases}$$

D.h. der Hamilton-Kreis durchläuft nur Klauseln, in denen x_i mit Belegung 1 vorkommt oder x_i mit Belegung 0. Also sind alle Klauseln und damit F erfüllt.

13.7 Ungerichteter Hamiltonkreis und Handlungsreisender

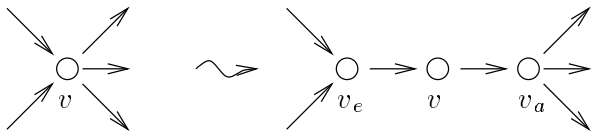
Satz

UHK := {ungerichtete Graphen mit Hamilton − Kreis} ist NP-vollständig.

Beweis

- UHK $\in$ NP $\surd$
- Zeige GHK $\leq_p$ UHK

  Gegeben sei ein gerichteter Graph. Dazu wird ein ungerichteter Graph wie folgt konstruiert:
  Nimm zu jedem Knoten v je einen Knoten v_e und v_a hinzu.



v_e erhält die eingehenden Kanten von v, v_a erhält die ausgehenden Kanten von v. Außerdem nimm die Kanten $v_e v$ und $v v_a$ hinzu. Lasse alle Richtungen fort.

Für diesen ungerichteten Graphen gilt

1. Hat der gerichtete Graph einen Hamilton-Kreis, dann auch der ungerichtete.
2. Hat der ungerichtete Graph einen Hamilton-Kreis, dann auch der gerichtete. Denn unmittelbar vor und nach Durchlaufen eines Knotens v im ungerichteten Hamilton-Kreis werden die zugehörigen Knoten v_e und v_a durchlaufen.

Also gilt GHK $\leq_p$ UHK.

Satz

HR := $\{(M,k) \mid M$ ist eine $n \times n$ Matrix von Entfernungen $m_{ij} \in \mathbb{R}_{+,0}$ und es existiert eine Permutation $(\pi_1, \ldots, \pi_n)$ von $(1, \ldots, n)$, so daß $m_{\pi_1\pi_2} + m_{\pi_2\pi_3} + \ldots + m_{\pi_n\pi_1} \leq k\}$

ist NP-vollständig.

Beweis

- HR $\in$ NP $\surd$
- Zeige UHK $\leq_p$ HR

  Sei $G = (\{1, \ldots, n\}, E)$ ein Graph und $m_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 2 & \text{falls } (i,j) \notin E \end{cases}$.

  Dann ist π ein Hamilton-Kreis von G genau dann, wenn $m_{\pi_1\pi_2} + \ldots + m_{\pi_n\pi_1} = n$ bzw. genau dann, wenn $(M,n) \in$ HR.

  Also ist UHK $\leq_p$ HR.

Folgerung

Wäre eine kürzeste Rundreise durch einen gerichteten Graphen in polynomialer Zeit bestimmbar, wäre auch HR $\in$ P.

Bemerkung

Ein *Euler-Kreis* ist ein Weg, der alle Kanten genau einmal durchläuft. Ein *Euler-Graph* ist ein Graph mit Euler-Kreis.
Offensichtlich ist ein Graph ein Euler-Graph, wenn alle Knoten geraden Grad haben.
Also gilt: {Euler-Graphen} $\in$ P.

13.8 Das Problem des Handlungsreisenden im $\mathbb{R}^2$

Gegeben: $V := \{\mathbf{p}_1, \ldots, \mathbf{p}_n\}$ Punktmenge des $\mathbb{R}^2$.

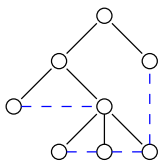


Abbildung 13.3: Tiefendurchlauf eines minimalen Spannbaums

Gesucht: möglichst kurze Rundreise.

(V, V^2) ist ein Graph. Die Kanten bekommen als „Kosten“ ihre Längen zugeordnet.

Lemma

Ein minimaler Spannbaum kann in polynomialer Zeit konstruiert werden.

Beweis bekannt

Bemerkung

Ein minimaler Spannbaum kann auch mit Aufwand $O(n \log n)$ konstruiert werden.

Lemma 2

Kosten des minimalen Spannbaums $\leq$ $R :=$ Länge der kürzesten Rundreise

Beweis

Entfernt man aus einer Rundreise eine Kante, erhält man einen Spannbaum.

Konstruktion einer Rundreise

Ein Tiefendurchlauf des minimalen Spannbaums liefert einen Rundweg. Dieser wird kürzer, wenn Knoten nicht doppelt besucht werden, sondern stattdessen zum nächsten noch nicht besuchten Knoten gesprungen wird (siehe Abbildung 13.3, gestrichelte Linien). Die Sprünge sind Abkürzungen, weil die Dreiecksungleichung gilt. Also erhalten wir die Abschätzung

$$|\text{Rundreise}| \leq 2 \cdot |\text{min. Spannbaum}| \leq 2R.$$

Verbesserung

Erweitere minimalen Spannbaum zum Euler-Graphen, indem die Knoten mit ungeradem Knotengrad gepaart werden. Im folgenden wird gezeigt, daß dies möglich ist und der Euler-Graph nicht länger als $3R/2$ ist.
In einem Graphen gilt:

$$\#\,\text{Kanten} \cdot 2 = \sum \text{Knotengrade} \in 2\mathbb{N}_0.$$

Daher ist die Zahl der Knoten mit ungeradem Grad gerade.

Lemma 3

Eine minimale Paarung kann in polynomialer Zeit gefunden werden (ohne Beweis).

Lemma 4

Die Länge einer minimalen Paarung ist $\leq \frac{R}{2}$

Beweis

Färbe jede 2. Kante einer minimalen Rundreise durch die Knoten des Spannbaums mit ungeradem Grad. Dann erhält man zwei Paarungen mit Gesamtlänge R. D.h. es gibt eine Paarung der Länge $\leq \frac{R}{2}$.

Zusammenfassend haben wir gezeigt

$$|\text{Euler-Graph}| = |\text{min. Spannbaum}| + |\text{min. Paarung}| \leq R + \frac{R}{2} = \frac{3}{2}R$$

13.9 Das Rucksack-Problem

Satz

$$\text{RUCKSACK} := \{(a_0, \ldots, a_k) \subset \mathbb{N}^{k+1} \mid \exists I \subset \{1, \ldots, n\} : \sum_{i \in I} a_i = a_0\}$$

ist NP-vollständig.

Beweis

- RUCKSACK $\in$ NP $\quad \surd$
- 3SAT $\leq_p$ RUCKSACK :

  Sei

  $$F(x_1, \ldots, x_m) = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} y_{ij}$$

  eine Formel. Dann konstruieren wir folgende $m + n$-stellige Zahlen in Dezimaldarstellung:

$$
\begin{array}{lcllllllllll}
a & = & 4 & \ldots & 4 & 1 & 1 & \ldots & & 1 \\
a_1 & = & k_{11} & \ldots & k_{11} & 1 & 0 & \ldots & & 0 \\
\overline{a}_1 & = & \overline{k}_{11} & \ldots & \overline{k}_{11} & 1 & 0 & \ldots & & 0 \\
\vdots & \vdots & & & & & & & & \\
a_m & = & k_{m1} & \ldots & k_{mn} & & 0 & \ldots & 0 & 1 \\
\overline{a}_m & = & \overline{k}_{m1} & \ldots & \overline{k}_{mn} & & 0 & \ldots & 0 & 1 \\
b_1 & = & 1 \quad 0 & \ldots & & & & & & 0 \\
c_1 & = & 2 \quad 0 & \ldots & & & & & & 0 \\
\vdots & \vdots & & & & & & & & \\
b_n & = & 0 & \ldots 0 & 1 & 0 & 0 & \ldots & & 0 \\
c_n & = & 0 & \ldots 0 & 2 & 0 & 0 & \ldots & & 0,
\end{array}
$$

wobei

$$k_{ij} := \begin{cases} 1 & \text{falls } x_i \text{ in Klausel } j \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

und

$$\overline{k}_{ij} := \begin{cases} 1 & \text{falls } \overline{x}_i \text{ in Klausel } j \text{ vorkommt} \\ 0 & \text{sonst} \end{cases} .$$

Dann gilt $F(x_1, \ldots, x_n) = 1$

$\Leftrightarrow$ $\forall j = 1, \ldots, n$

$\exists$ genau $1-3$ Variablen x_i mit $k_{ij} = 1$ und $x_i = 1$ oder $\overline{k}_{ij} = 1$ und $\overline{x}_i = 1$

$$\Leftrightarrow \quad 1 \ldots 1\, 1 \ldots 1 \leq \sum_{x_i = 1} a_i + \sum_{\overline{x}_i = 1} \overline{a}_i \leq 3 \ldots 3\, 1 \ldots 1$$

$$\Leftrightarrow \quad \sum_{x_i = 1} a_i + \sum_{\overline{x}_i = 1} \overline{a}_i + \text{ausgewählte } b_i \text{ und } c_i = 4 \ldots 4\, 1 \ldots 1 = a$$

$\Leftrightarrow$ $(a, a_1, \overline{a}_1, \ldots, a_m, \overline{a}_m, b_1, c_1, \ldots, b_n, c_n) \in$ RUCKSACK

Offenbar ist die angegebene Transformation in polynomialer Zeit durchführbar, so daß die Behauptung folgt.

13.10 Partition

Satz

$$\text{PARTITION} := \{(a_1, \ldots, a_k) \in \mathbb{N}^k \mid \exists I \subset \{1, \ldots, k\} : \sum_{i \in I} a_i = \sum_{i \notin I} a_i\}$$

ist NP-vollständig.

Beweis

- PARTITION ist in NP $\surd$
- RUCKSACK $\leq_p$PARTITION:

$$
\begin{aligned}
& (a_0, \ldots, a_k) \in \text{RUCKSACK} \\
\Leftrightarrow\ & \text{ObDA } a_0 = a_1 + \ldots + a_i \\
\Leftrightarrow\ & \text{ObDA } \sum a_i + a_0 + \overbrace{a_{i+1} + \ldots + a_k}^{=:a} = a_1 + \ldots + a_i + a + \sum a_i \\
\Leftrightarrow\ & (a + \sum_{i=1}^{k} a_i, a_0 + \sum_{i=1}^{k} a_i, a_1, \ldots, a_k) \in \text{PARTITION}
\end{aligned}
$$

13.11 Das Behälterproblem

Satz

$$
\begin{aligned}
\text{BEHÄLTER} := \quad & \{(k, b, a_1, \ldots, a_n) \in \mathbb{N}^{n+2} \mid \exists \text{ Zuordnung } f : \{1, \ldots, n\} \\
& \to \{1, \ldots, k\}, \forall j = 1, \ldots, k : \sum_{f(i)=j} a_i = b\}
\end{aligned}
$$

ist NP-vollständig.

Beweis

- BEHÄLTER$\in$ NP $\quad\surd$
- PARTITION $\leq_p$ BEHÄLTER:

$$
\begin{aligned}
& (a_1, \ldots, a_m) \in \text{PARTITION} \\
\Leftrightarrow\ & (2, \frac{a_1 + \ldots + a_m}{2}, a_1, \ldots, a_m) \in \text{BEHÄLTER}
\end{aligned}
$$

13.12 Äquivalenz regulärer Ausdrücke

Satz

ÄRA $:= \{(R_1, R_2) \mid$ Die regulären Ausdrücke R_1 und R_2 sind äquivalent$\}$
ist NP-hart.

Beweis

Zeige 3SAT $\leq_p$ ÄRA. Sei $F(x_1, \ldots, x_n)$ eine Formel in konjunktiver Normalform mit k Klauseln. Zu F konstruieren wir zwei reguläre Ausdrücke R und S über $\{0, 1\}$,

$$R := (\mathbf{v}_1 + \overline{\mathbf{v}}_1) \cdots (\mathbf{v}_n + \overline{\mathbf{v}}_n),$$

wobei

$$\mathbf{v}_i = v_{i1} \ldots v_{ik}$$

mit

$$v_{ij} = \begin{cases} 1 & \text{falls } x_i \text{ in Klausel } j \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}.$$

Analog definieren wir $\overline{\mathbf{v}}_i$ in Abhängigkeit von $\overline{x}_i$.
Damit ist F erfüllbar genau dann, wenn es eine „Belegung“

$$b_{11} \ldots b_{1k} \ldots b_{n1} \ldots b_{nk} \in \mathcal{L}(R)$$

für die x_i gibt, so daß die Matrix

$$\begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix}$$

keine Nullspalte besitzt.
Weiter sei

$$S = N_1 + \cdots + N_k$$

mit den regulären Ausdrücken

$$N_j = ((0+1)^{j-1}0(0+1)^{k-j})^n,$$

die alle Matrizen beschreiben, deren j-te Spalte Null ist. Dann gilt

$$\begin{aligned} F \text{ erfüllbar} &\Leftrightarrow \mathcal{L}(R) \not\subset \mathcal{L}(S) \\ &\Leftrightarrow \mathcal{L}(R+S) \neq \mathcal{L}(S) \\ &\Leftrightarrow (R+S, S) \notin \text{ÄRA}. \end{aligned}$$

Daraus folgt die Behauptung.

Bemerkung

Gezeigt wurde sogar, daß die Äquivalenz ungesternter regulärer Ausdrücke NP-hart ist.

Bemerkung

Zu einem regulären Ausdruck kann in polynomialer Zeit ein entsprechender NFA bzw. eine reguläre Grammatik konstruiert werden. D.h. auch die Äquivalenz von NFAs ist NP-hart. Dagegen bilden äquivalente Paare von DFAs eine Sprache in P.

13.13 Das Wortproblem für kontextsensitive Sprachen

Satz

Das Wortproblem für kontextsensitive Sprachen, $\{(G, \omega) \mid \omega \in \mathcal{L}(G), G \text{ kontextsensitiv}\}$, ist NP-hart.

Beweis

Zeige NP $\leq_p$Wortproblem. Sei M eine Turingmaschine und sei $t_M(\alpha) = O(p(\alpha))$, wobei p ein Polynom ist.
In 4.4 wurde eine kontextsensitive Grammatik G konstruiert, die M simuliert, solange M auf dem Bandbereich der Eingabe bleibt. Wird G um die Produktion

$$S \to (\square\,\square)^{p(n)} S (\square\,\square)^{p(n)}$$

erweitert, kann G die Turingmaschine M bei Eingabe von $\omega = a_1, \ldots, a_n$ simulieren. OBdA kann angenommen werden, daß M beim Akzeptieren das Band aufräumt. Also kann das Problem $\omega \in \mathcal{L}(M)$ polynomial auf das Problem

$$(\square\,\square)^{p(n)} \overbrace{\square\,\square}^{n} (\square\,\square)^{p(n)} \in \mathcal{L}(G)$$

reduziert werden. Weil G kontextsensitv ist, folgt die Behauptung.

Teil IV

Stochastische Algorithmen

Kapitel 14

Stochastische Algorithmen

14.1 Einführung

Beispiel

14 Kühe werden abends zufällig im Stall verteilt. Wie viele Kühe stehen im Mittel in ihrer Box?
Sei

$$X_i := \begin{cases} 1 & \text{falls } i\text{-te Kuh am eigenen Platz} \\ 0 & \text{sonst} \end{cases}$$

eine *Zufallsvariable,* d.h. eine Funktion über allen Permutationen. Zufallsvariablen, die auf $\{0,1\}$ abbilden, heißen *Indikatorvariablen.* Offenbar gilt

$$p \quad := \quad \Pr[X_i = 1] = \frac{1}{14}$$

und

$$\begin{aligned} E[X_i] &= && \text{Erwartungswert von } X_i \\ &\overset{\text{Indikatorvariable}}{=} && 0 \cdot (1-p) + 1 \cdot p \\ &= && p. \end{aligned}$$

Der Erwartungswert ist ein lineares Funktional. Also folgt

$$E\left[\sum_{i=1}^{14} X_i\right] = \sum_{i=1}^{14} E[X_i] = 14 \cdot \frac{1}{14} = 1.$$

14.2 Sortieren durch stochastisches Teilen

Wir betrachten folgenden Algorithmus.

Eingabe: $S := \{s_1, \ldots, s_n\} \subset \mathbb{N}$ mit $s_i \neq s_j$ für $i \neq j$

Ausgabe: S in sortierte Reihenfolge

1. Wähle ein Pivotelement y zufällig.

2. Zerlege $S \setminus \{y\}$ in S_1 und S_2 mit $S_1 < y < S_2$
3. Sortiere rekursiv S_1 und S_2 und gib erst S_1 sortiert, dann y, dann S_2 sortiert aus.

Aufwand (gemessen in der Anzahl der Vergleiche s_i mit s_j)

Sei

$$X_{ij} := \begin{cases} 1 & \text{falls } s_i \text{ mit } s_j \text{ verglichen wird} \\ 0 & \text{sonst} \end{cases}$$

und

$$p_{ij} := \Pr[X_{ij} = 1].$$

Dann gilt

$$E[X_{ij}] = p_{ij}.$$

Zur Herleitung von p_{ij} sei ohne Einschränkung der Allgemeinheit $s_1 < \ldots < s_n$. Dann folgt $X_{ij} = 1$ genau dann, wenn s_i oder s_j vor $s_{i+1}, \ldots, s_{j-1}$ als Pivotelement gewählt werden. Also ist

$$p_{ij} = \frac{2}{j - i + 1}.$$

Der erwartete Aufwand ist damit

$$\begin{aligned} & E[\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{ij}] = \sum \sum E[X_{ij}] \\ = \; & \sum \sum p_{ij} \\ \leq \; & 2 \cdot \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\ = \; & 2nH_n \\ = \; & O(n \log n), \text{ wobei } H_n = \sum_{k=1}^{n} \frac{1}{k} \text{ die } n-\text{te } \textit{harmonische Zahl} \text{ ist.} \end{aligned}$$

14.3 Der minimale Schnitt

Definition

Teilmengen einer Stichprobenmenge heißen *Ereignisse*. Zwei Ereignisse $\mathcal{E}_1$ und $\mathcal{E}_2$ heißen *unabhängig*, falls

$$\Pr[\mathcal{E}_1 \cap \mathcal{E}_2] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2].$$

Für abhängige und unabhängige Ereignisse $\mathcal{E}_1$ und $\mathcal{E}_2$ gilt allgemein

$$\Pr[\mathcal{E}_1 \cap \mathcal{E}_2] = \Pr[\mathcal{E}_1 \mid \mathcal{E}_2] \cdot \Pr[\mathcal{E}_2],$$

wobei $\Pr[\mathcal{E}_1 \mid \mathcal{E}_2]$ die *bedingte Wahrscheinlichkeit* für das Eintreten von $\mathcal{E}_1 \cap \mathcal{E}_2$ ist, nachdem $\mathcal{E}_2$ eingetreten ist. D.h. es gilt

$$\Pr[\mathcal{E}_1 \mid \mathcal{E}_2] = \frac{\Pr[\mathcal{E}_1 \cap \mathcal{E}_2]}{\Pr[\mathcal{E}_2]}.$$

Allgemein gilt

$$\Pr\left[\bigcap_{i=1}^{n} \mathcal{E}_i\right] = \Pr[\mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1] \cdot \Pr[\mathcal{E}_2 \mid \mathcal{E}_1 \cap \mathcal{E}_2] \ldots \Pr\left[\mathcal{E}_n \mid \bigcap_{i=1}^{n-1} \mathcal{E}_i\right].$$

Definition

Ein Graph mit mehreren Kanten zwischen zwei Knoten heißt *Multigraph.* Sei G ein Multigraph. Ein *Schnitt* von G ist eine Menge von Kanten, ohne die G in mehrere Komponenten zerfällt.

Algorithmus zur Berechnung eines Schnittes

Solange G noch 3 Knoten hat,

1. wähle zufällig eine Kante und verschmelze die beiden Endknoten mit allen verbindenden Kanten.

2. Gib die übrig gebliebenen Kanten aus.

Frage: Wie oft ist ein Schnitt minimal?
Zur Beantwortung dieser Frage betrachten wir einen minimalen Schnitt K und definieren

$$\begin{aligned} k &:= |K|, \\ n &:= \#\text{ Knoten in } G, \\ \mathcal{E}_i &:= \text{Ereignis, daß im Schritt } i \text{ des Algorithmus} \\ &\quad\ \text{keine Kante aus } K \text{ gewählt wird für } i = 1, \ldots, n-2. \end{aligned}$$

Daraus folgt

$$\begin{aligned} &|\text{Knoten zu Beginn von Schritt } i| = n - i + 1, \\ &\text{minimaler Knotengrad} \geq k, \\ &|\text{Kanten zu Beginn von Schritt } i| \geq k \cdot (n - i + 1)/2, \\ &\Pr\left[\,\overline{\mathcal{E}_1}\,\right] \leq \frac{k}{k\,n/2} = \frac{2}{n}, \\ &\Pr\left[\,\overline{\mathcal{E}_i} \mid \mathcal{E}_1 \cap \cdots \cap \mathcal{E}_{i-1}\,\right] \leq \frac{k}{k(n-i+1)/2} = \frac{2}{n-i+1} \end{aligned}$$

und hieraus

$$\begin{aligned}
\Pr\left[\bigcap_{i=1}^{n-2}\mathcal{E}_i\right] &= \Pr[\mathcal{E}_1]\cdot\Pr[\mathcal{E}_2\mid\mathcal{E}_1]\cdot\Pr[\mathcal{E}_3\mid\mathcal{E}_1\cap\mathcal{E}_2]\ldots\Pr\left[\mathcal{E}_{n-2}\mid\bigcap_{i=1}^{n=3}\mathcal{E}_i\right]\\
&\geq (1-\frac{2}{n})(1-\frac{2}{n-1})\cdots(1-\frac{2}{3})\\
&= \frac{n-2}{2}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdots\frac{1}{3}\\
&= \frac{2}{n(n-1)}\\
&\geq \frac{2}{n^2}
\end{aligned}$$

Also gilt für die Wahrscheinlichkeit p, daß K nicht gefunden wird

$$p \leq 1-2/n^2$$

Nach $n^2/2$ Wiederholungen des Algorithmus ist

$$p \leq (1-\frac{2}{n^2})^{n^2/2} \leq \frac{1}{e} = 0.37,$$

nach n^2 Wiederholungen

$$p \leq \frac{1}{e^2} = 0.14,$$

nach n^3 und n^4 Wiederholungen

$$p \leq \frac{1}{e^3} = 0.05 \quad \text{bzw.} \quad p \leq \frac{1}{e^4} = 0.02.$$

Definition

Ein stochastischer Algorithmus, der immer ein richtiges Ergebnis liefert, heißt ein *Las-Vegas-* und sonst *Monte-Carlo-Algorithmus.*

14.4 Komplexitätsklassen

Eine *stochastische Turingmaschine* ist eine Turingmaschine, die indeterministische Entscheidungen zufällig und gleichverteilt trifft.
Akzeptiert eine stochastische Turingmaschine M ein Wort $\alpha \in \mathcal{L} = \mathcal{L}(M)$, dann schreiben wir $M(\alpha) = 1$ und andernfalls $M(\alpha) = 0$.

Definition

Mit RP bezeichnen wir die Menge der Sprachen $\mathcal{L}$, die durch eine stochastische Turingmaschine mit polynomial beschränkter Laufzeit akzeptiert werden, so daß

$$(*) \qquad \Pr[M(\alpha)=1]\begin{cases} \geq \frac{1}{2} & \text{falls } \alpha\in\mathcal{L} \\ = 0 & \text{sonst}\end{cases}.$$

Bemerkung

Ohne Einschränkung der Allgemeinheit kann die Schranke $\frac{1}{2}$ durch ein $c \in (0,1)$ ersetzt werden. Denn ist $\Pr[M(\alpha)=1] \geq c$ für $\alpha \in \mathcal{L}$, dann gilt für die Turingmaschine A mit

$$A(\alpha) = \texttt{if } \overbrace{M(\alpha)=1 \texttt{ or } \ldots \texttt{ or } M(\alpha)=1}^{n \text{ unabhängige Aufrufe von } M} \texttt{ then } 1 \texttt{ else } 0$$

das folgende

$$\Pr[A(\alpha)=0] \leq (1-c)^n \underset{n\to\infty}{\to} 0,$$

d.h.

$$\Pr[A(\alpha)=1] \geq 1-(1-c)^n \underset{n\to\infty}{\to} 1.$$

Also ist $\mathcal{L}(M) = \mathcal{L}(A) \in \mathrm{RP}$.

Bezeichnung

Eine Turingmaschine M mit der Eigenschaft $(*)$ heißt RP-Turingmaschine: Synonym wollen wir M auch als RP-Algorithmus bezeichnen.

Bemerkung

RP-Algorithmen können sich nur irren, falls $\alpha \in \mathcal{L}$. Man nennt sie *Monte-Carlo-Algorithmen mit einseitigem Fehlverhalten.*

Definition

co-RP $:= \{\mathcal{L} \mid \overline{\mathcal{L}} \in \mathrm{RP}\}$ ist Menge aller Sprachen $\mathcal{L}$, die von einer stochastischen Turingmaschine M akzeptiert werden mit

$$\Pr[M(\alpha)=1] \begin{cases} =1 & \text{falls } \alpha \in \mathcal{L} \\ \leq \frac{1}{2} & \text{sonst} \end{cases}.$$

Definition (zero-sided error)

ZPP := RP$\cap$co-RP

Definition

PP=Menge aller Sprachen $\mathcal{L}$, die durch eine stochastische Turingmaschine M erkannt werden mit

$$\Pr[M(\alpha)=1] \begin{cases} > \frac{1}{2} & \text{falls } \alpha \in \mathcal{L} \\ < \frac{1}{2} & \text{falls } \alpha \notin \mathcal{L} \end{cases}.$$

Aufgabe

Ein Algorithmus mit Fehlerwahrscheinlichkeit $\frac{1}{2} - \frac{1}{2^n}$ kann durch polynomial viele Wiederholungen nicht in einen Algorithmus mit Fehlerwahrscheinlichkeit $\leq \frac{1}{4}$ umgewandelt werdern.

Definition

BPP-Algorithmen sind PP-Algorithmen mit Irrtumswahrscheinlichkeit $\leq \frac{1}{4}$.

Bemerkung

Irrtumswahrscheinlichkeit eines BPP-Algorithmuses kann durch polynomial viele Wiederholungen beliebig klein gemacht werden.

Offen

- $\mathrm{RP} = \mathrm{co} - \mathrm{RP}$?
- $\mathrm{BPP} \subset \mathrm{NP}$?

Beispiel

Der Algorithmus in 14.3 kann in einen PP-Algorithmus umgewandelt werden, der

$$\{(G,k) \mid \text{die minimalen Schnitte von } G \text{ haben Größe } k\}$$

berechnet und sogar in einen BPP-Algorithmus, aber nicht in einen RP-Algorithmus.
Leicht abgewandelt berechnet dieser Algorithmus die Sprache

$$\{(G,k) \mid G \text{ hat minimale Schnitte der Größe } \leq k\}$$

Es ist ein RP-Algorithmus.

14.5 Binäre Zerlegungen der Ebene

Geg.: $S_1, \ldots, S_n$ disjunkte Strecken im $\mathbb{R}^2$. G_i sei die von S_i aufgespannte Gerade (siehe Abbildung 14.1).

Ges.: Ein Binärbaum, dessen Knoten die S_i oder Teile der S_i enthalten, so daß die S_j im linken/rechten Teilbaum eines Knotens S_i links/rechts von G_i liegen.

Anwendung

Darstellung durch Übermalen

- Zeichne zuerst die Objekte *hinter* der Wurzel
- dann die Wurzel
- dann die Objekte *vor* der Wurzel (relativ zum Betrachter)

und entsprechend für jeden Teilbaum.

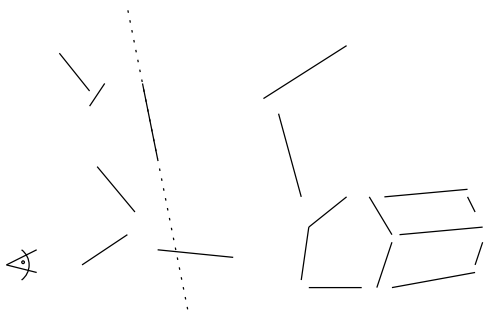


Abbildung 14.1: $S_1, \ldots, S_n$ und die aufgespannte Gerade G_i

Algorithmus

zur Berechnung einer binären Zerlegung von $\mathbb{R}^2$:

1. Wähle zufällig eine Permutation Π von $(1, \ldots, n)$
2. Konstruiere den binären Zerlegungsbaum so, daß jedes S_i vor allen S_j in den Nachfolgeknoten bezüglich Π ist

Satz

Erwartete Baumgröße $= O(n \log n)$

Beweis

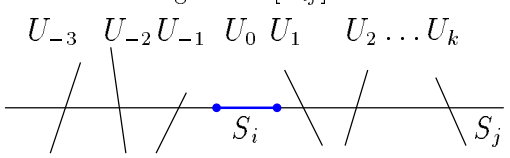
$i \neq j$ kurz für: G_i schneidet S_j und beide Teile sind Nachfolger von S_i.

$$X_{ij} := \begin{cases} 1 & \text{falls } i \neq j \\ 0 & \text{sonst} \end{cases}$$

Die mittlere Baumgröße ist damit

$$\begin{aligned} & n + E\left[\sum_{i=1}^{n}\sum_{j\neq i} X_{ij}\right] \underset{\text{s.u.}}{\leq} n + \sum_{i=1}^{n}\sum_{j\neq i}\frac{1}{k_{ij}+1} \\ \underset{\text{s.u.}}{\leq} \quad & n + \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{2}{j} \leq n + n \cdot H_n \\ = \quad & O(n \log n). \end{aligned}$$

Zur Berechnung von $E[X_{ij}]$:



$\ldots, U_{-1}, U_0, U_1, \ldots$ seien die von G_i geschnittenen Strecken S_j sortiert nach der Entfernung ihres Schnittpunktes mit G_i von S_i. Ist $U_k = S_j$, hängt k von i und j ab, also $k = k_{ij}$.

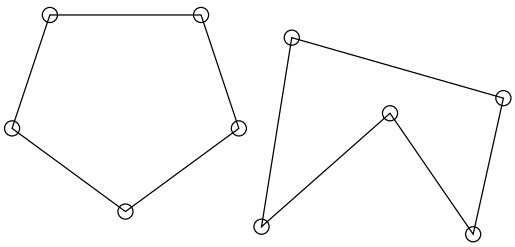


Abbildung 14.2: Ein konvexes und ein nichtkonvexes Polygon

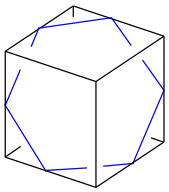


Abbildung 14.3: Zerlegung eines räumlichen Polyeders.

Offenbar folgt an

$i + j$, daß S_i bezüglich Π vor $U_1, \ldots, U_k$ kommt. Also ist
$\Pr[i+j] \leq \dfrac{1}{k_{ij}+1}$.

Das Ungleichheitszeichen deshalb, weil es außer den $U_1, \ldots, U_{k-1}$ andere G_l geben kann, die S_i und S_j trennen.

Bemerkung

Da erwartete Baumgröße $= O(n \log n)$, gibt es einen Baum der Größe $O(n \log n)$.

14.6 Binäre Raumzerlegung

Definition

Sei P ein konvexes, planares, orientiertes Polygon (siehe Abbildung 14.2) und P_+ und P_- die beiden Halbräume auf deren Rändern P liegt.

Eine rekursive Zerlegungsprozedur

für ein räumliches Polyeder G mit einer Menge $\mathcal{P}$ konvexer, planarer, orientierter Polygone $P \subset G$.
`Zerlege` $(G, \mathcal{P})$

1. Suche ein $P \in \mathcal{P}$, das G zerlegt. (Abbildung 14.3 zeigt ein zerlegendes Polygon)

2. Gibt es kein solches P, dann wähle zufällig ein anderes P.

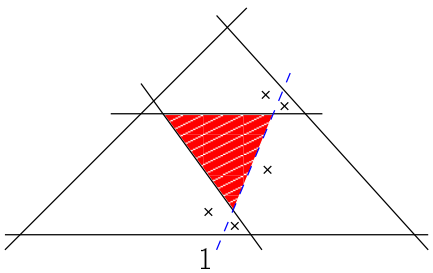


Abbildung 14.4: $\gamma_1 = 5$

3. Gibt es kein $P \neq \emptyset$, dann stop.
4. `Zerlege` $(G \cap P_+, \{Q \cap P_+ \mid Q \in \mathcal{P}\})$
5. `Zerlege` $(G \cap P_-, \{Q \cap P \mid Q \in \mathcal{P}\})$

Satz

Seien $\triangle_1, \ldots, \triangle_n$ disjunkte Dreiecke im $\mathbb{R}^3$. `Zerlege` $(\mathbb{R}^3, \{\triangle_1, \ldots, \triangle_n\})$ ergibt eine Zerlegung des $\mathbb{R}^3$ in $O(n^2)$ Gebiete.

Beweis

1. Betrachte k Geraden, die ein Dreieck schneiden, wie in Abbildung 14.4 illustriert. Sie zerlegen das Dreieck in gewisse Teile. Sei

   $$\gamma_i := \#\text{ Randteile, die von der } i\text{-ten Gerade mit berandet werden.}$$

   Dann gilt $\sum \gamma_i = O(k)$. Beweis als Aufgabe

2. Jedes $\triangle_i$ wird möglicherweise in mehrere Teile zerlegt. Jedes innere Teil zerlegt ein Gebiet der binären Raumzerlegung und wird im Algorithmus nicht mehr unterteilt. Bei weiterer Zerlegung durch ein $\triangle_j$ werden also nur Randteile von $\triangle_i$ zerlegt. Die Zahl dieser Teile sei y_{ij} .
   Also

$$\begin{aligned} E\,[\text{Zerlegungsgröße}] &:= E\left[\sum_{i=1}^{n}\sum_{j\neq i} y_{ij}\right] \\ &= \sum_{i=1}^{n} E\left[\sum_{j\neq i} y_{ij}\right] \\ &= \sum_{i=1}^{n} O(n) = O(n^2). \end{aligned}$$

Kapitel 15

Konvexe Hülle

15.1 Grundbegriffe

Geg.: $P := \{\mathbb{p}_1, \ldots, \mathbb{p}_n\} \subset \mathbb{R}^3$

Ges.: $\langle P \rangle :=$ konvexe Hülle von P

Annahme

Je vier Punkte $\mathbb{p}_i$ seien unabhängig, d.h. $\langle P \rangle$ ist ein Polyeder mit dreiseitigen Facetten.

Euler-Formel

Für einen planaren Graphen sei

$$\begin{aligned} v &:= \#\text{ Knoten} \\ e &:= \#\text{ Kanten} \\ f &:= \#\text{ Facetten} \end{aligned}$$

Dann gilt

$$v + f - e = 2$$

Für Dreiecksnetze gilt außerdem $3f = 2e$. Eingesetzt in Euler-Formel ergibt sich

$$\begin{aligned} v - e/3 = 2 \quad &\Rightarrow \quad e = 3v - 6 \\ v - f/2 = 2 \quad &\Rightarrow \quad f = 2v - 4 \end{aligned}$$

Bemerkung

Jeder planare Graph kann durch weitere Kanten in ein Dreiecksnetz überführt werden. Also gilt allgemein für planare Graphen:

$$e \leq 3v - 6 \quad \text{und} \quad f \leq 2v - 4$$

15.2 Duale Graphen

Eine Ebene im $\mathbb{R}^3$ hat die Gleichung

$$u_1 x + u_2 y + u_3 z + u_0 = 0$$

$\mathfrak{u} := \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix}$ sind die *Ebenenkoordinaten.*
Sie sind *homogen*, weil $\gamma \cdot \mathfrak{u}$ für alle $\gamma \neq 0$ dieselbe Ebene beschreibt.

$$\mathfrak{x} = \xi \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \xi \neq 0,$$

sind die homogenen Koordinaten des Punktes $\mathfrak{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$.
In homogenen Koordinaten hat eine Ebene die Darstellung

$$\mathfrak{u}^t \mathfrak{x} = 0.$$

Ebenen und Punkte mit denselben Koordinaten heißen *dual* zueinander.

Folgerung

Die durch die Punkte $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$ aufgespannte Ebene und der Schnittpunkt der Ebenen $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$ sind dual zueinander.

Beweis

$$\mathfrak{u}^t [\mathfrak{a}\mathfrak{b}\mathfrak{c}] = [000] \quad \Leftrightarrow \quad [\mathfrak{a}\mathfrak{b}\mathfrak{c}]^t \mathfrak{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

Alternierendes Produkt

Zu $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$ gibt es wegen des Laplaceschen Entwicklungssatzes ein $\mathfrak{u}$, so daß für alle $\mathfrak{v}$ gilt

$$\det[\mathfrak{a}\mathfrak{b}\mathfrak{c}\mathfrak{v}] = \mathfrak{u}^t \mathfrak{v}$$

$\mathfrak{u}$ heißt alternierendes Produkt von $\mathfrak{a}, \mathfrak{b}$ und $\mathfrak{c}$, geschrieben

$$\mathfrak{u} = \mathfrak{a} \wedge \mathfrak{b} \wedge \mathfrak{c}$$

Weil $\mathfrak{u}^t \mathfrak{a} = \mathfrak{u}^t \mathfrak{b} = \mathfrak{u}^t \mathfrak{c} = 0$ repräsentiert $\mathfrak{u}$ die Ebene, die durch $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$ aufgespannt wird bzw. den gemeinsamen Schnittpunkt der Ebenen $\mathfrak{a}, \mathfrak{b}, \mathfrak{c}$.

duale Paare	
Punkt	Ebene
Verbindung von n Punkten	Schnitt der n dualen Ebenen
Kante	Kante
Ecken, Kanten, Facetten eines Polyeders	Facetten, Kanten Punkte eines Polyeders

Definition

$\mathfrak{u}_+ := \{\mathfrak{x} \mid \mathfrak{u}^t\mathfrak{x} \geq 0 \text{ und } [0001]\mathfrak{x} \geq 0\}$ ist ein Halbraum zur Ebene $\mathfrak{u}$.
Dann gilt:

$$\langle P\rangle \bigcap_{\mathfrak{u}_+ \in \langle P\rangle H} \mathfrak{u}_+ = \bigcup_{\mathfrak{p}\in\langle P\rangle} \mathfrak{p}$$

Dual dazu gilt

$$\langle P\rangle * = \bigcup_{\mathfrak{u}\in H} \mathfrak{u} = \bigcap_{\mathfrak{p}\in\langle P\rangle} \mathfrak{p}_+$$

Seien $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$ Punkte im $\mathbb{R}^3$, deren homogene Koordinaten so gewählt seien, daß $[0001]P \geq [0 \ldots 0]$, wobei $P := [\mathfrak{p}_1 \ldots \mathfrak{p}_n]$. Die *konvexe Hülle* von P ist

$$\langle P\rangle := \{\sum_{i=1}^{n} \mathfrak{p}_i \varrho_i \mid \varrho_i \geq 0, \sum \varrho_i > 0\}.$$

Im folgenden benutzen wir die Abkürzung $K := \langle P\rangle$. *Dual* zu K ist die *Polarmenge*

$$K^* := \{\mathfrak{u} \mid \mathfrak{u}^t \cdot K \geq 0\},$$

für die wir die Abkürzung $H := K^*$ benutzen. In Worten: K^* besteht aus allen Halbräumen, die alle Punkte in K enthalten. K besteht aus allen Punkten, die in allen Halbräumen in K^* enthalten sind. D.h. H und K sind jeweils Lösung der Ungleichung $H^tK \geq 0$, wobei K oder H fest sind.

Folgerung

H ist konvex

Weiter

E, die Menge der Ecken von K, ist die kleinste Menge mit der Eigenschaft

$$\langle E\rangle = K.$$

Außerdem ist

$$E^tH \geq 0$$

das kleinste Gleichungssystem für H. D.h. E repräsentiert die Menge der Facettenebenen von H, wenn H als Punktmenge aufgefaßt wird.

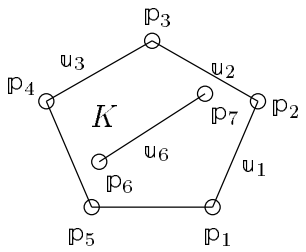


Abbildung 15.1: Die konvexe Hülle einer Punktmenge

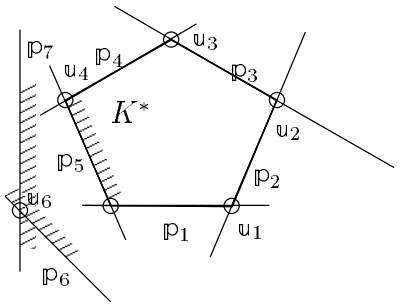


Abbildung 15.2: Die konvexe Hülle einer Menge von Halbebenen

Folgerung

Die Ecken, Kanten, Facetten von K sind dual zu den Facetten, Kanten und Ecken von H.

Bemerkung

Da $[0001]K \geq 0$, enthalten die durch K repräsentierten Halbräume alle den Ursprung.

15.3 Ein stochastischer inkrementeller Algorithmus

Geg.: $\mathbb{p}_1, \ldots, \mathbb{p}_n$, wobei $[0001][\mathbb{p}_1 \ldots \mathbb{p}_n] \geq 0$.

Ges.: $\langle \mathbb{p}_1 \ldots \mathbb{p}_n \rangle$

Werden die $\mathbb{p}_i$ als Punkte aufgefaßt, repräsentiert die konvexe Hülle $\langle \mathbb{p}_1 \ldots \mathbb{p}_n \rangle$ eine Punktmenge K. Abbildung 15.1 zeigt ein Beispiel im $\mathbb{R}^2$. Werden die $\mathbb{p}_i$ als Halbräume aufgefaßt, repräsentiert die konvexe Hülle $\langle \mathbb{p}_1 \ldots \mathbb{p}_n \rangle$ die Menge K^* der Halbräume, die K enthalten. Abbildung 15.2 zeigt das zu Abbildung 15.1 duale Bild.

Für den folgenden Algorithmus betrachten wir die $\mathbb{p}_i$ als Halbräume, deren Schnitt wir bestimmen.

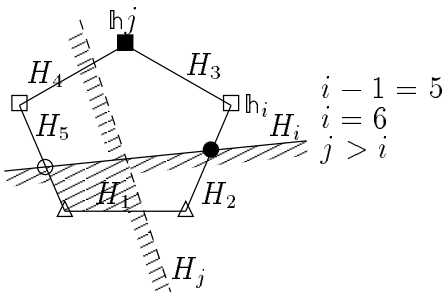


Abbildung 15.3: Illustration im $\mathbb{R}^2$

Annahme

Je vier Halbräume sind unabhängig, d.h. in jeder Ecke des Schnitts treffen sich genau 3 Facetten.

Algorithmus

1. Wähle eine zufällige Permutation $H_1, \ldots, H_n$ der Halbräume $\mathbb{p}_i$.
2. Für $i = 4, \ldots, n$ berechne $E_i = \{\text{Ecken von } H_1 \cap \cdots \cap H_i\}$. Zu jeder Ecke speichere Verweise auf die 3 Nachbarn.

Schritt 2 im Detail

a) Berechne E_4

b) $\forall\, j > 4$ verknüpfe H_j bidirektional mit einer Ecke $\mathbb{h}_j$ in $E_4 \setminus H_j$. Falls $E_i \setminus H_j = \emptyset$, dann entferne H_j

c) Für $i = 5, \ldots, n$

- Bilde E_i aus E_{i-1}, indem $\mathbb{h}_i$ und sukzessiv alle benachbarten Ecken in $E_{i-1} \setminus H_i$ entfernt werden.
- Ist ein Nachbarknoten in $H_i \cap E_{i-1}$, schneide die Kante mit Rand H_i und füge den Schnittpunkt in E_i hinzu.
- Verknüpfe alle H_j, die mit einem entfernten Knoten verknüpft sind, neu mit einem Knoten in $E_i \setminus E_{i-1} \setminus H_j$ oder entferne H_j.

Abbildung 15.3 illustriert das Vorgehen in Schritt 2c) für $i = 6$. Die Punktmengen, auf die es ankommt, sind dabei

$$\begin{aligned} E_{i-1} &= \{\triangle, \triangle, \square, \square, \blacksquare\} \\ E_i &= \{\triangle, \triangle, \circ, \bullet\} \\ E_{i-1} \setminus E_i &= \{\square, \blacksquare, \square\} \\ E_i \setminus E_{i-1} &= \{\circ, \bullet\} \end{aligned}$$

War H_i mit $\mathbb{h}_j \in E_{i-1} \setminus E_i$ verknüpft, wird H_j neu mit $\bullet \in E_i \setminus E_{i-1} \setminus H_j$ verknüpft.

Bemerkung

Wenn $E_i \setminus E_{i-1} \setminus H_j = \emptyset$, darf H_j entfernt werden. Denn H_i zerlegt $\langle E_i \rangle$ in zwei Teile und H_j einen oder beide dieser Teile. Das führt auf 3×2 Fälle:

(i) H_j zerlegt nur $\langle E_{i-1} \setminus E_i \rangle$

- $H_j \supset E_i \setminus E_{i-1} \Rightarrow H_j$ entfernbar
- $\overline{H}_j \supset E_i \setminus E_{i-1} \Rightarrow E_i \setminus E_{i-1} \setminus H_j \neq \emptyset$

(ii) H_j zerlegt nur $\langle E_i \rangle$

- $H_j \supset E_i \setminus E_{i-1}$
  $\Rightarrow$ H_j ist mit einem Knoten verknüpft, der nicht entfernt wurde
- $\overline{H}_j \supset E_i \setminus E_{i-1} \quad \Rightarrow \quad E_i \setminus E_{i-1} \setminus H_j \neq \emptyset$

(iii) H_j zerlegt beide Teile $\Rightarrow$
H_j zerlegt $E_i \setminus E_{i-1} \quad \Rightarrow \quad E_i \setminus E_{i-1} \setminus H_j \neq \emptyset$

15.4 Der Aufwand

Satz

Für den Algorithmus in 15.3 gilt

$$E[\text{Aufwand}] = O(n \log n)$$

Beweis

Der Aufwand ist abschätzbar durch

1. Zahl der neu geschaffenen Knoten $= \sum |E_i \setminus E_{i-1}|$. Es werden weniger Knoten entfernt als neue geschaffen. D.h. Entfernungen müssen nicht berücksichtigt werden.

2. Aufwand für Neuverknüpfungen der H_j mit einer Ecke in $E_i \setminus E_{i-1} \setminus H_j$ bzw. deren Entfernungen

Zu 1) (Rückwärtsanalyse)
E_i entsteht aus einem beliebigen E_{i-1}, da jede Ordnung der $H_1, \ldots, H_{i-1}$ gleichwahrscheinlich ist. Es gilt also

$$\begin{aligned} E[E_i \setminus E_{i-1}] &= \frac{3[E_i]}{\#\,\text{Facetten}} \\ &\overset{\text{s.u.}}{\leq} \frac{12i}{i} = 12, \end{aligned}$$

denn sei v, f, v^*, f^* die Zahl der Ecken und Facetten in $H_1 \cap \cdots \cap H_i = \langle E_i \rangle$ bzw. in $\langle E_i \rangle^*$.
Dann folgt aus 15.1 $v = f^* \leq 4v^* = 4f = 4i$ und daraus obige Ungleichung. Insgesamt erhalten wir für den erwarteten Aufwand

$$E[\sum_{i=5}^{n} |E_i \setminus E_i|] \leq 12n.$$

Zu 2)
Aufwand für Neuverknüpfung von H_j = Anzahl der zu durchsuchenden Knoten

$$\leq 3 \cdot \left| (\underbrace{E_{i-1} \setminus E_i}_{\text{entf.}} \cup \underbrace{E_i \setminus E_{i-1}}_{\text{neu}}) \setminus H_j \right|,$$

wobei 3 die Zahl der Nachbarn abschätzt, die untersucht werden, aber in H_j liegen. Dies kann weiiter nach oben abgeschätzt werden durch

$$6 \cdot |E_i \setminus E_{i-1} \setminus H_j|,$$

da es genügt, die neuen Knoten doppelt zu betrachten.
Da E_i aus einem beliebigem E_{i-1} entsteht (Dieser Schluß wird Rückwärtsanalyse genannt), erhalten wir für den Erwartungswert des letzten Terms

$$\begin{aligned} E[|E_i \setminus E_{i-1} \setminus H_j|] &= E[|\text{beliebige Facette } E_i \setminus H_j|] \\ &= \frac{1}{i} \sum_{\text{alle Facetten}} |\text{Facette } \setminus H_j| \\ &= \frac{3}{i} \left| \underbrace{E_i \setminus H_j}_{c_{ij}} \right| \end{aligned}$$

Somit ergibt sich für die erwartete Zahl der Neuverknüpfungen die Abschätzung

$$\begin{aligned} E[\text{Neuverknüpfungen}] &\leq \sum_i \sum_{j>i} \frac{3}{i} c_{ij} \\ &= 3 \sum \frac{n-i}{i} \underbrace{\frac{1}{n-i} \sum_{j>i} c_{ij}}_{= E[H \text{ entf. Knoten in Schritt } i+1]} \\ &\leq 36 \sum_i \frac{n-i}{i} \\ &\leq 36 \sum_i \frac{n}{i} \leq 36 n \log n. \end{aligned}$$

Kapitel 16

Sätze von Yao und Adleman

Ziel dieses Kapitels ist ein Vergleich stochastischer mit deterministischen Algorithmen und die Abschätzung der Laufzeit.

16.1 Spieltheorie

A und B seien zwei Spieler, die beide eine der drei Optionen Stein, Schere, Papier wählen können. Je nach Kombination dieser beiden Spielerentscheidungen gewinnt oder verliert A einen oder keinen Punkt. Dies ist in der folgenden Gewinnmatrix für A angegeben:

$A \backslash B$	Stein	Schere	Papier
Stein	0	1	-1
Schere	-1	0	1
Papier	1	-1	0

Das Spiel ist ein *2-Personen-Nullsummenspiel*, weil

$$\text{Gewinn von } A = \text{Verlust von } B$$

bzw.

$$\text{Gewinn von } A + \text{Gewinn von } B = 0.$$

Jede $m \times n$ Matrix $M = [m_{ij}]$ stellt ein 2-Personen-Nullsummenspiel dar.
Die Zeilen entsprechen den *Strategien* von A, die Spalten den *Strategien* von B.
Zeile i ist eine optimale Strategie von A, falls $\min m_{ij}$ maximal ist. Spalte j ist eine optimale Strategie für B, falls $\max\limits_i m_{ij}$ minimal ist. Also folgt

$$\max_i \min_j m_{ij} \underset{\substack{\uparrow \\ i \text{ optimal} \\ \text{für } A}}{\leq} \text{Gewinn}_A \underset{\substack{\uparrow \\ j \text{ optimal} \\ \text{für } B}}{\leq} \min_j \max_i m_{ij}$$

Bemerkung

Im Beispiel ist $\max\limits_i \min\limits_j m_{ij} = -1$ und $\min\limits_j \max\limits_i m_{ij} = 1$.

Definition

Falls

$$W := \max_i \min_j m_{ij} = \min_j \max_i m_{ij},$$

heißt W der Wert des Spiels und ein Paar (i, j) optimaler Strategien eine *Lösung*.

Beispiel

1. Im obigen Beispiel gibt es keine Lösung
2. Für $M = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & 1 & 0 \end{bmatrix}$ ist der Wert $W = 0$ und $(1,1)$ eine Lösung. Aber $(2,2)$ und $(3,3)$ sind keine Lösungen, obwohl $m_{22} = m_{33} = 0$.

Definition

Eine *gemischte Strategie* von A ist eine zufällige Wahl nach einer Wahrscheinlichkeitsverteilung.

$$\mathbb{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}, \text{ wobei } 0 \le p_i \text{ und } \sum p_i = 1.$$

Sei analog $\mathbb{q} = \begin{bmatrix} q_1 \\ \vdots \\ q_n \end{bmatrix}$ eine Wahrscheinlichkeitsverteilung für eine gemischte Strategie von B. Dann gilt

$$E[\text{Gewinn von } A] = \sum p_i m_{ij} q_j = \mathbb{q}^t M \mathbb{q}.$$

Eine Verteilung $\mathbb{p}$ ist optimal für A, falls

$$\min_{\mathbb{q}} \mathbb{p}^t M \mathbb{q}$$

maximal und eine Verteilung $\mathbb{q}$ ist optimal für B, falls

$$\max_{\mathbb{p}} \mathbb{p}^t M \mathbb{q}$$

minimal ist. Offenbar gilt

$$\max_{\mathbb{p}} \min_{\mathbb{q}} \mathbb{p}^t M \mathbb{q} \quad \underset{\substack{\uparrow \\ \mathbb{p} \text{ optimal} \\ \text{für } A}}{\le} \quad E[\text{Gewinn von } A] \quad \underset{\substack{\uparrow \\ \mathbb{q} \text{ optimal} \\ \text{für } B}}{\le} \quad \min_{\mathbb{q}} \max_{\mathbb{p}} \mathbb{p}^t M \mathbb{q}.$$

Satz (von Neumann)

Es gilt sogar

$$\max_{\mathbb{p}} \min_{\mathbb{q}} \mathbb{p}^t M \mathbb{q} = \min_{\mathbb{q}} \max_{\mathbb{p}} \mathbb{p}^t M \mathbb{q}$$

Beweis

Die wesentliche Idee des Beweises ist, daß die Funktion $f : (\mathbb{p}, \mathbb{q}) = \mathbb{p}^+ M \mathbb{q}$ bilinear ist. Daher wird $\min_{\mathbb{q}} f$ im Sattelpunkt von f maximal und analog $\max_{\mathbb{p}} f$ im Sattelpunkt minimal. Für ein festes $\mathbb{p}$ wird $\mathbb{p}^t M \mathbb{q}$ minimal für

$$\mathbb{q} = \mathbb{e}_j = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \text{ wenn die } j\text{-te Koordinate von } \mathbb{p}^t M \text{ minimal ist.}$$

Folgerung (Loomi)

$$\max_{\mathbb{p}} \min_{j} \mathbb{p}^t M \mathbb{e}_j = \min_{\mathbb{q}} \max_{i} \mathbb{e}_i^t M \mathbb{q}$$

16.2 Yaos Technik

Anwendung von Loomis Satz auf Algorithmendesign:
A wählt eine Eingabe E_j aus einer endlichen Menge von Eingaben $\mathcal{E}$.
B wählt einen Algorithmus A_i aus einer endlichen Menge deterministischer Algorithmen $\mathcal{A}$.
Sei m_{ij} die Laufzeit von Algorithmus A_j bei Eingabe E_i

Korollar (Yao)

Mittlere Laufzeit des schnellsten deterministischen Algorithmus A_j zu einer gegebenen Eingabeverteilung im ungünstigsten Fall = mittlere Laufzeit des schnellsten Las-Vegas-Algorithmus für die ungünstigste Eingabeverteilung.

Korollar (Yao)

Mittlere Laufzeit des schnellsten deterministischen Algorithmus A_j zu einer gegebenen Eingabeverteilung $\leq$ mittlere Laufzeit eines Las-Vegas-Algorithmus für die ungünstigste Eingabeverteilung.

Beispiel

Gegeben sei ein Las-Vegas-Algorithmus, der L Bits einer gegebenen Folge von n Bits anschaut, um zu entscheiden, ob 3 Einsen in der Bitfolge hintereinanderstehen. Mit Hilfe von Yaos Korollar läßt sich L folgendermaßen abschätzen:

- $L \geq \frac{n}{3}$, denn zur Eingabe $0 \ldots 0$ muß der schnellste deterministische Algorithmus jedes 3. Bit anschauen
- $L \geq \frac{n}{2}$, denn zur Eingabe $0101 \ldots 01$ muß der schnellste deterministische Algorithmus jedes 2. Bit anschauen

16.3 Satz von Adleman

Eine stochastische Boolesche Formel ist eine Boolesche Formel $F(x_1, \ldots, x_n, y_1, \ldots, y_m)$ mit den Zufallsvariablen $y_1, \ldots, y_m$.
F *berechnet* die Funktion $f(x_1, \ldots, x_n)$:

$$\Pr[F(x_1, \ldots, x_n, y_1, \ldots, y_m) = 1] \quad \begin{cases} \geq \frac{1}{2} & \text{falls } f(x_1, \ldots, x_n) = 1 \\ = 0 & \text{sonst} \end{cases}$$

Die Größe von F, kurz $|F|$, ist die Zahl der Operatoren $\wedge, \vee, \neg$. Sie entspricht der Auswertungszeit.
Eine Folge

$$B = (B_n(x_1 \ldots x_n, \ldots))_{n=1}^{\infty}$$

(stochastischer) Boolescher Formeln berechnet eine Funktion

$$f : \{0,1\}^* \to \{0,1\}$$

falls für alle n die Formel B_n die Restriktion $f\,|_{\{0,1\}^n}$ berechnet. B heißt von *polynomialer Größe*, falls es ein Polynom p gibt mit

$$|B_n| \leq p(n).$$

Satz (Adleman)

Wird f durch eine stochastische Folge Boolescher Formeln B_n polynomialer Größe berechnet, dann wird f auch durch eine Folge deterministischer Boolescher Formeln polynomialer Größe berechnet.

Beweis

B_n habe die stochastischen Variablen $y_1, \ldots, y_n$.
$x_1, \ldots, x_n$ stellt ein $x \in \{0, \ldots, 2^n - 1\}$ binär dar.
$y_1, \ldots, y_m$ stellt ein $y \in \{0, \ldots, 2^m - 1\}$ binär dar.
Also ist

$$M = [B_n(x, y)]_{x,y}$$

eine $2^n \times 2^n$ Matrix über $\{0,1\}$.

- Jede Zeile enthält mindestens zur Hälfte Einsen. Also gibt es eine Spalte, $y = r_1, \ldots, r_m$, die mindestens zur Hälfte aus Einsen besteht.

  $$C_1(x_1 \ldots, x_n) := B_n(x_1, \ldots, x_n, r_1, \ldots, r_m)$$

  ist eine deterministische Formel.

- Entferne alle Zeilen x, die in Spalte $y = r_1, \ldots, r_m$ eine Eins besitzen und entferne Spalte y. Dann haben die Zeilen von M mehr als zur Hälfte Einsen. Also gibt es eine Spalte, die zu mehr als der Hälfte aus Einsen besteht. Konstruiere $C_2(x_1, \ldots, x_n)$ analog zu oben.

- Mache so weiter, bis nach s Schritten alle Zeilen entfernt sind, $s \leq n$.
- Offenbar berechnet

$$C_1 \wedge \cdots \wedge C_s$$

die Funktion $f(x_1, \ldots, x_n)$.

$\Rightarrow$ Die Größe von $C_1 \wedge \cdots \wedge C_s$ ist kleiner als $s \cdot p(n) \leq n \cdot p(n)$.

Bemerkung

Die im Beweis konstruierten deterministischen Formeln sind im allgemeinen nicht ähnlich (uniform), auch wenn die stochastischen Formeln ähnlich waren.

Definition

Sei $\mathcal{A}$ ein Alphabet und $r : \mathbb{N}_0 \to \mathcal{A}^*$. Eine Turingmaschine M *benutzt den Rat* r, um $\alpha \in \mathcal{L}(M)$ zu entscheiden, wenn sie dabei auf $r(|\alpha|)$ zurückgreift. D.h. M ist ein gewisser Interpreter. Eine Turingmaschine heißt *uniform*, falls sie keinen Rat benutzt.

Definition

$$\begin{aligned} \text{P/poly} \quad := \quad & \{\mathcal{L}(M) \mid M \text{ ist nicht uniforme Turingmaschine,} \\ & \text{wobei } r(n) \text{ und } t_M \text{ polynomial beschränkt sind}\} \end{aligned}$$

Analog ist RP/poly definiert.

Folgerung

Eine Sprache $\mathcal{L}$ über $\{0, 1\}$ ist in RP/poly genau dann, wenn die charakteristische Funktion $\chi_{\mathcal{L}}$ durch eine Folge stochastischer Boolescher Formeln polynomialer Größe berechnet wird.

Folgerung (aus Adlemans Satz)

RP/poly $\subset$ P/poly

Folgerung

RP $\subset$ P/poly

Teil V

Schluß

Kapitel 17

Lindenmayersysteme

Definition

Ein *deterministisches Lindenmayer-System* L besteht aus einem Alphabet $\mathcal{A}$, Produktionen $\in (\mathcal{A}, \mathcal{A}^*)$ und einem Anfangswort $\alpha \in \mathcal{A}^+$.
Dabei gibt es zu jedem $a \in \mathcal{A}$ genau eine Produktion $a \to \beta$.

Direkte Ableitung

Ein Wort $\beta_1, \ldots, \beta_n$ heißt *direkt* aus $a_1, \ldots, a_n$ *ableitbar*, falls $a_i \to \beta_i$, $i = 1, \ldots, n$, Produktionen des Lindenmayer-Systems sind. Wir notieren dies kurz mit

$$a_1 \ldots a_n \quad \Rightarrow \quad \beta_1 \ldots \beta_n.$$

Erzeugte Sprache

$\mathcal{L}(L) := \{\omega \in \mathcal{A}^* \mid \alpha \stackrel{*}{\Rightarrow} \omega\}$

Beispiel

Die Produktionen

$$\begin{aligned} a &\to ac \\ b &\to a \\ c &\to c \end{aligned}$$

und das Anfangswort $\alpha = a$ bilden ein deterministisches Lindenmayer-System (im allgemeinen werden Produktionen $c \to c$ nicht mit aufgeführt).

Anwendung

Lindenmayer-Systeme können eingesetzt werden, um ansprechende Graphiken zu erzeugen. Die in Abbildung 17.1 gezeigte Graphik wird durch folgendes Wort über dem Alphabet $\mathcal{A} := \{F, +, -, [,]\}$ dargestellt:

$$F[+F]F[-F]F[+F]F$$

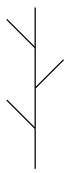


Abbildung 17.1: Graphische Darstellung eines Wortes über $\mathcal{A}$

Die Zeichen des Alphabets haben dabei die folgende Bedeutung

$$
\begin{array}{rcl}
F & : & \text{verschiebe die aktuelle Position um } \begin{bmatrix} \cos\varphi \\ \sin\varphi \end{bmatrix} \text{ und} \\
 & & \text{zeichne die Verschiebungsstrecke} \\
+ & : & \text{Setze } \varphi := \varphi + \delta \\
- & : & \text{Setze } \varphi := \varphi - \delta \\
[& : & \text{Speichere aktuelle Position und } \varphi \\
] & : & \text{Gehe zur letzten gespeicherten Position und} \\
 & & \text{ersetze } \varphi \text{ durch den zuletzt gespeicherten Wert}
\end{array}
$$

Bemerkung

Weitere mit Lindenmayer-Systemen erzeugte Bilder findet man unter der Adresse `http://www.greenworks.de/`.

Kapitel 18

Fraktale

18.1 Kontraktionen

Sei (X, δ) ein vollständiger metrischer Raum, d.h. jede Cauchy-Folge in X konvergiere gegen ein $x \in X$. Ferner sei $\mathcal{K} := \{\text{kompakte Teilmengen von } X\}$

Definition

Seien $A, B \in \mathcal{K}$, dann ist

$$d(A,B) := \max_{\mathbf{a}\in A} \min_{\mathbf{b}\in B} \delta(\mathbf{a},\mathbf{b})$$

der Abstand von A zu B (siehe Abbildung 18.1) und

$$h(A,B) := \max\{d(A,B), d(B,A)\}$$

der *Hausdorff-Abstand* zwischen A und B.

Bemerkung

$(\mathcal{K}, h)$ ist ein vollständiger metrischer Raum.

Definition

Eine Transformation $f : X \to X$ heißt *Kontraktion*, falls

$$\exists s \in [0,1)\, \forall x, y \in X : \delta(f(x), f(y)) \leq s \cdot \delta(x,y)$$

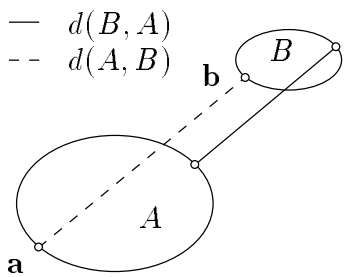


Abbildung 18.1: $d(A,B) = h(A,B)$

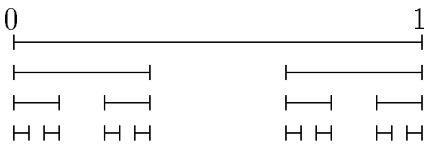


Abbildung 18.2: Cantor-Menge

Folgerung

Eine Kontraktion hat genau einen Fixpunkt $x \in X$ und

$$\forall y \in X : \lim_{n \to \infty} f^n(y) = x$$

18.2 Iterierte Funktionensysteme

$f_1, \ldots, f_n$ seien Kontraktionen auf $\mathcal{K}$ mit den Kontraktionsfaktoren $s_1, \ldots, s_n$. Die Transformation $f : \mathcal{K} \to \mathcal{K}$,

$$f(A) := f_1(A) \cup \ldots \cup f_n(A),$$

heißt *iteriertes Funktionensystem*, kurz $f := IFS(f_1, \ldots, f_n)$.

Satz

f ist eine Kontraktion mit Kontraktionsfaktor $s = \max_{i=1}^{n} s_i$.

Definition

Der Fixpunkt von f heißt *Attraktor*.

Beispiel (Cantor-Menge)

Sei

$$\begin{aligned} X &:= [0,1], \\ f_1(x) &:= \frac{1}{3}x, \\ f_2(x) &:= \frac{1}{3}x + \frac{2}{3}. \end{aligned}$$

Dann hat das iterierte Funktionensystem $IFS(f_1, f_2)$ die Cantor-Menge als Attraktor, siehe Abbildung 18.2.

Kollagentheorem

Sei f ein IFS mit Kontraktionsfaktor s. Dann gilt für alle $K \in \mathcal{K}$:

$$h(K, A) \leq \frac{1}{1-s} h(K, K_1),$$

wobei

$$K_n := f^n(K) \text{ und } A := \lim_{n \to \infty} K_n.$$

Beweis

$$\begin{aligned} h(K,A) &\leq h(K,K_1) + h(K_1,K_2) + h(K_2,K_3) + \cdots \\ &\leq h(K,K_1)(1 + s + s^2 + \ldots) \\ &= \frac{1}{1-s} h(K,K_1) \end{aligned}$$

Folgerung

Ein IFS, das K auf eine ähnliche Menge K_1 abbildet, d.h. $h(K,K_1) \leq \epsilon$, ist ein geeignetes IFS zur Erzeugung von K.

Beispiele

für „attraktive" Attraktoren iterierter Funktionensysteme findet man unter anderem in Halling, Möller: Mathematik fürs Auge, Spektrum Akademischer Verlag 1995 oder Steller: Computer und Kunst, BI 1992.