

Einführung in die Theoretische Informatik

Tobias Nipkow + Helmut Seidl

Fakultät für Informatik
TU München

<http://theo.in.tum.de>

Sommersemester 2013

© T. Nipkow / H. Seidl

1

Kapitel 0 Organisatorisches

Siehe

<http://www2.in.tum.de/hp/Main?nid=220>

2

- Vorkenntnisse:
 - Einführung in die Informatik 1/2
 - Diskrete Strukturen
- Weiterführende Vorlesungen:
 - Automaten, Formale Sprachen, Berechenbarkeit und Entscheidbarkeit
 - Komplexitätstheorie
 - Logik
 - Compilerbau
 - ...

3

1. Themen und Ziel der Vorlesung

Themengebiete:

- Berechenbarkeitstheorie
 - Untersuchung der Grenzen, was Rechner prinzipiell können
- Komplexitätstheorie
 - Untersuchung der Grenzen, was Rechner mit begrenzten Ressourcen können
- Automatentheorie
 - Rechner mit endlichem oder kellerartigem Speicher
- Grammatiken
 - Syntax von Programmiersprachen

4

Geschichte:

- 1936 Berechenbarkeitstheorie: Church & Turing
- 1956 Automaten und reguläre Ausdrücke: Kleene
- 1956 Grammatiken: Chomsky
- 1971 Komplexitätstheorie: Cook

5

Gliederung (1. & 2. Semesterhälfte):

- ① Automaten und formale Sprachen
 - Endliche Automaten und reguläre Ausdrücke
(rm *.pdf)
 - Kontextfreie Grammatiken
($\langle A \rangle ::= \langle A \rangle + \langle A \rangle \mid \langle A \rangle * \langle A \rangle \mid \dots$)
- ② Berechenbarkeit
 - Turing Maschinen, Berechenbarkeit, Entscheidbarkeit, Aufzählbarkeit
 - Komplexitätsklassen (P und NP)

6

Ziele:

Abstraktion von *irrelevanten* Details:

zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:

zB deterministische durch nichtdeterministische Maschinen

Äquivalenz von Formalismen:

zB endliche Automaten und reguläre Ausdrücke.

Reduktion von einem Problem auf ein anderes:

zB Formeln auf Automaten, ...

Endliche Beschreibungen unendlicher Mengen

**Informatik ist keine (schwarze) Kunst
sondern eine exakte Wissenschaft.**

7

2. Literatur



John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman.
Introduction to Automata Theory, Languages, and Computation.



Dexter Kozen.
Automata and Computability.



Katrin Erk, Lutz Priese.
Theoretische Informatik: Eine umfassende Einführung.



Uwe Schöning.
Theoretische Informatik — kurzgefasst.

8

Kapitel I Formale Sprachen und Automaten

1. Grundbegriffe

Definition 1.1

- Ein Alphabet Σ ist eine endliche Menge.
Bsp: $\{0, 1\}$, ASCII, Unicode.
 - Ein Wort/String über Σ ist eine endliche Folge von Zeichen aus Σ , zB 010.
 - Das leere Wort wird mit ϵ bezeichnet.
 - Sind u und v Wörter, so ist uv ihre Konkatenation.
 - Ist w ein Wort, so ist w^n definiert durch
 $w^0 = \epsilon$ und $w^{n+1} = ww^n$. Bsp: $(ab)^3 = ababab$.
 - $|w|$ ist die Länge eines Wortes w .
 - Σ^* ist die Menge aller Wörter über Σ .
 - Eine Teilmenge $L \subseteq \Sigma^*$ ist eine (formale) Sprache.

9

Beispiel 1.2 (Formale Sprachen)

Definition 1.3 (Operationen auf Sprachen)

Seien $A, B \subseteq \Sigma^*$.

- Konkatenation: $AB = \{uv \mid u \in A \wedge v \in B\}$
Bsp: $\{ab, b\}\{a, bb\} = \{aba, abbb, ba, bbb\}$
NB: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \dots w_n \mid w_1, \dots, w_n \in A\} = A \cdots A$
Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \dots w_n \mid n \geq 0 \wedge w_1, \dots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \dots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
Bsp: $\Sigma^+ =$ Menge aller nicht-leeren Wörter über Σ

Achtung:

- Für alle A : $\epsilon \in A^*$
- $\emptyset^* =$

11

Einige Rechenregeln:

Lemma 1.4

- $\emptyset A = \emptyset$
- $\{\epsilon\}A = A$

Lemma 1.5

- $A(B \cup C) = AB \cup AC$
- $(A \cup B)C = AC \cup BC$

Achtung: i.A. gilt $A(B \cap C) = AB \cap AC$ *nicht*.

Lemma 1.6

$$A^* A^* = A^*$$

12

Erinnerung: Eine Menge M ist **abzählbar** falls sie gleich mächtig wie eine Teilmenge von \mathbb{N} ist,

- d.h., es gibt eine Bijektion zw. M und einer Teilmenge von \mathbb{N} ,
- d.h., es gibt eine Nummerierung der Elemente von M so dass $M = \{m_0, m_1, \dots\}$.

Eine Menge ist **überabzählbar** wenn sie nicht abzählbar ist.

Lemma 1.7

Σ^* ist abzählbar (falls Σ endlich).

Beweis:

Durch Beispiel:

$$\begin{array}{rcl} \{0, 1\}^* & = & \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\} \\ \mathbb{N} & = & \{0, 1, 2, 3, 4, 5, 6, 7, \dots\} \end{array}$$

□

13

Satz 1.8

Die Menge der Sprachen über einem nicht-leeren Alphabet ist überabzählbar.

Beweis:

Indirekt. Nimm an, die Menge der Sprachen sei abzählbar.

	w_0	w_1	w_2	\dots	$= \Sigma^*$
L_0	0	1	1	\dots	
L_1	1	0	0	\dots	
L_2	1	1	1	\dots	
\vdots	\vdots	\vdots	\vdots	\ddots	

Eine Sprache L , die nicht in der Tabelle ist: \neg Diagonale!

L	1	1	0	\dots
-----	---	---	---	---------

Widerspruch!

Formal: $L := \{w_i \mid w_i \notin L_i\}$

Dann $L \neq L_i$ für alle i , da $w_i \in L \Leftrightarrow w_i \notin L_i$.

□

14

Definition 1.9

Eine Sprache $M \subseteq \Sigma^*$ ist entscheidbar gdw es einen Algorithmus gibt, der ihre **charakteristische Funktion**

$$\chi_M(x) = \begin{cases} 1 & \text{falls } x \in M \\ 0 & \text{falls } x \notin M \end{cases}$$

berechnet.

Beispiel 1.10 (Entscheidbare Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- $\{(ab)^n \mid n \in \mathbb{N}\}$
- $\{a^n b^n \mid n \in \mathbb{N}\}$
- Die Menge aller Primzahlen (dezimal kodiert)
- \emptyset
- $\{\epsilon\}$

Satz 1.11

Es gibt nicht-entscheidbare Sprachen.

Beweis:

Jeder Algorithmus ist ein Wort.

Daher gibt es nur abzählbar viele Algorithmen.

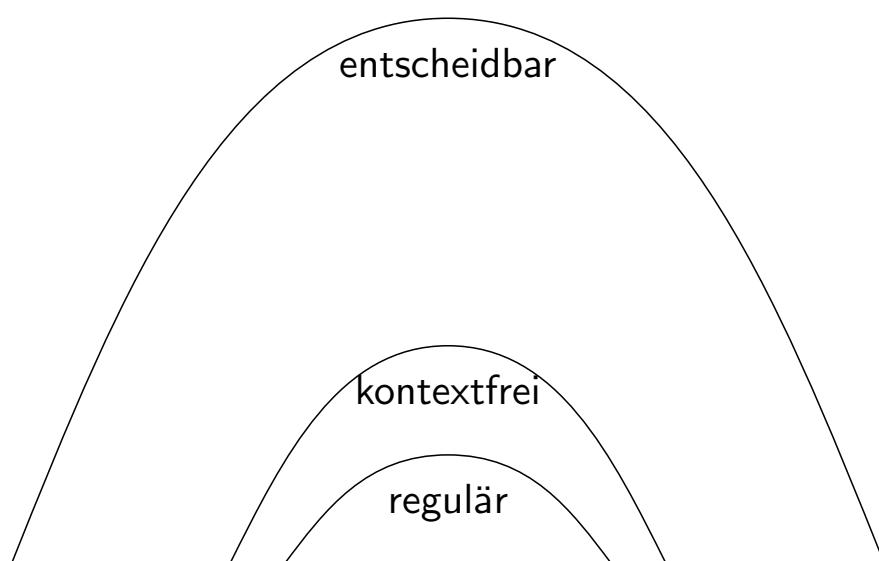
Aber überabzählbar viele Sprachen, also mehr als Algorithmen. \square

Wir haben sogar gezeigt:

- Es gibt höchstens *abzählbar viele entscheidbare Sprachen*.
- Damit verbleiben *überabzählbar viele unentscheidbare Sprachen*.

17

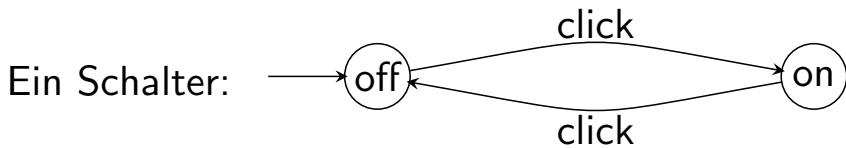
alle formalen Sprachen



18

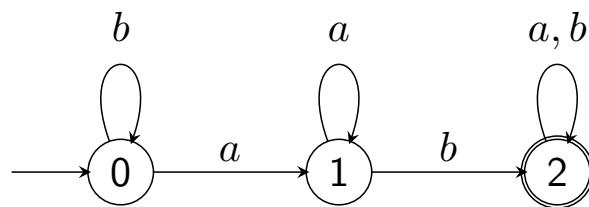
2. Reguläre Sprachen

2.1 Deterministische endliche Automaten



UML state charts sind endliche Automaten.

Ein endlicher Automat mit Endzustand:



Eingabewort $baba \rightsquigarrow$ Zustandsfolge 0,0,1,2,2.

Menge der Wörter, die vom Start- in den Endzustand führen?

Alle Wörter, die enthalten.

19

Definition 2.1

Ein **deterministischer endlicher Automat** (*deterministic finite automaton*, DFA) $M = (Q, \Sigma, \delta, q_0, F)$ besteht aus

- einer endlichen Menge von **Zuständen** Q ,
- einem (endlichen) **Eingabealphabet** Σ ,
- einer (totalen!) **Übergangsfunktion** $\delta : Q \times \Sigma \rightarrow Q$,
- einem **Startzustand** $q_0 \in Q$, und
- einer Menge von **Endzuständen** $F \subseteq Q$.

Endzustände heißen auch **akzeptierenden Zustände**.

Die von M **akzeptierte** Sprache ist

$$L(M) := \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\},$$

wobei $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ rekursiv definiert ist durch

$$\begin{aligned}\hat{\delta}(q, \epsilon) &= q \\ \hat{\delta}(q, aw) &= \hat{\delta}(\delta(q, a), w) \quad \text{für } a \in \Sigma, w \in \Sigma^*\end{aligned}$$

20

Definition 2.2

Eine Sprache ist **regulär** gdw sie von einem DFA akzeptiert wird.

Offensichtlich gilt:

Fakt 2.3

Reguläre Sprachen sind entscheidbar.

21

Bemerkungen

- Endliche Automaten können durch gerichtete und markierte **Zustandsgraphen** veranschaulicht werden:
 - Knoten $\hat{=}$ Zuständen
 - Kanten $\hat{=}$ Übergängen: $p \xrightarrow{a} q \hat{=} \delta(p, a) = q$

Der Anfangszustand wird durch einen Pfeil, Endzustände werden durch doppelte Kreise gekennzeichnet.

- Die Operation $\hat{\cdot}$ ist ein Funktional. In OCaml:

```
let rec dach f q = function
  [] -> q |
  a::w -> dach f (f q a) w
```

D.h. $dach = fold_left$!

- Die Erweiterung von f auf \hat{f} funktioniert für beliebige $f : A \times B \rightarrow A$
- Für $a \in \Sigma$ gilt $\hat{\delta}(q, a) = \hat{\delta}(q, a\epsilon) = \hat{\delta}(\delta(q, a), \epsilon) = \delta(q, a)$

22

Das Beweisprinzip Induktion über die Länge von Wörtern hat folgenden häufigen Spezialfall:

- **Basis:** Zeige die Behauptung für $w = \epsilon$
- **Induktionsannahme:** Die Behauptung stimmt für w .
Schritt: Zeige die Behauptung für aw , $a \in \Sigma$.
Alternative: Zeige die Behauptung für wa , $a \in \Sigma$.

Damit gilt die Behauptung für alle Wörter.

23

Lemma 2.4

Für $x \in \Sigma$ und $w \in \Sigma^*$ gilt $\hat{\delta}(q, wx) = \delta(\hat{\delta}(q, w), x)$

Beweis:

Mit Induktion über die Länge von w :

- **Basis:** $w = \epsilon$:

$$\hat{\delta}(q, \epsilon x) = \hat{\delta}(q, x\epsilon) = \hat{\delta}(\delta(q, x), \epsilon) = \delta(q, x) = \delta(\hat{\delta}(q, \epsilon), x)$$
- **Schritt:**

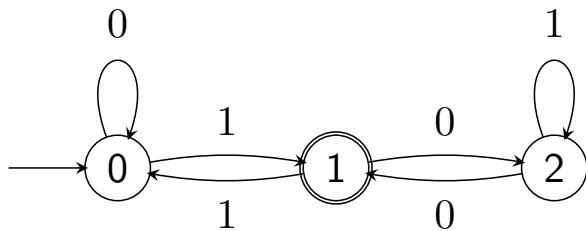
$$\begin{aligned}
 & \hat{\delta}(q, awx) \\
 &= \hat{\delta}(\delta(q, a), wx) && \text{Def. von } \hat{\delta} \\
 &= \delta(\hat{\delta}(\delta(q, a), w), x) && \text{Ind.Hyp.} \\
 &= \delta(\hat{\delta}(q, aw), x) && \text{Def. von } \hat{\delta}
 \end{aligned}$$

□

24

Beispiel 2.5

Ein DFA M :



Für $w \in \{0, 1\}^*$ sei $\#w$ die von w binär repräsentierte Zahl, zB $\#100 = 4$.

$$L(M) = \{w \mid \text{ } \}$$

Beweis:

1. $\delta(q, b) =$
2. $\hat{\delta}(0, w) =$

□

25

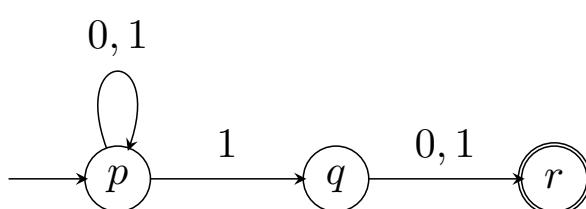
2.2 Nichtdeterministische endliche Automaten

Verallgemeinerung: $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$

$[\mathcal{P}(Q) = \text{Potenzmenge} = \text{Menge aller Teilmengen} = 2^Q]$

Beispiel 2.6

Erkennung der Binärzahlen, deren vorletzte Ziffer 1 ist:



Wort wird akzeptiert gdw es einen Weg zum Endzustand gibt.

Intuitive Vorstellung:

- Der Automat durchsucht alle möglichen Wege (parallel/sequentiell).
- Der Automat “räts” den richtigen Weg.

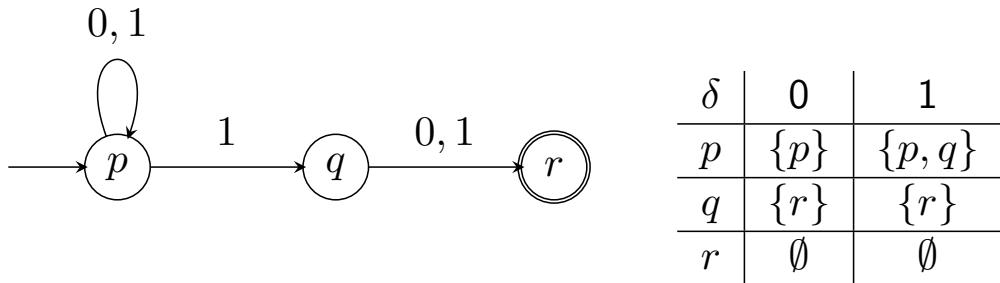
26

Definition 2.7

Ein **nichtdeterministischer endlicher Automat** (*nondeterministic finite automaton*, NFA) ist ein 5-Tupel $N = (Q, \Sigma, \delta, q_0, F)$, so dass

- Q, Σ, q_0 und F sind wie bei einem DFA
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$

Beispiel



Alternative: Relation $\delta \subseteq Q \times \Sigma \times Q$

27

Erweiterung von $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$
auf $\bar{\delta} : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$:

$$\bar{\delta}(S, a) := \bigcup_{q \in S} \delta(q, a)$$

$$\Rightarrow \hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

Intuition:

$\hat{\delta}(S, w)$ ist Menge aller Zustände,
die sich von einem Zustand in S aus mit w erreichen lassen.

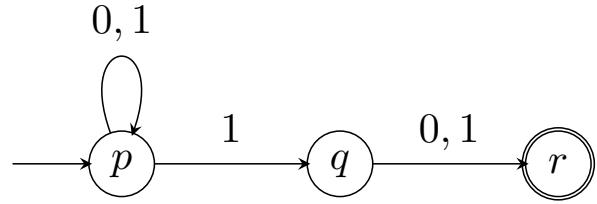
Die von $N = (Q, \Sigma, \delta, q_0, F)$ **akzeptierte** Sprache ist

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

Um Tod durch Notation zu vermeiden schreiben wir oft nur δ statt $\bar{\delta}$.

28

Beispiel



$$\hat{\delta}(\{p, q\}, 10) =$$

$$\hat{\delta}(\bar{\delta}(\{p, q\}, 1), 0) =$$

$$\hat{\delta}(\delta(p, 1) \cup \delta(q, 1), 0) =$$

$$\hat{\delta}(\{p, q, r\}, 0) =$$

$$\bar{\delta}(\{p, q, r\}, 0) =$$

$$\delta(p, 0) \cup \delta(q, 0) \cup \delta(r, 0) =$$

$$\{p\} \cup \{r\} \cup \emptyset = \{p, r\}$$

29

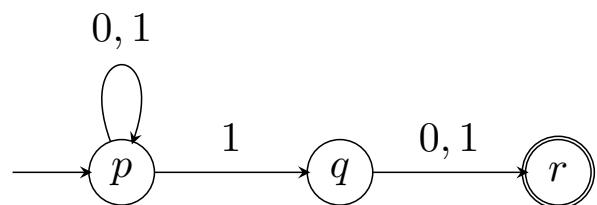
2.3 Äquivalenz von NFA und DFA

Satz 2.8

Für jede von einem NFA akzeptierte Sprache L gibt es einen DFA M mit

$$L = L(M).$$

Beispiel



30

Beweis:

Sei $N = (Q, \Sigma, \delta, q_0, F)$ ein NFA.

Definiere den DFA $M = (\mathcal{P}(Q), \Sigma, \bar{\delta}, \{q_0\}, F_M)$:

$$F_M := \{S \subseteq Q \mid S \cap F \neq \emptyset\}$$

Dann gilt:

$$\begin{aligned} w \in L(N) &\Leftrightarrow \hat{\delta}(\{q_0\}, w) \cap F \neq \emptyset \quad \text{Def.} \\ &\Leftrightarrow \hat{\delta}(\{q_0\}, w) \in F_M \quad \text{Def.} \\ &\Leftrightarrow w \in L(M) \quad \text{Def.} \end{aligned}$$

□

Dies nennt man die **Potenzmengen-** oder **Teilmengenkonstruktion**.

31

Warum NFAs?

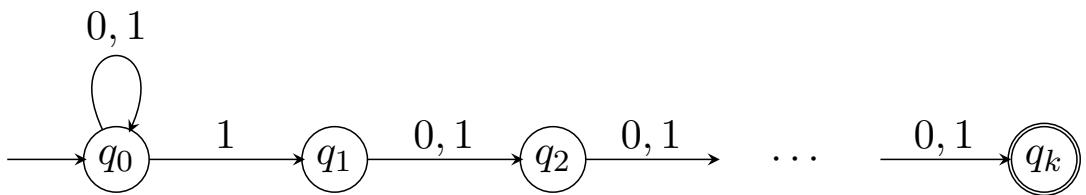
Mit NFAs lassen sich reguläre Sprachen
(u.U. exponentiell) kompakter darstellen.

32

Beispiel 2.9

$$L_k := \{w \in \{0, 1\}^* \mid \text{das } k\text{-letzte Bit von } w \text{ ist } 1\}$$

Ein NFA für diese Sprache ist gegeben durch:



Die Potenzmengenkonstruktion liefert einen DFA für L_k mit 2^{k+1} Zuständen. Geht es kompakter?

Lemma 2.10

Jeder DFA M mit $L(M) = L_k$ hat mindestens 2^k Zuständen.

Im schlimmsten Fall ist ein exponentieller Sprung unvermeidlich.

33

Beweis:

Indirekt. Sei M ein DFA mit $< 2^k$ Zuständen, so dass $L(M) = L_k$.

- Es gibt $w_1, w_2 \in \{0, 1\}^k$ mit $w_1 \neq w_2$ aber $\hat{\delta}(q_0, w_1) = \hat{\delta}(q_0, w_2)$
- Sei $w_1 = wa_i \dots a_k$ und $w_2 = wb_i \dots b_k$ mit $a_i \neq b_i$
- OE sei $a_i = 1, b_i = 0$
- $w_1 0^{i-1} = wa_i \dots a_k 0^{i-1} \in L_k$ und $w_2 0^{i-1} = wb_i \dots b_k 0^{i-1} \notin L_k$
- $\hat{\delta}(q_0, w_1 0^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_2), 0^{i-1}) = \hat{\delta}(q_0, w_2 0^{i-1}) \not\in$

□

34

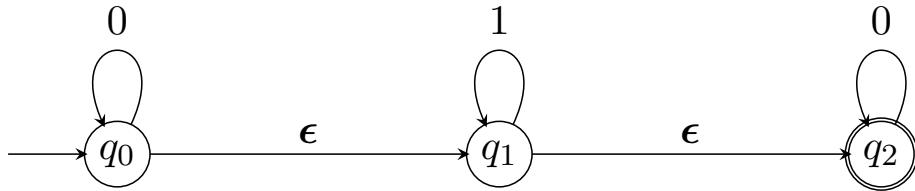
2.4 NFAs mit ϵ -Übergängen

Definition 2.11

Ein NFA mit ϵ -Übergängen (auch ϵ -NFA) ist ein NFA mit einem speziellen Symbol $\epsilon \notin \Sigma$ und mit

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \rightarrow \mathcal{P}(Q) .$$

Ein ϵ -Übergang darf ausgeführt werden, ohne dass ein Eingabeezeichen gelesen wird.



Akzeptiert: $\epsilon, 00, 11, \dots$ Nicht akzeptiert: \dots

Bemerkung: $\epsilon \neq \epsilon$; ϵ ist ein einzelnes Symbol, ϵ das leere Wort.

35

Formal betrachten wir einen ϵ -NFA $N = (Q, \Sigma, \delta, q_0, F)$ als kompakte Repräsentation eines ϵ -freien NFA $N' = (Q, \Sigma, \delta', q_0, F')$

- $\delta' : Q \times \Sigma \rightarrow \mathcal{P}(Q)$:

$$\delta'(q, a) := \bigcup_{i \geq 0, j \geq 0} \hat{\delta}(\{q\}, \epsilon^i a \epsilon^j) .$$

- Falls N das leere Wort ϵ akzeptiert, also falls

$$\exists i \geq 0. \hat{\delta}(\{q_0\}, \epsilon^i) \cap F \neq \emptyset$$

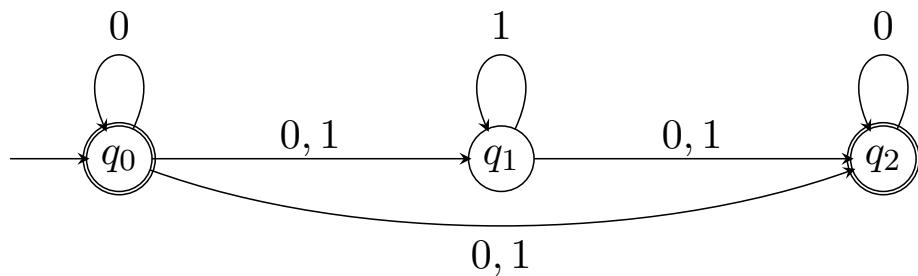
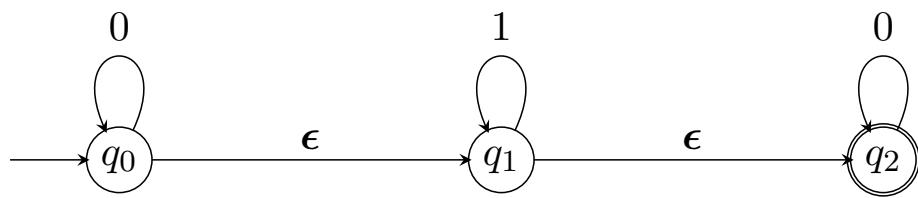
dann setze $F' := F \cup \{q_0\}$, sonst setze $F' := F$.

Damit gilt per definitionem:

Jeder ϵ -NFA ist äquivalent zu einem NFA.

36

Beispiel 2.12



37

Warum ϵ -NFAs?

Sie sind praktisch.

Ab jetzt: „ ϵ -Übergang“ und „ ϵ -NFA“

38

Vorläufiges Fazit:

Die folgenden Automatentypen sind gleich mächtig:

- DFA
- NFA
- ϵ -NFA

39

2.5 Reguläre Ausdrücke

Reguläre Ausdrücke sind eine kompakte Notation für formale Sprachen.

Definition 2.13

Reguläre Ausdrücke (*regular expressions*, REs) sind induktiv definiert:

- \emptyset ist ein regulärer Ausdruck.
- ϵ ist ein regulärer Ausdruck.
- Für jedes $a \in \Sigma$ ist a ist ein regulärer Ausdruck.
- Wenn α und β reguläre Ausdrücke sind, dann auch
 - $\alpha\beta$
 - $\alpha | \beta$ (oft $\alpha + \beta$ geschrieben)
 - α^* .

Nichts sonst ist ein regulärer Ausdruck.

40

Notation:

- Reguläre Ausdrücke können bzw. müssen geklammert werden.
- Bindungsstärke: $*$ stärker als Konkatenation stärker als $|$
- $ab^* = a(b^*) \neq (ab)^*$
- $ab | c = (ab) | c \neq a(b | c)$

41

Definition 2.14

Zu einem regulären Ausdruck γ ist die zugehörige Sprache $L(\gamma)$ rekursiv definiert:

- $L(\emptyset) = \emptyset$
- $L(\epsilon) = \{\epsilon\}$
- $L(a) = \{a\}$
- $L(\alpha\beta) = L(\alpha)L(\beta)$
- $L(\alpha | \beta) = L(\alpha) \cup L(\beta)$
- $L(\alpha^*) = L(\alpha)^*$

42

Beispiel 2.15

Sei das zugrunde liegende Alphabet $\Sigma = \{0, 1\}$.

- Alle Wörter, die mit 00 enden:

$$(0|1)^*00$$

- Alle Wörter gerader Länge, in denen 0 und 1 alternieren:

$$(01)^* \mid (10)^*$$

- Alle Wörter, die eine gerade Anzahl von 1'en enthalten:

$$(0^*10^*1)^*0^*$$

- Alle Wörter, die die Binärdarstellung einer durch 3 teilbaren Zahl darstellen, also

$$0, 11, 110, 1001, 1100, 1111, 10010, \dots$$

Hausaufgabe!

43

Beispiel 2.16

Gleitkommazahlen: $\Sigma = \{+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$

$$(+ \mid - \mid \epsilon)(DD^* \mid DD^*.D^* \mid D^*.DD^*)$$

wobei $D = (0|1|2|3|4|5|6|7|8|9)$

44

Erweiterte reguläre Ausdrücke in UNIX:

- $= a_1 | \dots | a_n$ wobei $\Sigma = \{a_1, \dots, a_n\}$
- $[a_1 \dots a_n] = a_1 | \dots | a_n$
- $[^a_1 \dots a_n] = b_1 | \dots | b_m$ wobei $\{b_1, \dots, b_m\} = \Sigma \setminus \{a_1, \dots, a_n\}$
- $\alpha? = \epsilon | \alpha$
- $\alpha^+ = \alpha \alpha^*$
- $\alpha\{n\} = \alpha \dots \alpha$ (n copies)

45

Strukturelle Induktion

Da die regulären Ausdrücke induktiv definiert sind, gilt für sie das Prinzip der **strukturellen Induktion**:

Um zu beweisen, dass Eigenschaft P für alle regulären Ausdrücke γ gilt, also $P(\gamma)$, beweise

- $P(\emptyset)$
- $P(\epsilon)$
- $P(a)$ für alle $a \in \Sigma$
- $P(\alpha) \wedge P(\beta) \Rightarrow P(\alpha\beta)$
- $P(\alpha) \wedge P(\beta) \Rightarrow P(\alpha | \beta)$
- $P(\alpha) \Rightarrow P(\alpha^*)$

Komfortabler Spezialfall der Induktion über die Länge von γ .

46

Satz 2.17 (Kleene 1956)

Eine Sprache $L \subseteq \Sigma^*$ ist genau dann durch einen regulären Ausdruck darstellbar, wenn sie regulär ist.

Beweis:

“ \implies ” :

Sei $L = L(\gamma)$.

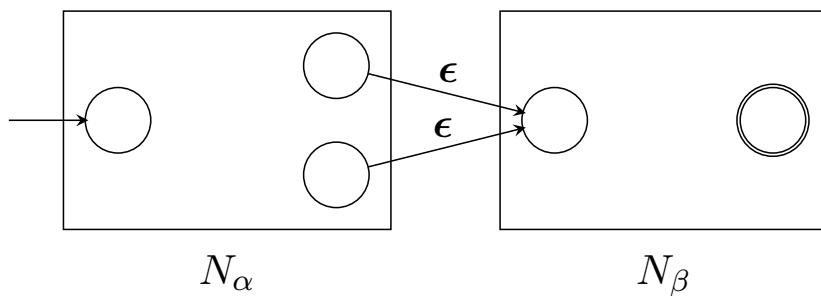
Wir konstruieren einen ϵ -NFA N mit $L = L(N)$ mit Hilfe struktureller Induktion über γ .

Die Basisfälle $\gamma = \emptyset$, $\gamma = \epsilon$, und $\gamma = a \in \Sigma$ sind offensichtlich.

47

Fall $\gamma = \alpha\beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



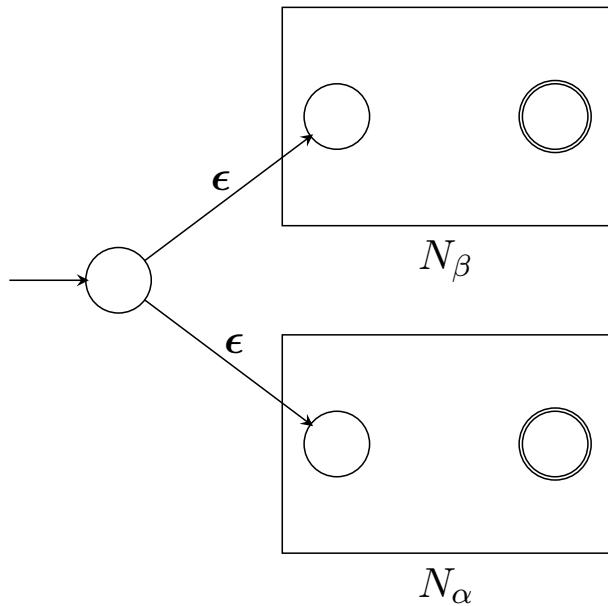
Formal:

$$\begin{aligned}
 N_\alpha &= (Q_\alpha, \Sigma, \delta_\alpha, q_{0\alpha}, F_\alpha) \\
 N_\beta &= (Q_\beta, \Sigma, \delta_\beta, q_{0\beta}, F_\beta) \quad (\text{mit } Q_\alpha \cap Q_\beta = \emptyset) \\
 N_{\alpha\beta} &:= (Q_\alpha \cup Q_\beta, \Sigma, \delta, q_{0\alpha}, F_\beta) \\
 \delta &:= \delta_\alpha \cup \delta_\beta \cup \{(f, \epsilon) \mapsto \{q_{0\beta}\} \mid f \in F_\alpha\}
 \end{aligned}$$

48

Fall $\gamma = \alpha \mid \beta$:

Nach Induktionsannahme können wir ϵ -NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.

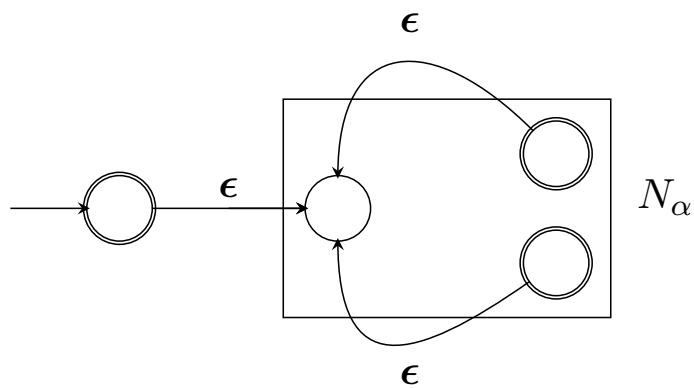


49

Fall $\gamma = \alpha^*$:

Nach Induktionsannahme können wir einen ϵ -NFA N_α konstruieren mit

$$L(N_\alpha) = L(\alpha) .$$



50

“ \Leftarrow ”:

Sei $M = (Q, \Sigma, \delta, q_1, F)$ ein DFA.

Wir konstruieren einen RE γ mit $L(M) = L(\gamma)$.

Sei $Q = \{q_1, \dots, q_n\}$. Wir setzen

$R_{ij}^k := \{w \in \Sigma^* \mid \text{die Eingabe } w \text{ führt von } q_i \text{ in } q_j, \text{ wobei alle Zwischenzustände (ohne ersten und letzten) einen Index } \leq k \text{ haben}\}$

Behauptung: Für alle $i, j \in \{1, \dots, n\}$ und $k \in \{0, \dots, n\}$ können wir einen RE α_{ij}^k konstruieren mit $L(\alpha_{ij}^k) = R_{ij}^k$.

51

Bew.:

Induktion über k :

$k = 0$: Hier gilt

$$R_{ij}^0 = \begin{cases} \{a \in \Sigma \mid \delta(q_i, a) = q_j\}, & \text{falls } i \neq j \\ \{a \in \Sigma \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\}, & \text{falls } i = j \end{cases}$$

Setze

$$\alpha_{ij}^0 := \begin{cases} a_1 | \dots | a_l & \text{falls } i \neq j \\ a_1 | \dots | a_l | \epsilon & \text{falls } i = j \end{cases}$$

wobei $\{a_1, \dots, a_l\} = \{a \in \Sigma \mid \delta(q_i, a) = q_j\}$.

52

Bew.:

Induktion über k :

$k \Rightarrow k + 1$: Hier gilt

$$R_{ij}^{k+1} = R_{ij}^k \cup R_{i(k+1)}^k (R_{(k+1)(k+1)}^k)^* R_{(k+1)j}^k$$

weil man jede Folge von Zahlen $\leq k + 1$ schreiben kann als

$$F_1, k + 1, F_2, k + 1, \dots, k + 1, F_m$$

wobei jede F_l Folge von Zahlen $\leq k$ ist.

Wir definieren rekursiv

$$\alpha_{ij}^{k+1} = \alpha_{ij}^k \mid \alpha_{i(k+1)}^k (\alpha_{(k+1)(k+1)}^k)^* \alpha_{(k+1)j}^k$$

Somit gilt

$$L(M) = L(\alpha_{1i_1}^n \mid \dots \mid \alpha_{1i_r}^n)$$

wobei $F = \{i_1, \dots, i_r\}$.

□

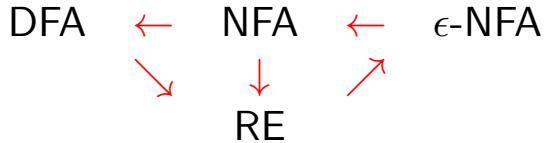
53

Fakt 2.18

Alle endlichen Sprachen sind regulär.

54

Unsere Konversionen auf einen Blick:



$\text{RE} \rightarrow \epsilon\text{-NFA}$: RE der Länge $n \rightsquigarrow O(n)$ Zustände

$\epsilon\text{-NFA} \rightarrow \text{NFA}$: $Q \rightsquigarrow Q$

$\text{NFA} \rightarrow \text{DFA}$: n Zustände $\rightsquigarrow O(2^n)$ Zustände

$\text{FA} \rightarrow \text{RE}$: n Zustände $\rightsquigarrow \text{RE}$ der Länge $O(n4^n)$

Beweis FA→RE: $m_k :=$ maximale Länge von α_{ij}^k

- $m_0 = c_1|\Sigma|$
- $m_{k+1} = 4 * m_k + c_2$
- $m_k \in O(4^k)$
- Länge des Gesamtausdrucks: $O(n4^n)$

55

2.6 Abschlusseigenschaften regulärer Sprachen

Satz 2.19

Seien $R, R_1, R_2 \subseteq \Sigma^*$ reguläre Sprachen. Dann sind auch

$$R_1 R_2, R_1 \cup R_2, R^*, \overline{R} (:= \Sigma^* \setminus R), R_1 \cap R_2, R_1 \setminus R_2$$

reguläre Sprachen.

Beweis:

$R_1 R_2, R_1 \cup R_2, R^*$ klar.

\overline{R} Sei $R = L(A)$, $A = (Q, \Sigma, \delta, q_0, F)$ DFA.

Betrachte $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$.

Dann ist $L(A') = \overline{L(A)} = \overline{R}$

$$R_1 \cap R_2 = \overline{\overline{R_1} \cup \overline{R_2}} \quad (\text{De Morgan})$$

$$R_1 \setminus R_2 =$$

□

56

Bemerkung

Komplementierung (\bar{R}) durch Vertauschen von Endzuständen und Nicht-Endzuständen funktioniert nur bei DFAs, nicht bei NFAs!

Übungsaufgabe: Finde Gegenbeispiel für NFAs

Bei NFAs:

Komplementierung erzwingt Determinierung.

Komplementierung ist (potenziell) teuer.

57

Die **Produkt-Konstruktion**:

Durchschnitt direkt auf DFAs, ohne Umweg über de Morgan.

Beide DFAs laufen synchron parallel, Wort wird akzeptiert wenn *beide* akzeptieren.

Parallelismus = Kreuzprodukt der Zustandsräume

Satz 2.20

Sind $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ und $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ DFAs, dann ist der **Produkt-Automat**

$$M := (Q_1 \times Q_2, \Sigma, \delta, (s_1, s_2), F_1 \times F_2)$$
$$\delta((q_1, q_2), a) := (\delta_1(q_1, a), \delta_2(q_2, a))$$

ein DFA der $L(M_1) \cap L(M_2)$ akzeptiert.

Erinnerung: $|Q_1 \times Q_2| = |Q_1| |Q_2|$

58

Beweis:

Mit Induktion über w :

$$\hat{\delta}((q_1, q_2), w) = (\hat{\delta}_1(q_1, w), \hat{\delta}_2(q_2, w)).$$

Damit gilt:

$$\begin{aligned} w \in L(M) &\Leftrightarrow (\hat{\delta}((s_1, s_2), w) \in F_1 \times F_2 \\ &\Leftrightarrow (\hat{\delta}_1(s_1, w), \hat{\delta}_2(s_2, w)) \in F_1 \times F_2 \\ &\Leftrightarrow \hat{\delta}_1(s_1, w) \in F_1 \wedge \hat{\delta}_2(s_2, w) \in F_2 \\ &\Leftrightarrow w \in L(M_1) \wedge w \in L(M_2) \\ &\Leftrightarrow w \in L(M_1) \cap L(M_2) \end{aligned}$$

□

Funktioniert Durchschnitt durch Produkt auch für NFAs?

59

Definition 2.21

Die **Umkehrung (Spiegelung)** von $w = a_1 \dots a_n$ ist $w^R := a_n \dots a_1$.

Die Umkehrung einer Sprache A ist $A^R := \{w^R \mid w \in A\}$

Satz 2.22

Ist A eine reguläre Sprache, dann auch A^R .

Beweis:

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein DFA mit $L(M) = A$.

Wir konstruieren nun einen ϵ -NFA M' mit $L(M') = A^R$:

- Kehre alle Übergänge um: $p \xrightarrow{a} q \rightsquigarrow p \xleftarrow{a} q$
- Füge einen neuen Startzustand q'_0 hinzu mit $q'_0 \xrightarrow{\epsilon} f$ für alle $f \in F$.
- Mache q_0 zum (einzigsten) Endzustand.

Dann gilt $L(M') = A^R$.

60

Beweis (Forts.):

Formal: $M' = (Q \cup \{q'_0\}, \Sigma, \delta', q'_0, \{q_0\})$ mit

- $q'_0 \notin Q$,
- $\delta'(q, a) = \{p \mid \delta(p, a) = q\}$
- $\delta'(q'_0, \epsilon) = F$

Dann gilt $w^R \in L(M') \Leftrightarrow w \in L(M)$.

Beweis mit Induktion über w . □

61

2.7 Rechnen mit regulären Ausdrücken

Definition 2.23

Zwei reguläre Ausdrücke sind äquivalent gdw sie die gleiche Sprache darstellen:

$$\alpha \equiv \beta \Leftrightarrow L(\alpha) = L(\beta)$$

Beispiel zum Unterschied von $=$ (syntaktische Identität) und \equiv (Bedeutungsäquivalenz): $\epsilon \equiv \emptyset^*$ aber $\epsilon \neq \emptyset^*$.

Wird leicht (bewusst oder unbewusst) verwechselt.

62

Null und Eins:

Lemma 2.24

- $\emptyset | \alpha \equiv \alpha | \emptyset \equiv \alpha$
- $\emptyset\alpha \equiv \alpha\emptyset \equiv \emptyset$
- $\epsilon\alpha \equiv \alpha\epsilon \equiv \alpha$
- $\emptyset^* \equiv \epsilon$
- $\epsilon^* \equiv \epsilon$

63

Lemma 2.25

Assoziativitat:

- $(\alpha | \beta) | \gamma \equiv \alpha | (\beta | \gamma)$
- $(\alpha\beta)\gamma \equiv \alpha(\beta\gamma)$

Kommutativitat:

- $\alpha | \beta \equiv \beta | \alpha$

Distributivitat:

- $\alpha(\beta | \gamma) \equiv \alpha\beta | \alpha\gamma$
- $(\alpha | \beta)\gamma \equiv \alpha\gamma | \beta\gamma$

Idempotenz:

- $\alpha | \alpha \equiv \alpha$

64

Stern:

Lemma 2.26

- $\epsilon \mid \alpha\alpha^* \equiv \alpha^*$
- $\alpha^*\alpha \equiv \alpha\alpha^*$
- $(\alpha^*)^* \equiv \alpha^*$

Beispiel 2.27

Herleitung einer Äquivalenz aus obigen Lemmas:

$$\begin{aligned}\epsilon &\mid \alpha^* \\ \equiv \epsilon &\mid (\epsilon \mid \alpha\alpha^*) && \text{Stern Lemma} \\ \equiv (\epsilon &\mid \epsilon) \mid \alpha\alpha^* && \text{Assoziativität} \\ \equiv \epsilon &\mid \alpha\alpha^* && \text{Idempotenz} \\ \equiv \alpha^* &&& \text{Stern Lemma}\end{aligned}$$

65

Lässt sich jede gültige Äquivalenz $\alpha \equiv \beta$ aus den obigen Lemmas für \equiv herleiten?

Satz 2.28 (Redko 1964)

Es gibt keine endliche Menge von gültigen Äquivalenzen aus denen sich alle gültigen Äquivalenzen herleiten lassen.

Wenn man mehr als nur Äquivalenzen zulässt:



Arto Salomaa.

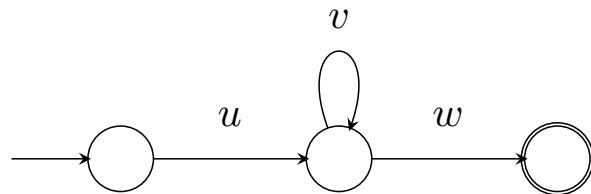
Two Complete Axiom Systems for the Algebra of Regular Events. Journal of the ACM, 1966.

66

2.8 Pumping Lemma

Oder: Wie zeigt man, dass eine Sprache nicht regulär ist?

Beispiel 2.29



Satz 2.30 (Pumping Lemma für reguläre Sprachen)

Sei $R \subseteq \Sigma^*$ regulär. Dann gibt es ein $n > 0$, so dass sich jedes $z \in R$ mit $|z| \geq n$ so in $z = uvw$ zerlegen lässt, dass

- $v \neq \epsilon$,
- $|uv| \leq n$, und
- $\forall i \geq 0. uv^i w \in R$.

67

Die logische Struktur des Satzes:

$$\forall \text{reg. } R. \exists n > 0. \forall z \in R. |z| \geq n \Rightarrow \exists u v w. z = uvw \wedge \dots$$

Als Spiel:

- Gib mir eine reguläre Sprache R
- Dann gebe ich dir ein $n > 0$ (abhängig von $R!!!$)
- Gibst du mir dann ein $z \in R$ mit $|z| \geq n$
- So kann ich z in uvw zerlegen so dass ...

Sprechweise: n ist eine Pumping-Lemma-Zahl für R , falls alle $z \in R$ mit $|z| \geq n$ sich so wie im Pumping-Lemma zerlegen und aufpumpen lassen.

68

Beweis:

Sei $R = L(A)$, $A = (Q, \Sigma, \delta, q_0, F)$.

Sei $n = |Q|$. Sei nun $z = a_1 \dots a_m \in R$ mit $m \geq n$.

Die beim Lesen von z durchlaufene Zustandsfolge sei

$$q_0 = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \dots \xrightarrow{a_m} p_m$$

Dann muss es $0 \leq i < j \leq n$ geben mit $p_i = p_j$.

Wir teilen z wie folg auf: $\underbrace{a_1 \dots a_i}_u \underbrace{a_{i+1} \dots a_j}_v \underbrace{a_{j+1} \dots a_{|z|}}_w$

Damit gilt:

- $|uv| \leq n$,
- $v \neq \epsilon$, und
- $\forall l \geq 0. uv^l w \in R$.

□

[Darf A NFA sein?]

69

Fazit:

Falls $L(M) = R$ so ist $|Q_M|$ eine Pumping-Lemma-Zahl für R .

Ist $|Q_M| + 1$ auch eine Pumping-Lemma-Zahl für R ?

70

Beispiel für die Anwendung des Pumping Lemmas:

Satz 2.31

Die Sprache $\{a^i b^i \mid i \in \mathbb{N}\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = a^n b^n \in L$.

Dann ist z zerlegbar in uvw mit

$u, v \in \{a\}^*$ (weil $|uv| \leq n$) und $v \neq \epsilon$.

Damit müsste gelten $a^{n-|v|} b^n = uw \in L$. $\not\subseteq$

□

Endliche Automaten können nicht unbegrenzt zählen

71

Ist die Sprache $\{a^n b^n \mid n \leq 10^6\}$ regulär?

72

Beispiel für die Anwendung des Pumping Lemmas:

Satz 2.32

$L = \{0^{m^2} \mid m \geq 0\}$ ist nicht regulär.

Beweis:

Angenommen, L sei doch regulär.

Sei n eine Pumping-Lemma-Zahl für L .

Wähle $z = 0^{n^2} \in L$. Dann ist z zerlegbar in uvw mit

$$1 \leq |v| \leq |uv| \leq n$$

und $uv^l w \in L$ für alle $l \in \mathbb{N}$. D.h. insb. $|uv^2w|$ ist Quadratzahl.

Dies führt zu einem Widerspruch:

$$\textcolor{red}{n^2} = |z| = |uvw| < |\textcolor{red}{uv^2w}| \leq n^2 + n < n^2 + 2n + 1 = (\textcolor{red}{n+1})^2$$

Denn zwischen n^2 und $(n+1)^2$ liegt keine Quadratzahl. □

Übungsaufgabe:

$\{1^p \mid p \text{ prim}\}$ ist nicht regulär.

Pumping-Lemma-Zahl für $L(ab^*c)$?

Pumping-Lemma-Zahl für $\{aaaaaa\}$?

Erinnerung:

n ist Pumping-Lemma-Zahl für L
gdw
alle $z \in L$ mit $|z| \geq n$ lassen sich aufpumpen.

Bemerkung

Es gibt nicht-reguläre Sprachen, für die das Pumping-Lemma gilt!

⇒ Pumping-Lemma hinreichend aber nicht notwendig um Nicht-Regularität zu zeigen.

regulär \subset Pumping-Lemma gilt \subset alle Sprachen

2.9 Entscheidbarkeit

- Welche Probleme sind für reguläre Sprachen entscheidbar?
Sehr viele!
- Wie hängt die Komplexität mit der Repräsentation zusammen:
DFA , NFA und RE?
Kommt darauf an ...

77

Statt Mengen verwendet man oft **Eigenschaften** oder **Probleme**.
Bsp: "ist prim" statt "ist Element der Primzahlen".

Eine Eigenschaft nennt man **entscheidbar** gdw die zugehörige Menge entscheidbar ist.

Bsp:

„Es ist entscheidbar, ob eine Zahl prim ist“

≡

„Die Menge der Primzahlen ist entscheidbar“

78

Definition 2.33

Sei D eine Beschreibung einer Sprache (DFA, NFA, RE, Grammatik, etc.).

Wortproblem: Gegeben w , gilt $w \in L(D)$?

Leerheitsproblem: Gilt $L(D) = \emptyset$?

Endlichkeitsproblem: Ist $L(D)$ endlich?

79

Das Wortproblem für DFAs ist in linearer Zeit entscheidbar:

Fakt 2.34

Sei M ein DFA. Das Problem $w \in L(M)$ ist in Zeit $O(|w|)$ entscheidbar.

Lemma 2.35

Jede reguläre Sprache ist in linearer Zeit entscheidbar.

Beweis:

R reguläre \implies Es gibt DFA M mit $L(M) = R$
 $\implies w \in R$ ist mit Hilfe von M in Zeit $O(|w|)$ entscheidbar. \square

80

Bisher Automat bzw Sprache fix. Die Eingabe besteht nur aus w .

Jetzt: Wort *und* Automat Eingabe.

Lemma 2.36

Das Problem $w \in L(N)$ ist für beliebiges Wort w und NFA N in Zeit $O(|Q|^2|w|)$ entscheidbar.

Beweis:

Sei $Q = \{1, \dots, s\}$, $q_0 = 1$ und $w = a_1 \dots a_n$.

$S := \{1\}$

for $i := 1$ **to** n **do** $S := \bigcup_{j \in S} \delta(j, a_i)$

return ($S \cap F \neq \emptyset$)

□

81

Lemma 2.37

Das Leerheitsproblem ist für NFAs und DFAs entscheidbar (in Zeit $O(|Q|^2|\Sigma|)$ bzw $O(|Q||\Sigma|)$).

Beweis:

$L(M) = \emptyset$ gdw kein Endzustand von q_0 erreichbar ist.

Dies ist eine einfache Suche in einem Graphen, die jede Kante maximal ein Mal benutzen muss.

Ein NFA hat $\leq |Q|^2|\Sigma|$ Kanten, ein DFA hat $\leq |Q||\Sigma|$ Kanten. □

Ist Σ fix, z.B. ASCII, so wird daraus $O(|Q|^2)$ bzw $O(|Q|)$.

82

Global: $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$

```
Reach( $K$ ) =  
   $R := \emptyset$   
   $W := K$   
  while  $W \neq \emptyset$  do  
    pick and remove some  $p \in W$   
    if  $p \notin R$  then  
       $R := R \cup \{p\}$   
       $W := W \cup \bigcup_{a \in \Sigma} \delta(p, a)$   
  return  $R$ 
```

83

Lemma 2.38

Für endliche Automaten ist das Endlichkeitsproblem entscheidbar.

Beweis:

$|L(M)| = \infty$ gdw von q_0 aus eine nicht-leere Schleife erreichbar ist, von der aus F erreichbar ist:

```
finite( $Q, \Sigma, \delta, q_0, F$ ) =  
   $R := \text{Reach}(\{q_0\})$   
   $C := \{p \in R \mid p \in \text{Reach}(\bigcup_{a \in \Sigma} \delta(p, a))\}$   
  return ( $\text{Reach}(C) \cap F = \emptyset$ )
```

□

84

Definition 2.39

Seien D_1 und D_2 Sprachbeschreibungen (DFAs, NFAs, REs, Grammatiken, etc.).

Äquivalenzproblem: Gilt $L(D_1) = L(D_2)$?

Lemma 2.40

Das Äquivalenzproblem ist für DFAs entscheidbar.

Beweis:

Folgt direkt aus

$$\begin{aligned}L_1 \subseteq L_2 &\Leftrightarrow L_1 \cap \overline{L_2} = \emptyset \\L_1 = L_2 &\Leftrightarrow L_1 \subseteq L_2 \wedge L_2 \subseteq L_1\end{aligned}$$

da für DFAs Komplement und Durchschnitt wieder endliche Automaten liefern und das Leerheitsproblem für endliche Automaten entscheidbar ist. □

85

Satz 2.41

Das Äquivalenzproblem für DFAs ist in Zeit $O(|Q_1||Q_2|)$ entscheidbar (bei fixem Σ).

Beweis:

Gegeben: DFAs M_1 mit m und M_2 mit n Zuständen.
Mit Hilfe der Produkt-Konstruktion für \cap folgt:

	Anzahl der Zustände
$L(M_1) \cap \overline{L(M_2)}$	mn
$\overline{L(M_1)} \cap L(M_2)$	mn

Das Leerheitsproblem ist jeweils in Zeit $O(mn)$ entscheidbar. □

86

Korollar 2.42

Das Äquivalenzproblem für NFAs ist in Zeit $O(2^{|Q_1|+|Q_2|})$ entscheidbar (bei fixem Σ).

Beweis: 2 NFAs mit m und n Zuständen \rightsquigarrow
2 DFAs mit 2^m und 2^n Zuständen \rightsquigarrow
Äquivalenztest in Zeit $O(2^m 2^n)$

Korollar 2.43

Das Äquivalenzproblem für reguläre Ausdrücke ist entscheidbar.

Fazit:

Die Kodierung der Eingabe (DFA, NFA, RE, ...) kann entscheidend für die Komplexität eines Problems sein.

2.10 Automaten und Gleichungssysteme

Nicht mehr *Beweisen* von Äquivalenzen

Gilt $XX^* \equiv X^*X$ für alle X ?

sondern *Lösen*:

Für welches X gilt $X \equiv aX \mid b$?

Anwendung:

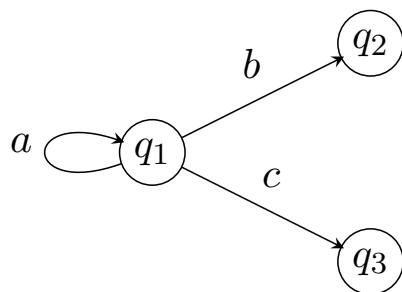
Automat \rightsquigarrow Gleichungssystem \rightsquigarrow RE

Technische Vereinfachung: die X_i stehen für (unbekannte) REs.

89

Beispiel 2.44

Ein Automatenfragment:



$$\begin{aligned} L_i &:= \{w \mid \hat{\delta}(q_i, w) \in F\} \\ &\implies \\ L_1 &= \{a\}L_1 \cup \{b\}L_2 \cup \{c\}L_3 \end{aligned}$$

Da die L_i regulär sein müssen(?), arbeiten wir direkt mit REs:

$$X_1 \equiv aX_1 \mid bX_2 \mid cX_3$$

Lösung X_i ist RE für die von q_i aus akzeptierte Sprache.

90

Satz 2.45 (Ardens Lemma)

Sind A , B und X Sprachen mit $\epsilon \notin A$, so gilt

$$X = AX \cup B \implies X = A^*B$$

Korollar 2.46

Sind α , β und X reguläre Ausdrücke mit $\epsilon \notin L(\alpha)$, so gilt

$$X \equiv \alpha X \mid \beta \implies X \equiv \alpha^* \beta$$

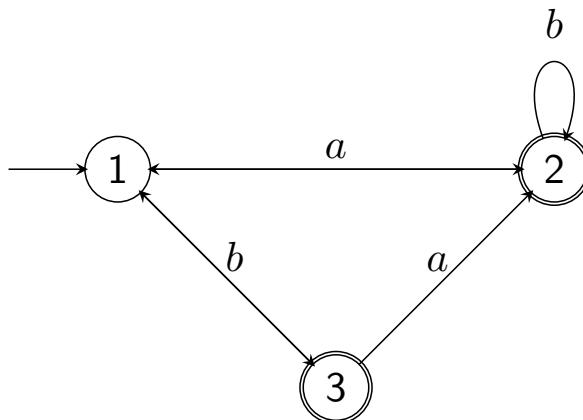
Bemerkungen

- $X = \{\epsilon\}X \cup B$ hat keine eindeutige Lösung:
jede Sprache $X \supseteq B$ ist Lösung.
- $X \equiv aXb \mid \epsilon$ hat keine reguläre Lösung.

91

Umwandlung eines DFAs in einen äquivalenten regulären Ausdruck:

Beispiel 2.47



Äquivalentes Gleichungssystem:

X_i ist ein RE für die von q_i aus akzeptierte Sprache.

$$X_1 \equiv aX_2 \mid bX_3$$

$$X_2 \equiv aX_1 \mid bX_2 \mid ?$$

$$X_3 \equiv bX_1 \mid aX_2 \mid ?$$

92

Lösen des Gleichungssystems:

$$\begin{aligned} X_1 &\equiv aX_2 \mid bX_3 \\ X_2 &\equiv aX_1 \mid bX_2 \mid \epsilon \\ X_3 &\equiv bX_1 \mid aX_2 \mid \epsilon \end{aligned}$$

Löse $X_2 \equiv bX_2 \mid (aX_1 \mid \epsilon)$ nach X_2 auf:

$$X_2 \equiv b^*(aX_1 \mid \epsilon)$$

Zurück einsetzen:

$$\begin{aligned} X_1 &\equiv a(b^*(aX_1 \mid \epsilon)) \mid bX_3 \\ X_3 &\equiv bX_1 \mid a(b^*(aX_1 \mid \epsilon)) \mid \epsilon \end{aligned}$$

Ausmultiplizieren und X_i ausklammern:

$$\begin{aligned} X_1 &\equiv ab^*aX_1 \mid ab^* \mid bX_3 \\ X_3 &\equiv (b \mid ab^*a)X_1 \mid ab^* \mid \epsilon \end{aligned}$$

93

$$\begin{aligned} X_1 &\equiv ab^*aX_1 \mid bX_3 \mid ab^* \\ X_3 &\equiv (b \mid ab^*a)X_1 \mid ab^* \mid \epsilon \end{aligned}$$

X_3 ist gelöst, in 1. Gleichung einsetzen:

$$X_1 \equiv ab^*aX_1 \mid b((b \mid ab^*a)X_1 \mid ab^* \mid \epsilon) \mid ab^*$$

Ausmultiplizieren und X_1 ausklammern:

$$X_1 \equiv (ab^*a \mid bb \mid bab^*a)X_1 \mid bab^* \mid b \mid ab^*$$

Nach X_1 auflösen:

$$X_1 \equiv (ab^*a \mid bb \mid bab^*a)^*(bab^* \mid b \mid ab^*)$$

94

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

- Wandle FA mit n Zuständen in ein System von n Gleichungen mit n Variablen um:

$$X_i \equiv a_{i1}X_1 \mid \cdots \mid a_{in}X_n \mid b_i$$

$$a_{ij} := c_1 \mid \cdots \mid c_k \quad \text{falls } \{c_1, \dots, c_k\} = \{c \in \Sigma \mid q_i \xrightarrow{c} q_j\}$$

wobei $a_{ij} := \emptyset$ falls $q_i \xrightarrow{c} q_j$ für kein $c \in \Sigma$

$$b_i := \begin{cases} \epsilon & \text{falls } q_i \in F \\ \emptyset & \text{sonst} \end{cases}$$

- Löse das System durch schrittweise Elimination von Variablen mit Hilfe von Ardens Lemma für REs (Korollar 2.46).
- Ist k der Startzustand, so beschreibt X_k die vom Automaten akzeptierte Sprache.

95

Das System in Matrix-Schreibweise, mit $+$ statt $|$:

$$x = Ax + b$$

wobei

$$x = \begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} \quad A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{pmatrix} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$$

Matrix-Addition und Multiplikation sind wie üblich definiert, aber auf der Element-Ebene mit

- REs statt Zahlen,
- Konkatenation statt Multiplikation,
- Alternative statt Addition.

96

Beispiel 2.48

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} aX_1 + bX_2 \\ cX_1 + dX_2 \end{pmatrix}$$

Achtung: Die Einträge in den Matrizen sind beliebige REs!

Wenn wir + und · auf Matrizen haben, was könnte * sein?

$$A^* := A^0 + A^1 + A^2 + \dots$$

Existiert das? Was ist das?

Es gilt: $A(i, j)$ beschreibt alle Wege von i nach j der Länge 1.

$A^n(i, j)$ beschreibt alle Wege von i nach j der Länge n .

Daher sollte gelten:

$A^*(i, j)$ beschreibt alle Wege von i nach j endlicher Länge.

Aus dem Beweis des Satzes von Kleene wissen wir:

$A^*(i, j)$ existiert und ist der reguläre Ausdruck α_{ij}^n .

97

Satz 2.49

Sei A eine $n \times n$ Matrix regulärer Ausdrücke.

Seien x und b Vektoren der Größe n .

Falls $\epsilon \notin L(A(i, j))$ für alle i, j , dann gilt

$$x = Ax + b \implies x = A^*b$$

Beweis: zB Kozen.

Berechnung von A^* : zB Kozen oder Satz von Kleene.

Bemerkung:

Die Nebenbedingung $\epsilon \notin L(A(i, j))$ ist automatisch gegeben, wenn A von einem Automaten ohne ϵ -Übergänge stammt, denn $A(i, j) = a_{ij}$ ist die Summe der Buchstaben, die von i nach j führen.

98

Wann wendet man welches Verfahren zum Lösen von Gleichungssystemen an:

- Schrittweises Lösen des Gleichungssystems mit Ardens Lemma: für Berechnungen per Hand.
- Matrizen-Ansatz mit A^* : für Theorie und Programmierung.

99

Beweis

von Ardens Lemma:

Wir nehmen an $X = AX \cup B$.

$$\begin{aligned} X &= A(AX \cup B) \cup B = A^2X \cup AB \cup B \\ &= A^2(AX \cup B) \cup AB \cup B = A^3X \cup A^2B \cup AB \cup B = \dots \end{aligned}$$

Mit Induktion zeigt man für alle $n \in \mathbb{N}$:

$$X = A^{n+1}X \cup \bigcup_{i \leq n} A^i B$$

0: Behauptung wird zu $X = AX \cup B$, der Annahme.

$n + 1$:

$$\begin{aligned} X &= A^{n+1}X \cup \bigcup_{i \leq n} A^i B \\ &= A^{n+1}(AX \cup B) \cup \bigcup_{i \leq n} A^{i+1}B \\ &= A^{n+2}X \cup A^{n+1}B \cup \bigcup_{i \leq n} A^i B \\ &= A^{n+2}X \cup \bigcup_{i \leq n+1} A^i B \end{aligned}$$

100

$$X = A^{n+1}X \cup \bigcup_{i \leq n} A^i B$$

Wir zeigen nun $X = A^*B$.

$$\begin{aligned} A^*B \subseteq X: \quad w \in A^*B &= \bigcup_{i \in \mathbb{N}} A^i B \\ &\implies \exists n. \quad w \in A^n B \\ &\implies w \in X \end{aligned}$$

$$X \subseteq A^*B: \quad \text{Sei } w \in X \text{ und } n := |w|.$$

$$\begin{aligned} \epsilon \notin A & \\ \implies \forall u \in A^{n+1}. \quad |u| &\geq n+1 \\ \implies w &\notin A^{n+1}X \\ \implies w \in \bigcup_{i \leq n} A^i B &\subseteq \bigcup_{i \in \mathbb{N}} A^i B = A^*B \end{aligned}$$

□

101

2.11 Minimierung endlicher Automaten

- ① Beispiele
- ② Algorithmen
- ③ Minimalitätsbeweis

102

Der Algorithmus zur Minimierung eines DFA:

- ① Entferne alle von q_0 aus nicht erreichbaren Zustände.
- ② Berechne die äquivalenten Zustände des Automaten.
- ③ Kollabiere den Automaten durch Zusammenfassung aller äquivalenten Zustände.

Zustände p und q sind unterschiedbar wenn es $w \in \Sigma^*$ gibt mit $\hat{\delta}(p, w) \in F$ und $\hat{\delta}(q, w) \notin F$ oder umgekehrt.

Zustände sind äquivalent wenn sie nicht unterschiedbar sind, d.h. wenn für alle $w \in \Sigma^*$ gilt:

$$\hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$$

Sind $\delta(p, a)$ und $\delta(q, a)$ unterschiedbar, dann auch p und q .

⇒ Unterscheidbarkeit pflanzt sich rückwärts fort.

103

Berechnung äquivalenter Zustände eines DFA

durch Berechnung der unterschiedbaren Zustände

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

Ausgabe: Äquivalenzrelation auf Q .

Datenstruktur: Eine Menge U ungeordneter Paare $\{p, q\} \subseteq Q$.

Algorithmus U:

- ① $U := \{\{p, q\} \mid p \in F \wedge q \notin F\}$
- ② **while** $\exists \{p, q\} \notin U. \exists a \in \Sigma. \{\delta(p, a), \delta(q, a)\} \in U$
do $U := U \cup \{\{p, q\}\}$

Invariante: $\{p, q\} \in U \implies p$ und q unterschiedbar

Lemma 2.50

Am Ende gilt: U ist Menge aller unterschiedbaren Zustände.

Beweis:

$\{p, q\} \in U \implies p$ und q unterschiedbar: Invariante

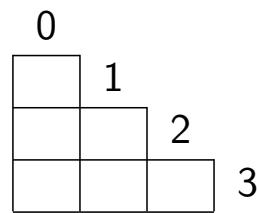
p und q unterschiedbar $\implies \{p, q\} \in U$:

Induktion über die Länge eines unterscheidenden Worts. □

104

Implementierung von U :

Tabelle von anfangs unmarkierten Paaren $\{p, q\}$, $p \neq q$.



for all $p \in F$, $q \in Q \setminus F$ **do** markiere $\{p, q\}$
while \exists unmarkiertes $\{p, q\} \exists a \in \Sigma$. $\{\delta(p, a), \delta(q, a)\}$ ist markiert
do markiere $\{p, q\}$

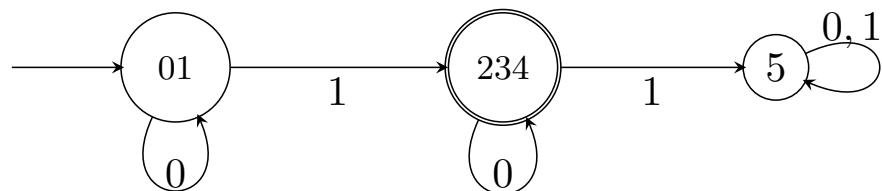
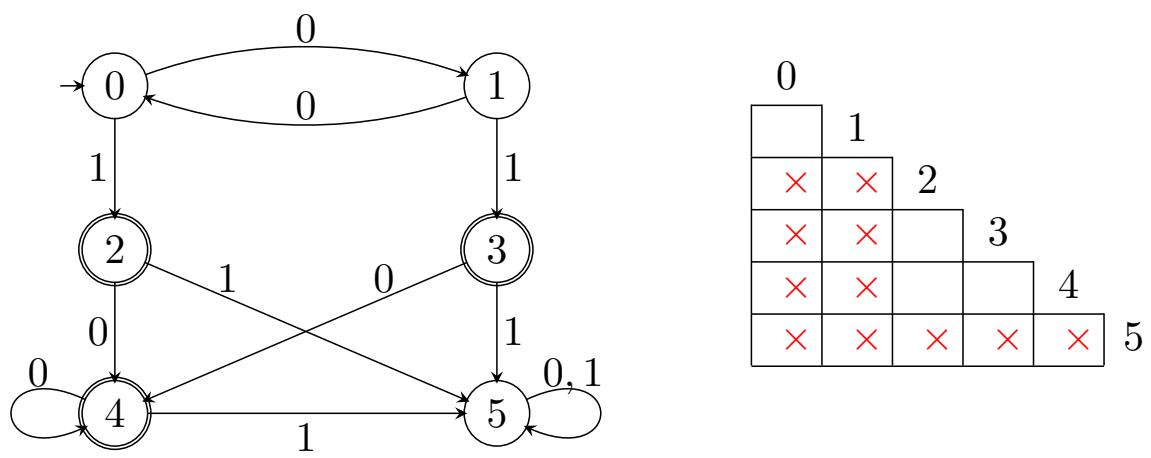
Komplexität:

$$O\left(\binom{n}{2} + \binom{n}{2}\binom{n}{2}|\Sigma|\right) = O\left(\frac{n(n-1)}{2} + \frac{n^2(n-1)^2}{4}|\Sigma|\right) = O(n^4)$$

Bei fixem Σ .

105

Beispiel 2.51



106

Von $O(n^4)$ zu $O(n^2)$ mit Abhängigkeitsanalyse:

$\{p', q'\}$ unterscheidbar \implies
 $\{p, q\}$ unterscheidbar falls $\{p, q\} \xrightarrow{a} \{p', q'\}$

$D[\{p', q'\}]$: Menge der Paare $\{p, q\}$ wie oben, anfangs leer

```
for all {p, q} ⊆ Q mit p ≠ q, a ∈ Σ do
    p' := δ(p, a); q' := δ(q, a)
    if p' ≠ q' then D[{p', q'}] := D[{p', q'}] ∪ {{p, q}}
    for all p' ∈ F, q' ∈ Q \ F do mark({p', q'})
```

```
mark({p', q'}) =
if {p', q'} unmarkiert then
    markiere {p', q'}
    for all pq ∈ D[{p', q'}] do mark(pq)
```

Komplexität: $O(n^2 + n^2) = O(n^2)$.

Korrektheit?

107



John Hopcroft.

An $n \log n$ Algorithm for Minimizing the States in a Finite Automaton. 1971.

108

Noch eine Anwendung: Äquivalenztest von DFAs.

- ① Gegeben DFAs A und B , bilde disjunkte Vereinigung.
„Male A und B nebeneinander.“)
- ② Berechne Menge der äquivalenten Zustände.
- ③ $L(A) = L(B)$ gdw die beiden Startzustände äquivalent sind.

109

Bisher: Der Minimierungsalgorithmus (zur Erinnerung).

- ① Entferne alle von q_0 aus nicht erreichbaren Zustände.
- ② Berechne die äquivalenten Zustände des Automaten.
- ③ Kollabiere den Automaten durch Zusammenfassung aller äquivalenten Zustände.

Jetzt: Die Präzisierung.

- ① Was ist der „kollabierte Automat“?
- ② Ist das wirklich der minimale Automat?

110

Eine Relation $\approx \subseteq A \times A$ ist eine Äquivalenzrelation falls

- Reflexivität: $\forall a \in A. a \approx a$
- Symmetrie: $\forall a, b \in A. a \approx b \implies b \approx a$
- Transitivität: $\forall a, b, c \in A. a \approx b \wedge b \approx c \implies a \approx c$

Äquivalenzklasse:

$$[a]_{\approx} := \{b \mid a \approx b\}$$

Es gilt:

$$[a]_{\approx} = [b]_{\approx} \Leftrightarrow a \approx b$$

Quotientenmenge:

$$A/\approx := \{[a]_{\approx} \mid a \in A\}$$

111

Im Folgenden sei $A = (Q, \Sigma, \delta, q_0, F)$ ein DFA ohne unerreichbare Zustände.

Definition 2.52 (Äquivalenz von Zuständen)

$$p \equiv_A q \Leftrightarrow (\forall w \in \Sigma^*. \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F)$$

Fakt 2.53

\equiv_A ist eine Äquivalenzrelation.

Wir schreiben \equiv statt \equiv_A wenn A klar ist.

Erinnerung:

Lemma 2.54

$$p \equiv_A q \implies \delta(p, a) \equiv_A \delta(q, a)$$

Lemma 2.55

Algorithmus U liefert die unterscheidbaren Zustände, also \neq .

In der weiteren Analyse beziehen wir uns direkt auf \equiv , nicht mehr auf den Algorithmus.

112

Die „Kollabierung“ von A bzgl. \equiv ist der *Quotentenautomat*:

Definition 2.56 (Quotentenautomat)

$$\begin{aligned} A/\equiv &:= (Q/\equiv, \Sigma, \delta', [q_0]_\equiv, F/\equiv) \\ \delta'([p]_\equiv, a) &:= [\delta(p, a)]_\equiv \end{aligned}$$

Die Definition von δ' ist wohlgeformt da unabhängig von der Wahl des Repräsentanten p :

$$\begin{aligned} [p]_\equiv = [p']_\equiv &\implies p \equiv p' \implies \delta(p, a) \equiv \delta(p', a) \\ &\implies [\delta(p, a)]_\equiv = [\delta(p', a)]_\equiv \end{aligned}$$

Lemma 2.57

$$L(A/\equiv) = L(A)$$

Beweis zur Übung.

113

Beobachtung

Für $p := \hat{\delta}(q_0, u)$ und $q := \hat{\delta}(q_0, v)$ gilt:

$$\begin{aligned} p \equiv_A q &\iff \forall w \in \Sigma^*. \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F \\ &\iff \forall w \in \Sigma^*. \hat{\delta}(q_0, uw) \in F \iff \hat{\delta}(q_0, vw) \in F \\ &\iff \forall w \in \Sigma^*. uw \in L(A) \iff vw \in L(A) \end{aligned}$$

Definition 2.58

Jede Sprache $L \subseteq \Sigma^*$ induziert eine Äquivalenzrelation $\equiv_L \subseteq \Sigma^* \times \Sigma^*$:

$$u \equiv_L v \iff \forall w \in \Sigma^*. uw \in L \iff vw \in L$$

D.h. u und v sind durch Anhängen von Wörtern bzgl. $\in L$ nicht unterscheidbar.

Obige Beobachtung lässt sich nun schreiben als

$$u \equiv_{L(A)} v \iff \hat{\delta}(q_0, u) \equiv_A \hat{\delta}(q_0, v)$$

114

Achtung

$p \equiv_A q$ ist eine Relation auf Zuständen von A

$u \equiv_L v$ ist eine Relation auf Wörtern

115

$$u \equiv_{L(A)} v \Leftrightarrow \hat{\delta}(q_0, u) \equiv_A \hat{\delta}(q_0, v)$$

Da alle Zustände von q_0 erreichbar sind, gilt sogar: Die Abbildung

$$[u]_{\equiv_{L(A)}} \mapsto [\hat{\delta}(q_0, u)]_{\equiv_A}$$

ist eine Bijektion zwischen den $\equiv_{L(A)}$ und \equiv_A Äquivalenzklassen.

Satz 2.59

Ist A ein DFA ohne unerreichbare Zustände, so ist der von Algorithmus U berechnete Quotientenautomat A/\equiv ein minimaler DFA für $L(A)$.

Beweis:

Sei $L := L(A)$ und A' ein DFA mit $L(A') = L$. Dann gilt:

$$|Q'| \geq |Q'/\equiv_{A'}| = |\Sigma^*/\equiv_L| = |Q/\equiv_A|$$

□

116

Es gilt sogar (Übung!):

Fakt 2.60

Alle Quotientenautomaten A/\equiv_A für die gleiche Sprache $L(A)$ haben die gleiche Struktur, d.h. sie unterscheiden sich nur durch eine Umbenennung der Zustände.

Daher beschriften wir die Zustände des kanonischen Minimalautomaten für eine Sprache L mit \equiv_L Äquivalenzklassen.

Beispiel 2.61

Sei $L := \{w \in \{0, 1\}^* \mid w \text{ endet mit } 00\}$.

Die einzigen drei \equiv_L Äquivalenzklassen sind:

$$[\epsilon]_{\equiv_L} = \{w \mid w \text{ endet nicht mit } 0\}$$

$$[0]_{\equiv_L} = \{w \mid w \text{ endet mit } 0, \text{ aber nicht mit } 00\}$$

$$[00]_{\equiv_L} = \{w \mid w \text{ endet mit } 00\}$$

117

Definition 2.62 (Kanonischer Minimalautomat)

$$M_L := (\Sigma^*/\equiv_L, \Sigma, \delta_L, [\epsilon]_{\equiv_L}, F_L)$$

mit $\delta_L([w]_{\equiv_L}, a) := [wa]_{\equiv_L}$ und $F_L := \{[w]_{\equiv_L} \mid w \in L\}$.

Man sieht: δ_L ist wohldefiniert und $\hat{\delta}_L([\epsilon]_{\equiv_L}, w) = [w]_{\equiv_L}$.
Dann gilt offensichtlich $L(M_L) = L$.

Satz 2.63 (Myhill-Nerode)

Eine Sprache $L \subseteq \Sigma^*$ ist genau dann regulär, wenn \equiv_L endlich viele Äquivalenzklassen hat.

Beweis:

„ \Rightarrow “: Ist L regulär, so wird L von einem DFA A akzeptiert.

Daher hat \equiv_L so viele Äquivalenzklassen wie A/\equiv_A Zustände.

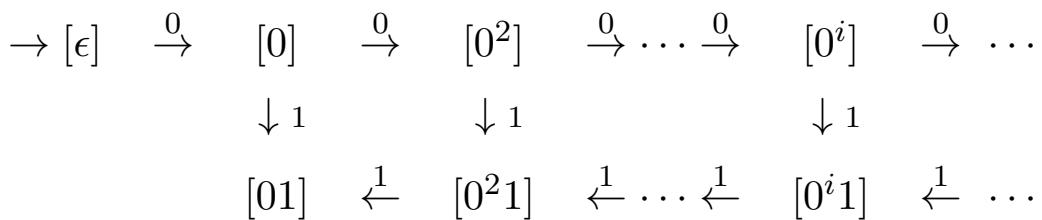
„ \Leftarrow “: Hat \equiv_L endlich viele Äquivalenzklassen,
so ist M_L ein DFA mit $L(M_L) = L$. □

118

Wie sieht M_L aus wenn L nicht regulär ist?

Beispiel 2.64

$$L = \{0^i 1^i \mid i \in \mathbb{N}\}$$



Warum $0^i \not\equiv_L 0^j$ falls $i \neq j$?

Was fehlt noch?

Vollständige Methode um **Nichtregularität** von L zu zeigen:

Gib **unendliche Menge** w_1, w_2, \dots an mit $w_i \not\equiv_L w_j$ falls $i \neq j$.

Bemerkung

Eindeutigkeit des minimalen Automaten (modulo Umbenennung der Zustände) gilt nur bei DFAs, nicht bei NFAs!

Aufgabe: Finde Gegenbeispiel für NFAs

Knobelaufgabe:

Nach Def. gilt $p \not\equiv_A q$ gdw. $\hat{\delta}(p, w) \in F \nLeftrightarrow \hat{\delta}(q, w) \in F$

Man zeige: Falls $p \not\equiv_A q$, dann gibt es ein w der Länge $< |Q|$, das p und q unterscheidet, d.h. $\hat{\delta}(p, w) \in F \nLeftrightarrow \hat{\delta}(q, w) \in F$.

121

Rückblick reguläre Sprachen

- DFA, NFA, ϵ -NFA und RE definieren die gleiche Sprachklasse.
 - Potenzmengenkonstruktion (exponentiell)
 - Strukturelle Übersetzung von RE nach ϵ -NFA
 - Übersetzung NFA nach RE, zB über Gleichungssysteme (exponentiell)
- Regularitätserhaltende Konstruktionen auf Sprachen bzw Automaten:
 - $\cup, \cap, \dots, {}^R, \dots$
 - Für welche Darstellung wie teuer?
(Komplement, Produkt, ...)
 - NFA kompakt aber oft teuer; DFA oft billiger
- Entscheidungsprobleme auf Automaten und REs:
 - Direkt entscheidbar?
Auf Automat? (Oft mit Erreichbarkeit) Auf RE?
 - Reduzierbar auf anderes Problem?
ZB $L(A) \subseteq L(B)$ auf $L(A) \cap \overline{L(B)} = \emptyset$
 - Für welche Darstellung wie teuer?

122

- Pumping-Lemma
- Äquivalenzen \equiv zw REs:
 - Standardregeln: Kommutativität etc, Beweis mit $L(\cdot)$
 - $\alpha \equiv \beta$ entscheidbar da $L(\alpha) = L(\beta)$ über Automaten entscheidbar
 - Auch für REs mit Variablen entscheidbar: betrachte Variablen als Konstanten
 - Gleichungssysteme lösbar durch Variablenelimination mit Ardens Lemma
- Minimierung
 - Algorithmen zur Kollabierung
 - Relationen \equiv_A und \equiv_L
 - Quotientenautomat A/\equiv_A minimal, da gleichgroß wie kanonischer minimaler Automat $M_{L(A)}$
 - L genau dann regulär wenn \equiv_L endlich viele Klassen hat, dh wenn M_L endlich ist (Myhill-Nerode).

123

3. Kontextfreie Sprachen

3.1 Kontextfreie Grammatiken

Beispiel 3.1 (Arithmetische Ausdrücke)

$$\begin{aligned}
 <\text{Expr}> &\rightarrow <\text{Term}> \\
 <\text{Expr}> &\rightarrow <\text{Expr}> + <\text{Term}> \\
 <\text{Term}> &\rightarrow <\text{Factor}> \\
 <\text{Term}> &\rightarrow <\text{Term}> * <\text{Factor}> \\
 <\text{Factor}> &\rightarrow a \\
 <\text{Factor}> &\rightarrow (<\text{Expr}>)
 \end{aligned}$$

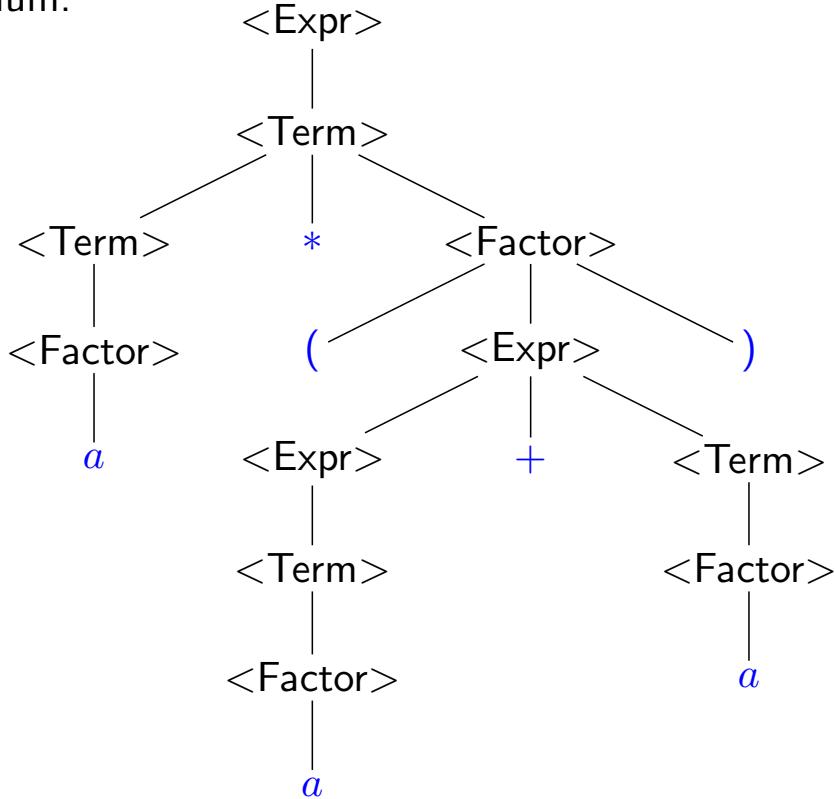
Eine (Links)Ableitung:

$<\text{Expr}> \rightarrow$

$\rightarrow a * (a + a)$

124

Der Syntaxbaum:



Die Blätter des Baums, von links nach rechts gelesen,
ergeben das abgeleitete Wort

125

Bemerkungen

- Der vollständige Syntaxbaum enthält die gesamte Information über die Ableitung, bis auf die (irrelevante) Reihenfolge des Aufbaus.
- Kontextfreie Grammatiken dienen zur Spezifikation von Sprachen. Ihre Implementierung ist das **Parsen**:
 - Die Überprüfung, ob ein Wort von einer Grammatik abgeleitet werden kann, bzw
 - Die Erzeugung des Syntaxbaums (*parse tree*).

Parsen ist die Transformation eines Worts in einen Syntaxbaum.

126

Definition 3.2

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

- V ist eine endlichen Menge, die **Nichtterminalzeichen** (oder **Variablen**),
- Σ ist ein Alphabet, die **Terminalzeichen**, disjunkt von V ,
- $P \subseteq V \times (V \cup \Sigma)^*$ eine endlichen Menge, die **Produktionen**, und
- $S \in V$ ist das **Startsymbol**.

Konventionen:

- A, B, C, \dots sind Nichtterminale,
- a, b, c, \dots (und Sonderzeichen wie $+, *, \dots$) sind Terminalen,
- $\alpha, \beta, \gamma, \dots \in (V \cup \Sigma)^*$
- Produktionen schreiben wir $A \rightarrow \alpha$ statt $(A, \alpha) \in P$.
- Statt $A \rightarrow \alpha_1, A \rightarrow \alpha_2, A \rightarrow \alpha_3$ schreiben wir einfach

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3$$

127

Beispiel 3.3 (Arithmetische Ausdrücke)

$$V = \{E, T, F\}$$

$$\Sigma = \{a, +, *, (,)\}$$

$$P =$$

$$\left\{ \begin{array}{l} E \rightarrow T \mid E + T \\ T \rightarrow F \mid T * F \\ F \rightarrow a \mid (E) \end{array} \right\}$$

$$S = E$$

128

Definition 3.4

Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$ induziert eine Ableitungsrelation \rightarrow_G auf Wörtern über $V \cup \Sigma$:

$$\alpha \rightarrow_G \beta$$

gdw es eine Regel $A \rightarrow \gamma$ in P gibt, und Wörter α_1, α_2 , so dass

$$\alpha = \alpha_1 A \alpha_2 \quad \text{und} \quad \beta = \alpha_1 \gamma \alpha_2$$

Beispiel:

$$a + T + a \quad \rightarrow_G \quad a + T * F + a$$

129

Definition 3.5 (Reflexive transitive Hülle)

$$\alpha \rightarrow_G^0 \alpha$$

$$\alpha \rightarrow_G^{n+1} \gamma \quad :\Leftrightarrow \quad \exists \beta. \alpha \rightarrow_G^n \beta \rightarrow_G \gamma$$

$$\alpha \rightarrow_G^* \beta \quad :\Leftrightarrow \quad \exists n. \alpha \rightarrow_G^n \beta$$

$$\alpha \rightarrow_G^+ \beta \quad :\Leftrightarrow \quad \exists n > 0. \alpha \rightarrow_G^n \beta$$

Beispiel: $E \rightarrow_G^{11} a * (a + a)$ und daher $E \rightarrow_G^* a * (a + a)$.

Wir nennen

$$\alpha_1 \rightarrow_G \alpha_2 \rightarrow_G \cdots \rightarrow_G \alpha_n$$

eine Linksableitung gdw in jedem Schritt das linkste Nichtterminal in α_i ersetzt wird.

130

Definition 3.6 (Kontextfreie Sprache)

Eine kontextfreien Grammatik $G = (V, \Sigma, P, S)$ erzeugt die Sprache

$$L(G) := \{w \in \Sigma^* \mid S \xrightarrow{G}^* w\}$$

Eine Sprache $L \subseteq \Sigma^*$ heißt kontextfrei gdw es eine kontextfreie Grammatik G gibt mit $L = L(G)$.

Abkürzungen:

CFG Kontextfreie Grammatik (*context-free grammar*)

CFL Kontextfreie Sprache (*context-free language*)

Konvention:

Ist G aus dem Kontext eindeutig ersichtlich, so schreibt man auch nur $\alpha \rightarrow \beta$ statt $\alpha \rightarrow_G \beta$.

131

Beispiel 3.7

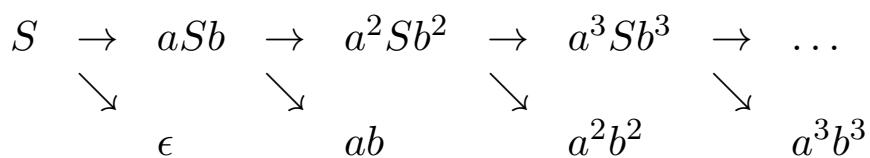
Die nicht-reguläre Sprache $L = \{a^n b^n \mid n \in \mathbb{N}\}$ ist kontextfrei, da sie von der CFG

$$S \rightarrow aSb \mid \epsilon$$

erzeugt wird. Genauer: $L = L(G)$ wobei $G = (V, \Sigma, P, S)$ mit

$$\begin{aligned} V &= \{S\} \\ \Sigma &= \{a, b\} \\ P &= \{S \rightarrow aSb \mid \epsilon\} \end{aligned}$$

Der unendliche Baum aller möglichen (Links-)Ableitungen:



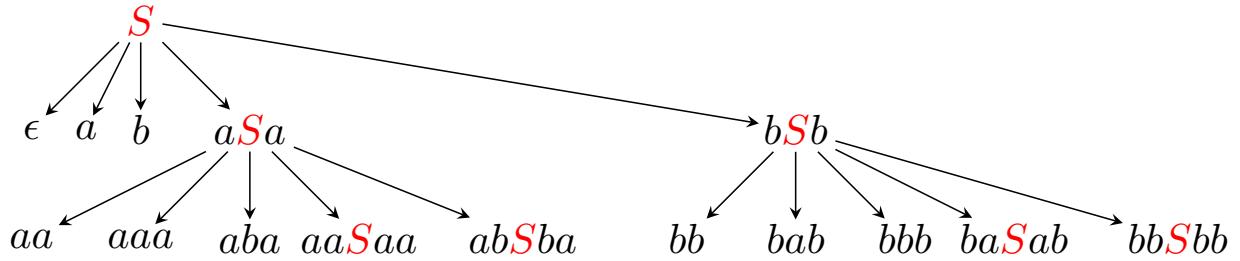
132

Beispiel 3.8

Die nicht-reguläre Sprache der Palindrome ($w = w^R$) über $\{a, b\}$ wird von folgender CFG erzeugt:

$$S \rightarrow \epsilon \mid a \mid b \mid aSa \mid bSb$$

Der Anfang des unendlichen Baums aller möglichen (Links-)Ableitungen:



Achtung, dies ist *kein* Syntaxbaum!

133

Lemma 3.9 (Dekompositionslemma)

$$\begin{aligned} \alpha_1 \alpha_2 &\xrightarrow{G} \beta \\ \Leftrightarrow \\ \exists \beta_1, \beta_2, n_1, n_2. \quad \beta &= \beta_1 \beta_2 \quad \wedge \quad n = n_1 + n_2 \quad \wedge \quad \alpha_i \xrightarrow{G} \beta_i \quad (i = 1, 2) \end{aligned}$$

Beweis:

Übung!

□

134

Definition 3.10

Eine CFG heißt **rechts-linear** gdw jede Produktion von der Form

$$A \rightarrow aB \quad \text{oder} \quad A \rightarrow \epsilon \quad \text{ist.}$$

Eine CFG heißt **links-linear** gdw jede Produktion von der Form

$$A \rightarrow Ba \quad \text{oder} \quad A \rightarrow \epsilon \quad \text{ist.}$$

Lemma 3.11

Die rechts-linearen und links-linearen Grammatiken erzeugen jeweils genau die regulären Sprachen.

Beweis: Übung!

Korollar 3.12

Die regulären Sprachen sind eine echte Teilklasse der kontextfreien Sprachen.

3.2 Induktive Definitionen, Syntaxbäume und Ableitungen

- ① Beispiel: Balancierte Klammern
- ② Induktive Definitionen und Syntaxbäume allgemein
- ③ Äquivalenz von Ableitung, Syntaxbaum und induktiver Erzeugung

Beispiel 3.13

$$S \rightarrow \epsilon \mid [S] \mid SS$$

Um zu zeigen, dass diese Grammatik die Menge aller balancierten Klammerwörter in $\{[,]\}^*$ erzeugt, betrachten wir die Grammatik als induktive Definition einer Sprache $L_G(S)$:

$\epsilon \in L_G(S)$
$u \in L_G(S) \implies [u] \in L_G(S)$
$u \in L_G(S) \wedge v \in L_G(S) \implies uv \in L_G(S)$

Damit gilt zB:

$$\begin{aligned}\epsilon \in L_G(S) &\implies [] \in L_G(S) \implies [[]] \in L_G(S) \\ &\implies [[]] [] \in L_G(S)\end{aligned}$$

137

Bemerkungen

- Die Produktionen (\rightarrow) erzeugen Wörter *top-down*, d.h. von einem Nichtterminal zu einem Wort hin.
- Die induktive Definition (\implies) erzeugt Wörter *bottom-up*, d.h. sie setzt kleinere Wörter zu größeren zusammen.
- Die induktive Definition betrachtet nur Wörter aus Σ^* .

138

Zur induktiven Definition von $L_G(S)$ gehört ein Induktionsprinzip:

Um zu zeigen, dass für alle $u \in L_G(S)$ eine Eigenschaft $P(u)$ gilt,
dh $\forall u \in L_G(S). P(u)$, zeige:

$P(\epsilon)$
$P(u) \implies P([u])$
$P(u) \wedge P(v) \implies P(uv)$

„Induktion über die Erzeugung von u “

Lemma 3.14

Alle $u \in L_G(S)$ enthalten gleich viele [wie].

Beweis:

Mit Induktion über die Erzeugung von u :

- ϵ enthält 0 [und].
- Enthält u gleich viele [wie], so auch $[u]$.
- Enthalten u und v gleich viele [wie], so auch uv .

□

139

Hinweis

Die Aussagen

$$\forall x \in M. P(x)$$

und

$$\forall x. x \in M \implies P(x)$$

sind logisch äquivalent.

Der Allquantor wird dann oft weggelassen:

$$x \in M \implies P(x)$$

140

Definition 3.15

Präfix:

$$u \preceq w \iff \exists v. uv = w$$

Anzahl der Vorkommen:

$$\#_a(w) := \text{Anzahl der Vorkommen von } a \text{ in } w$$

Für unser Beispiel führen wir zwei Abk. ein:

$$A(w) := \#_{[]} (w) \quad B(w) := \#_{]} (w)$$

Wir nennen $w \in \{[], []\}^*$ balanciert gdw

- (1) $A(w) = B(w)$ und
- (2) für alle Präfixe u von w gilt $A(u) \geq B(u)$

- Balanciert: $[]$, $[[]]$, $[] []$, $[[[]]]$
- Nicht balanciert: $] [$, $][][$

141

Satz 3.16

Die Grammatik $S \rightarrow \epsilon \mid [S] \mid SS$ erzeugt genau die Menge der balancierten Wörter.

Beweis:

$$u \in L_G(S) \implies u \text{ balanciert:}$$

Mit Ind. über die Erzeugung von u . (1) bewiesen, wir zeigen (2).

$$\epsilon: p \preceq \epsilon \implies p = \epsilon \implies A(p) = B(p)$$

[u]: Sei $p \preceq [u]$

Fall $p = \epsilon$: $A(p) = B(p)$

Fall $p = [u]$: $A(p) = A(u) + 1 = B(u) + 1 = B(p)$

Fall $p = [q]$ mit $q \preceq u$:

$$A(p) = A(q) + 1 \geq B(q) + 1 > B(q) = B(p)$$

uv : Sei $p \preceq uv$.

Fall $p \preceq u$: $A(p) \geq B(p)$ mit IA für u

Fall $p = uq$ und $q \preceq v$:

$$A(p) = A(u) + A(q) = B(u) + A(q) \geq B(u) + B(q) = B(uq) = B(p)$$

142

Beweis (Forts.)

u balanciert $\Rightarrow u \in L_G(S)$:

Mit vollständiger Induktion über $n := |u|$. (dh $u = a_1 \dots a_n$)

IA: Jedes balancierte Wort $< n$ liegt in $L_G(S)$.

Zz: $u \in L_G(S)$.

Falls $n = 0$, so $u = \epsilon \in L_G(S)$.

Sei $n > 0$.

Betrachte Werteverlauf von $h(w) := A(w) - B(w)$:

$h(\epsilon), h(a_1), h(a_1 a_2), \dots, h(a_1 \dots a_n)$ (alle $\geq 0!$).

Insb. gilt $a_1 = [$ und $a_n =]$.

- Es gibt nur die Nullstellen $h(\epsilon)$ und $h(a_1 \dots a_n)$.

Dann ist $v := a_2 \dots a_{n-1}$ balanciert:

$$(1) \quad A(v) = A([v]) - 1 = B([v]) - 1 = B(v)$$

$$(2) \quad p \preceq v: h([p]) = A([p]) - B([p]) > 0 \Rightarrow$$

$$A(p) = A([p]) - 1 > B([p]) - 1 = B(p) - 1 \Rightarrow$$

$$A(p) \geq B(p)$$

$$\Rightarrow v \in L_G(S) \text{ (nach IA)} \Rightarrow u = [v] \in L_G(S)$$

143

Beweis (Forts.):

- Es gibt noch eine Nullstelle $h(a_1 \dots a_k)$.

Sei $u_1 := a_1 \dots a_k$, $u_2 := a_{k+1} \dots a_n$

Dann sind u_1 und u_2 balanciert:

$$(1) \quad A(u_1) = B(u_1) \text{ da } h(u_1) = 0;$$

$$A(u_2) = A(u) - A(u_1) = B(u) - B(u_1) = B(u_2)$$

$$(2) \quad p \preceq u_1 \Rightarrow p \preceq u \Rightarrow A(p) \geq B(p)$$

$$p \preceq u_2: A(p) = A(u_1 p) - A(u_1) \geq B(u_1 p) - B(u_1) = B(p)$$

$$\Rightarrow u_1, u_2 \in L_G(S) \text{ (nach IA, da } |u_i| < n\text{)}$$

$$\Rightarrow u = u_1 u_2 \in L_G(S)$$

□

144

Moral:

- „ $w \in L_G(S) \implies P(w)$ “ beweist man immer schematisch mit Induktion über die Erzeugung von w .
- „ $P(w) \implies w \in L_G(S)$ “ beweist man oft mit Induktion über $|w|$. Erfordert meist Kreativität.

145

Wir übertragen jetzt die Idee der induktiven Erzeugung und der dazugehörige Induktionsregel auf beliebige CFGs $G = (V, \Sigma, P, S)$. Sei $V = \{A_1, \dots, A_k\}$. Damit ist jede Produktion von der Form

$$A_i \rightarrow w_0 A_{i_1} w_1 \dots w_{n-1} A_{i_n} w_n$$

Man kann G wie folgt als eine (simultane) induktive Definition von Sprachen $L_G(A_i)$ für all $A_i \in V$ sehen:

Jede Produktion korrespondiert zu einer Erzeugungsregel

$$u_1 \in L_G(A_{i_1}) \wedge \dots \wedge u_n \in L_G(A_{i_n}) \implies w_0 u_1 w_1 \dots w_{n-1} u_n w_n \in L_G(A_i)$$

Aussagen der Form

$$(u \in L_G(A_1) \implies P_1(u)) \wedge \dots \wedge (u \in L_G(A_k) \implies P_k(u))$$

kann man durch **simultane Induktion über die Erzeugung von u** beweisen, indem man für jede Produktion zeigt, dass

$$P_{i_1}(u_1) \wedge \dots \wedge P_{i_n}(u_n) \implies P_i(w_0 u_1 w_1 \dots w_{n-1} u_n w_n)$$

146

Beispiel 3.17

Grammatik:

$$A \rightarrow \epsilon \mid aB \quad B \rightarrow Aa$$

Induktive Definition:

- $\epsilon \in L_G(A)$
- $w \in L_G(B) \implies aw \in L_G(A)$
- $w \in L_G(A) \implies wa \in L_G(B)$

Beim induktiven Beweis von

$$(w \in L_G(A) \implies \#_a(w) \text{ ist gerade}) \wedge \\ (w \in L_G(B) \implies \#_a(w) \text{ ist ungerade})$$

muss man zeigen

- $\#_a(\epsilon) \text{ ist gerade}$
- $\#_a(w) \text{ ist ungerade} \implies \#_a(aw) \text{ ist gerade}$
- $\#_a(w) \text{ ist gerade} \implies \#_a(wa) \text{ ist ungerade}$

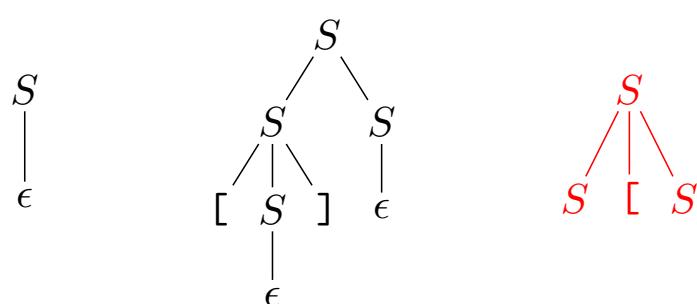
147

Definition 3.18 (Syntaxbaum)

Ein **Syntaxbaum** für eine Ableitung mit einer Grammatik $G = (V, \Sigma, P, S)$ ist ein Baum, so dass

- jedes Blatt mit einem Zeichen aus $\Sigma \cup \{\epsilon\}$ beschriftet ist,
- jeder innere Knoten mit einem $A \in V$ beschriftet ist, und falls die Nachfolger (von links nach rechts) mit $X_1, \dots, X_n \in V \cup \Sigma \cup \{\epsilon\}$ beschriftet sind, dann ist $A \rightarrow X_1 \dots X_n$ eine Produktion in P ,
- ein Blatt ϵ der einzige Nachfolger seines Vorgängers ist.

Beispiel 3.19 (Syntaxbäume für $S \rightarrow \epsilon \mid [S] \mid SS$)



148

Satz 3.20

Für eine CFG und ein $w \in \Sigma^*$ sind folgende Bedingungen äquivalent:

- ① $A \xrightarrow{G}^* w$
- ② $w \in L_G(A)$ (gemäß der induktiven Definition)
- ③ Es gibt einen Syntaxbaum, dessen **Rand** (Blätter von links nach rechts gelesen) das Wort w ist.

Beweis:

Skizze (Details: [HMU])

1 \Rightarrow 2: Sei $A \xrightarrow{G}^n w$. Wir konstruieren dazu einen Beweis für $w \in L_G(A)$ mit Induktion über n . Die Ableitung muss von folgender Form sein:

$$A \xrightarrow{G} w_0 A_1 w_1 \dots A_m w_m \xrightarrow{G}^{n-1} w$$

Nach dem Dekompositionslemma muss es u_i und $n_i \leq n - 1$ geben mit $A_i \xrightarrow{G}^{n_i} u_i$ und $w = w_0 u_1 w_1 \dots u_m w_m$. Nach IA (da $n_i < n$) gilt, gilt $u_i \in L_G(A_i)$ und daher (mit einem weiteren Erzeugungsschritt) auch $w \in L_G(A)$.

149

2 \Rightarrow 3: Parallel zur induktiven Erzeugung von $w \in L_G(A)$ kann man einen Syntaxbaum mit Rand w erzeugen. Formal: Induktion über die Erzeugung von w .

3 \Rightarrow 1: Jeder Baum lässt sich in eine Ableitung, sogar eine Linksableitung(!) umformen. Induktion über die Höhe des Baums: Transformiere die Unterbäume t_i mit Beschriftung A_i in Ableitungen $A_i \xrightarrow{G}^* u_i$ und füge diese mit dem Dekompositionslemma zu einer grossen Ableitung $A \xrightarrow{G} w_0 A_1 w_1 \dots A_m w_m \xrightarrow{G}^* w_0 u_1 w_1 \dots u_m w_m$ zusammen.

Definition 3.21

- Eine CFG G heißt **mehrdeutig** gdw es ein $w \in L(G)$ gibt, das zwei verschiedene Syntaxbäume hat, also zwei verschiedene Syntaxbäume mit Wurzel S und Rand w .
- Eine CFL L heißt **inhärent mehrdeutig** gdw jede CFG G mit $L(G) = L$ mehrdeutig ist.

Beispiel 3.22

Die Grammatik $E \rightarrow E + E \mid E * E \mid (E) \mid a$ ist mehrdeutig — betrachte $a + a * a$. Die erzeugte Sprache ist aber nicht inhärent mehrdeutig (siehe Beispiel Arithmetische Ausdrücke).

Satz 3.23

Die Sprache $\{a^i b^j c^k \mid i = j \vee j = k\}$ ist inhärent mehrdeutig.

151

3.3 Die Chomsky-Normalform

Definition 3.24

Eine kontextfreie Grammatik G ist in **Chomsky-Normalform** gdw alle Produktionen eine der Formen

$$A \rightarrow a \quad \text{oder} \quad A \rightarrow BC$$

haben.

Wir konstruieren und beweisen nun schrittweise:

Satz 3.25

Zu jeder CFG G kann man eine CFG G' in Chomsky-Normalform konstruieren mit $L(G') = L(G) \setminus \{\epsilon\}$.

Wer auf $\epsilon \in L(G')$ nicht verzichten möchte:

Füge am Ende wieder $S \rightarrow \epsilon$ hinzu.

152

Wir nennen $A \rightarrow \epsilon$ eine ϵ -Produktion.

Lemma 3.26

Zu jeder CFG $G = (V, \Sigma, P, S)$ kann man eine CFG G' konstruieren, die keine ϵ -Produktionen enthält, so dass gilt
 $L(G') = L(G) \setminus \{\epsilon\}$

Beweis:

Wir erweitern P induktiv zu einer Obermenge \hat{P} :

- ① Jede Produktion aus P ist in \hat{P}
- ② Sind $B \rightarrow \epsilon$ und $A \rightarrow \alpha B \beta$ in \hat{P} , so füge auch $A \rightarrow \alpha \beta$ hinzu.

Offensichtlich gilt $L(\hat{G}) = L(G)$:

Jede neue Produktion kann von 2 alten simuliert werden.

Das Ergebnis G' ist \hat{G} ohne die ϵ -Produktionen.

Denn diese sind nun überflüssig:

153

Beweis (Forts.):

Sei $S \xrightarrow[\hat{G}]{}^* w$, $w \neq \epsilon$, eine Ableitung minimaler Länge.

Käme darin $B \rightarrow \epsilon$ vor, also

$$S \xrightarrow[\hat{G}]{}^* \gamma B \delta \xrightarrow[\hat{G}]{} \gamma \delta \xrightarrow[\hat{G}]{}^* w$$

so wäre γ oder δ nicht leer, dh B muss durch eine Produktion $A \rightarrow \alpha B \beta$ eingeführt worden sein:

$$S \xrightarrow[\hat{G}]{}^k \alpha' A \beta' \xrightarrow[\hat{G}]{} \alpha' \alpha B \beta \beta' \xrightarrow[\hat{G}]{}^m \gamma B \delta \xrightarrow[\hat{G}]{} \gamma \delta \xrightarrow[\hat{G}]{}^n w$$

Dann wäre aber folgende Ableitung mit $(A \rightarrow \alpha \beta) \in \hat{P}$ kürzer:

$$S \xrightarrow[\hat{G}]{}^k \alpha' A \beta' \xrightarrow[\hat{G}]{} \alpha' \alpha \beta \beta' \xrightarrow[\hat{G}]{}^m \gamma \delta \xrightarrow[\hat{G}]{}^n w$$

Also kommen in Ableitungen minimaler Länge keine ϵ -Produktionen vor. Letztere sind also überflüssig. □

154

Beispiel 3.27

$$S \rightarrow AB, \quad A \rightarrow aAA \mid \epsilon, \quad B \rightarrow bBB \mid \epsilon$$

155

Wir nennen $A \rightarrow B$ eine **Kettenproduktion**.

Lemma 3.28

Zu jeder CFG $G = (V, \Sigma, P, S)$ kann man eine CFG G' konstruieren, die keine Kettenproduktionen enthält, so dass gilt $L(G') = L(G)$.

Beweis:

Wir erweitern P induktiv zu einer Obermenge \hat{P} :

- ① Jede Produktion aus P ist in \hat{P}
- ② Sind $A \rightarrow B$ und $B \rightarrow \alpha$ in \hat{P} , so füge auch $A \rightarrow \alpha$ hinzu.

Das Ergebnis G' ist \hat{G} ohne die (nun überflüssigen) Kettenproduktionen.

Rest des Beweises analog zur Elimination von ϵ -Produktionen. □

156

Beispiel 3.29

$$A \rightarrow a \mid B, \quad B \rightarrow b \mid C, \quad C \rightarrow A$$

157

Konstruktion einer Chomsky-Normalform

Eingabe: Eine kontextfreie Grammatik $G = (V, \Sigma, P, S)$

- ① Eliminiere alle ϵ -Produktionen.
- ② Eliminiere alle Kettenproduktionen.
- ③ Füge für jedes $a \in \Sigma$, das in einer rechten Seite der Länge ≥ 2 vorkommt, ein neues Nichtterminal A_a zu V hinzu, ersetze a in allen rechten Seiten der Länge ≥ 2 durch A_a , und füge $A_a \rightarrow a$ zu P hinzu.
- ④ Ersetze jede Produktion der Form

$$A \rightarrow B_1 B_2 \cdots B_k \ (k \geq 3)$$

durch

$$A \rightarrow B_1 C_2, \ C_2 \rightarrow B_2 C_3, \ \dots, \ C_{k-1} \rightarrow B_{k-1} B_k$$

wobei C_2, \dots, C_{k-1} neue Nichtterminale sind.

158

Definition 3.30

Eine CFG ist in [Greibach-Normalform](#) (benannt nach Sheila Greibach, UCLA), falls jede Produktion von der Form

$$A \rightarrow aA_1 \dots A_n$$

ist.

Satz 3.31

Zu jeder CFG G gibt es eine CFG G' in Greibach-Normalform mit $L(G') = L(G) \setminus \{\epsilon\}$.

3.4 Das Pumping-Lemma für kontextfreie Sprachen

Zur Erinnerung: Das Pumping-Lemma für reguläre Sprachen: Für jede reguläre Sprache L gibt es ein $n \in \mathbb{N}$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in $z = uvw$ mit $|uv| \leq n$, $v \neq \epsilon$ und $uv^*w \subseteq L$.

Zum Beweis haben wir $n = |Q|$ gewählt, wobei Q die Zustandsmenge eines L erkennenden DFA war. Das Argument war dann, dass beim Erkennen von z (mindestens) ein Zustand zweimal besucht werden muss und damit der dazwischen liegende Weg im Automaten beliebig oft wiederholt werden kann.

Ein ähnliches Argument kann auch auf kontextfreie Grammatiken angewendet werden.

Satz 3.32 (Pumping-Lemma)

Für jede kontextfreie Sprache L gibt es ein $n \geq 1$, so dass sich jedes Wort $z \in L$ mit $|z| \geq n$ zerlegen lässt in

$$z = uvwxy,$$

mit

- $vx \neq \epsilon$,
- $|vwx| \leq n$, und
- $\forall i \in \mathbb{N}. uv^iwx^i y \in L$.

161

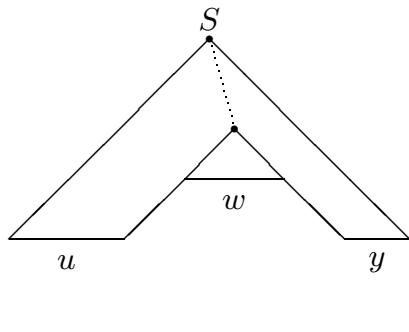
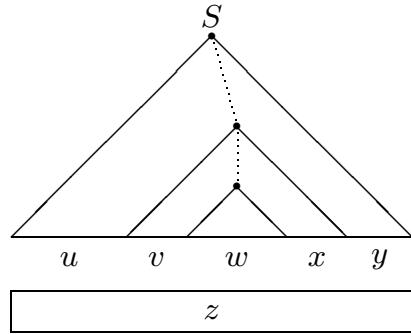
Beweis:

Sei $G = (V, \Sigma, P, S)$ eine CFG in Chomsky-Normalform mit $L(G) = L \setminus \{\epsilon\}$. Wähle $n = 2^{|V|}$. Sei $z \in L(G)$ mit $|z| \geq n$.

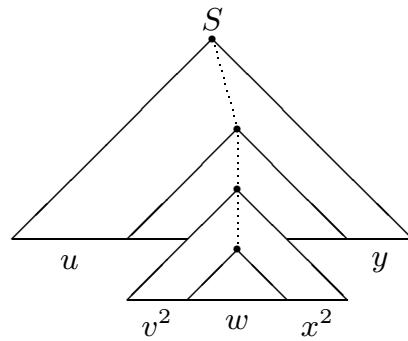
Jeder Binärbaum mit $\geq 2^k$ Blättern hat die Höhe $\geq k$

Dann hat der Syntaxbaum für z (ohne die letzte Stufe für die Terminale) mindestens einen Pfad der Länge $\geq |V|$, da er wegen der Chomsky-Normalform ein Binärbaum ist: Wir betrachten einen solchen Pfad maximaler Länge. Auf diesem Pfad der Länge $\geq |V|$ kommen $\geq |V| + 1$ Nichtterminale vor, also mindestens eins doppelt — nennen wir es A . Die zwischen zwei Vorkommen von A liegende Teilableitung kann nun beliebig oft wiederholt werden.

162



Dieser Ableitungsbaum zeigt
 $uvwy \in L$



Dieser Ableitungsbaum zeigt
 $uv^2wx^2y \in L$

Beweis (Forts.):

Die Bedingung $vx \neq \epsilon$ folgt, da das obere der beiden A 's mit $A \rightarrow BC$ abgeleitet werden muss.

Um $|vwx| \leq n$ zu erreichen, darf das obere A maximal $|V| + 1$ Schritte von einem Blatt entfernt sein. Da der ausgewählte Pfad maximale Länge hat, genügt es, dass das obere A vom Blatt dieses Pfads $|V| + 1$ Schritte entfernt ist. Da es nur $|V|$ Nichtterminale gibt, muss ein solches doppeltes A existieren. □

Beispiel 3.33

Wir zeigen, dass die Sprache

$$L := \{a^i b^i c^i \mid i \in \mathbb{N}\}$$

nicht kontextfrei ist.

Wäre L kontextfrei, so gäbe es eine dazugehörige Pumping-Lemma Zahl n und wir könnten jedes $z \in L$ mit $|z| \geq n$, insb. $z = a^n b^n c^n$, zerlegen und aufpumpen, ohne aus der Sprache herauszufallen.

Eine Zerlegung $z = uvwxy$ mit $|vwx| \leq n$ bedeutet aber, dass vwx nicht a 's und c 's enthalten kann. OE nehmen wir an, vwx enthält nur a 's und b 's. Da $vx \neq \epsilon$, muss vx mindestens ein a oder ein b enthalten. Damit gilt aber $\#_a(uv^2wx^2y) > \#_c(uv^2wx^2y)$ oder $\#_b(uv^2wx^2y) > \#_c(uv^2wx^2y)$. Also $uv^2wx^2y \notin L$. $\textcolor{red}{\sharp}$

165

Beispiel 3.34

Mit dem Pumping-Lemma kann man zeigen, dass die Sprache $\{ww \mid w \in \{a, b\}^*\}$ nicht kontextfrei ist. Dies zeigt, dass Kontextbedingungen in Programmiersprachen, etwa „*Deklaration vor Benutzung*“, nicht durch kontextfreie Grammatiken ausgedrückt werden können.

166

3.5 Abschlusseigenschaften

Satz 3.35

Seien kontextfreie Grammatiken $G_1 = (V_1, \Sigma_1, P_1, S_1)$ und $G_2 = (V_2, \Sigma_2, P_2, S_2)$ gegeben. Dann kann man in linearer Zeit kontextfreie Grammatiken für

- ① $L(G_1) \cup L(G_2)$
- ② $L(G_1)L(G_2)$
- ③ $(L(G_1))^*$
- ④ $(L(G_1))^R$

konstruieren.

Die Klasse der kontextfreien Sprachen ist also unter *Vereinigung*, *Konkatenation*, *Stern* und *Spiegelung* abgeschlossen.

167

Beweis:

OE nehmen wir an, dass $V_1 \cap V_2 = \emptyset$.

- ① $V = V_1 \cup V_2 \cup \{S\}$; S neu
 $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\}$
- ② $V = V_1 \cup V_2 \cup \{S\}$; S neu
 $P = P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}$
- ③ $V = V_1 \cup \{S\}$; S neu
 $P = P_1 \cup \{S \rightarrow \epsilon \mid S_1 S\}$
- ④ $P = \{A \rightarrow \alpha^R \mid (A \rightarrow \alpha) \in P_1\}$



168

Satz 3.36

Die Menge der kontextfreien Sprachen ist **nicht** abgeschlossen unter Durchschnitt oder Komplement.

Beweis:

$L_1 := \{a^i b^i c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_2 := \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$ ist kontextfrei

$L_1 \cap L_2 = \{a^i b^i c^i \mid i \in \mathbb{N}\}$ ist nicht kontextfrei

Wegen **de Morgan** (1806–1871) können die CFLs daher auch nicht unter Komplement abgeschlossen sein. \square

169

3.6 Algorithmen für kontextfreie Grammatiken

Wie üblich: $G = (V, \Sigma, P, S)$ ist eine CFG.

Ein Symbol $X \in V \cup \Sigma$ ist

nützlich gdw es eine Ableitung $S \xrightarrow{*} G w \in \Sigma^*$ gibt, in der X vorkommt.

erzeugend gdw es eine Ableitung $X \xrightarrow{*} G w \in \Sigma^*$ gibt.

erreichbar gdw es eine Ableitung $S \xrightarrow{*} G \alpha X \beta$ gibt.

Fakt 3.37

Nützliche Symbole sind erzeugend und erreichbar.

Aber nicht notwendigerweise umgekehrt:

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

Ziel: Elimination der unnützen Symbole und der Produktionen, in denen sie vorkommen.

170

Beispiel 3.38

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

- ① Elimination nicht erzeugender Symbole:

$$S \rightarrow a, \quad A \rightarrow b$$

- ② Elimination unerreichbarer Symbole:

$$S \rightarrow a$$

Umgekehrte Reihenfolge:

- ① Elimination unerreichbarer Symbole:

$$S \rightarrow AB \mid a, \quad A \rightarrow b$$

- ② Elimination nicht erzeugender Symbole:

$$S \rightarrow a, \quad A \rightarrow b$$

171

Satz 3.39

Eliminiert man aus einer Grammatik G

- ① alle nicht erzeugenden Symbole, mit Resultat G_1 ,
- ② aus G_1 alle unerreichbaren Symbole, mit Resultat G_2 ,

dann enthält G_2 nur noch nützliche Symbole und $L(G_2) = L(G)$.

Beweis:

Wir zeigen zuerst $L(G_2) = L(G)$.

Da $P_2 \subseteq P$ gilt $L(G_2) \subseteq L(G)$.

Umgekehrt, sei $w \in L(G)$, dh $S \xrightarrow{G}^* w$.

Jedes Symbol in dieser Ableitung ist erreichbar und erzeugend.

Also gilt auch $S \xrightarrow{G_2}^* w$, dh $w \in L(G_2)$.

172

Beweis (Forts.): $[G_1 : \text{erzeugend in } G, G_2 : \text{erreichbar in } G_1]$

Wir zeigen: Alle $X \in V_2 \cup \Sigma_2$ sind nützlich in G_2 , dh

$$S \rightarrow_{G_2}^* \dots X \dots \rightarrow_{G_2}^* \dots \in \Sigma_2^*$$

X muss in G_1 erreichbar sein, dh $S \rightarrow_{G_1}^* \alpha X \beta$.

Da alle Symbole in der Ableitung erreichbar sind: $S \rightarrow_{G_2}^* \alpha X \beta$.

Alle Symbole in $\alpha X \beta$ müssen in G erzeugend sein:

$$\forall Y \in \alpha X \beta. \exists u \in \Sigma^*. Y \rightarrow_G^* u$$

Da alle Symbole in den Ableitungen $Y \rightarrow_G^* u$ erzeugend sind:

$$\forall Y \in \alpha X \beta. \exists u \in \Sigma_1^*. Y \rightarrow_{G_1}^* u$$

Also gibt es $w \in \Sigma_1^*$ mit $\alpha X \beta \rightarrow_{G_1}^* w$.

Da $S \rightarrow_{G_1}^* \alpha X \beta$: Die Ableitung $\alpha X \beta \rightarrow_{G_1}^* w$ enthält nur Symbole, die in G_1 erreichbar sind. Daher auch $\alpha X \beta \rightarrow_{G_2}^* w \in \Sigma_2^*$.

$$S \rightarrow_{G_2}^* \alpha X \beta \rightarrow_{G_2}^* w \in \Sigma_2^*$$

□

173

Satz 3.40

Die Menge der erzeugenden Symbole einer CFG sind berechenbar.

Beweis:

Wir berechnen die erzeugenden Symbole induktiv:

- Alle Symbole in Σ sind erzeugend.
- Falls $(A \rightarrow \alpha) \in P$ und alle Symbole in α sind erzeugend, dann ist auch A erzeugend.

□

Beispiel 3.41

$$S \rightarrow SAB, \quad A \rightarrow BC, \quad B \rightarrow C, \quad C \rightarrow c$$

Erzeugend: $\{c, C, B, A\}$.

Korollar 3.42

Für eine kontextfreie Grammatik G ist entscheidbar, ob $L(G) = \emptyset$.

Beweis:

□

174

Satz 3.43

Die Menge der erreichbaren Symbole einer CFG ist berechenbar.

Beweis:

Wir berechnen die erreichbaren Symbole induktiv:

- S ist erreichbar.
- Ist A erreichbar und $(A \rightarrow \alpha X \beta) \in P$, so ist auch X erreichbar.

□

175

Satz 3.44

Das Wortproblem ($w \in L(G) ?$) ist für eine CFG G entscheidbar.

Beweis:

OE sei $w \neq \epsilon$. Wir eliminieren zuerst alle ϵ -Produktionen aus G (wie in Lemma 3.26).

Dann berechnen wir induktiv die Menge R aller von S ableitbaren Wörter $\in (V \cup \Sigma)^*$, die nicht länger als w sind:

- $S \in R$
- Wenn $\alpha B \gamma \in R$ und $(B \rightarrow \beta) \in P$ und $|\alpha \beta \gamma| \leq |w|$, dann auch $\alpha \beta \gamma \in R$.

Man zeigt:

$$w \in L_V(G) \Leftrightarrow w \in R$$

wobei $L_V(G) := \{w \in (V \cup \Sigma)^* \mid S \xrightarrow{G}^* w\}$.

Da R endlich ist ($|R| \leq |V \cup \Sigma|^{|w|}$), ist $w \in R$ entscheidbar, und damit auch $w \in L_V(G)$, und damit auch $w \in L(G)$. □

176

3.7 Der Cocke-Younger-Kasami-Algorithmus

Der CYK-Algorithmus entscheidet das Wortproblem für kontextfreie Grammatiken in Chomsky-Normalform.

Eingabe: Grammatik $G = (V, \Sigma, P, S)$ in Chomsky-Normalform, $w = a_1 \dots a_n \in \Sigma^*$.

Definition 3.45

$$V_{ij} := \{A \in V \mid A \xrightarrow{G}^* a_i \dots a_j\} \quad \text{für } i \leq j$$

Damit gilt:

$$w \in L(G) \Leftrightarrow S \in V_{1n}$$

177

Der CYK-Algorithmus berechnet die V_{ij} rekursiv nach wachsendem $j - i$:

$$\begin{aligned} V_{ii} &= \{A \in V \mid (A \rightarrow a_i) \in P\} \\ V_{ij} &= \left\{ A \in V \mid \begin{array}{l} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j} \\ (A \rightarrow BC) \in P \end{array} \right\} \quad \text{für } i < j \end{aligned}$$

Korrektheitsbeweis: Induktion nach $j - i$.

Die V_{ij} als Tabelle (mit ij statt V_{ij}):

	14			
13	24			
12	23	34		
11	22	33	44	
	a_1	a_2	a_3	a_4

178

Beispiel 3.46

$$\begin{array}{lcl} S & \rightarrow & AB \mid BC \\ A & \rightarrow & BA \mid a \\ B & \rightarrow & CC \mid b \\ C & \rightarrow & AB \mid a \end{array}$$

15					
14	25				
13	24	35			
12	23	34	45		
11	22	33	44	55	
	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>

179

Satz 3.47

Der CYK-Algorithmus entscheidet das Wortproblem $w \in L(G)$ für eine fixe CFG G in Chomsky-Normalform in Zeit $O(|w|^3)$.

Beweis:

Sei $n := |w|$. Es werden $\frac{n(n-1)}{2} \in O(n^2)$ Mengen V_{ij} berechnet.

$$V_{ij} = \left\{ A \in V \mid \begin{array}{c} \exists i \leq k < j, B \in V_{ik}, C \in V_{k+1,j}. \\ (A \rightarrow BC) \in P \end{array} \right\} \quad (i < j)$$

Für jede dieser Mengen werden

- $j - i < n$ Werte für k betrachtet,
- für jedes k wird für alle Produktionen $A \rightarrow BC$ untersucht, ob $B \in V_{ik}$ und $C \in V_{k+1,j}$, wobei $|V_{ik}|, |V_{k+1,j}| \leq |V|$.

Gesamtzeit: $O(n^3)$

Denn $|P|$ und $|V|$ sind Konstanten unabhängig von n .

[Konstruktion jeder Menge V_{ii} : $O(\)$. Für alle V_{ii} also $O(\)$.] □

180

Erweiterung

Der CYK-Algorithmus kann so erweitert werden, dass er nicht nur das Wortproblem entscheidet, sondern auch die Menge der Syntaxbäume für die Eingabe berechnet.

Realisierung:

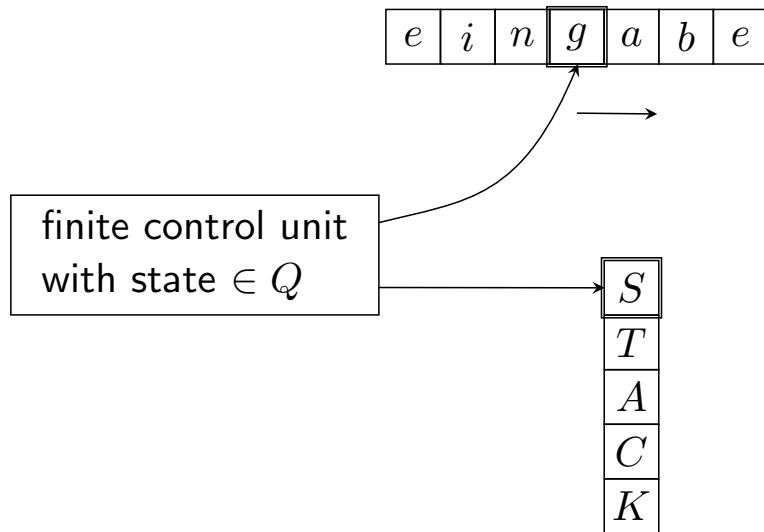
- V_{ij} ist die Menge der Syntaxbäume mit Rand $a_i \dots a_j$.
- Statt A enthält V_{ij} einen Syntaxbaum, dessen Wurzel mit A beschriftet ist.

Vorschau

Für CFGs sind folgende Probleme nicht entscheidbar:

- Äquivalenz: $L(G_1) = L(G_2)$?
- Schnittproblem: $L(G_1) \cap L(G_2) = \emptyset$?
- Regularität: $L(G)$ regulär?
- Mehrdeutigkeit: Ist G mehrdeutig?

3.8 Kellerautomaten



183

Anwendungsgebiete von Kellerautomaten:

- Syntaxanalyse von Programmiersprachen
- Analyse von Programmen mit Rekursion

184

Definition 3.48

- Ein PDA M akzeptiert $w \in \Sigma^*$ mit Endzustand gdw

$$(q_0, w, Z_0) \xrightarrow{M}^* (f, \epsilon, \gamma) \text{ für ein } f \in F, \gamma \in \Gamma^*.$$

$$L_F(M) := \{w \mid \exists f \in F, \gamma \in \Gamma^*. (q_0, w, Z_0) \xrightarrow{M}^* (f, \epsilon, \gamma)\}$$

- Ein PDA M akzeptiert $w \in \Sigma^*$ mit leeren Keller gdw

$$(q_0, w, Z_0) \xrightarrow{M}^* (q, \epsilon, \epsilon) \text{ für ein } q \in Q.$$

$$L_\epsilon(M) := \{w \mid \exists q \in Q. (q_0, w, Z_0) \xrightarrow{M}^* (q, \epsilon, \epsilon)\}$$

Konvention: Wir blenden die F -Komponente von M aus, wenn wir nur an $L_\epsilon(M)$ interessiert sind.

185

Beispiel 3.49

Die Sprache $L = \{ww^R \mid w \in \{0, 1\}^*\}$ wird vom PDA

$$M = (\{p, q, r\}, \{0, 1\}, \{0, 1, Z_0\}, p, Z_0, \delta, \{r\})$$

$$\begin{aligned}\delta(p, a, Z) &= \{(p, aZ)\} \quad \text{für } a \in \{0, 1\}, Z \in \{0, 1, Z_0\} \\ \delta(p, \epsilon, Z) &= \{(q, Z)\} \quad \text{für } Z \in \{0, 1, Z_0\} \\ \delta(q, a, a) &= \{(q, \epsilon)\} \quad \text{für } a \in \{0, 1\} \\ \delta(q, \epsilon, Z_0) &= \{(r, \epsilon)\}\end{aligned}$$

sowohl mit Endzustand als auch mit leerem Keller akzeptiert.

Bsp: $(p, 1111, Z_0) \xrightarrow{M}^* (r, \epsilon, \epsilon)$.

Definiert man $\delta(q, \epsilon, Z_0)$ als $\{(q, \epsilon)\}$ statt $\{(r, \epsilon)\}$ so gilt:

$$L_F(M') = \emptyset \text{ aber } L_\epsilon(M') = L.$$

186

Bemerkungen: PDAs und das Wortproblem

- Mit einem NFA A kann man $w \in L(A)$ durch parallele Verfolgung aller Berechnungspfade entscheiden, da sie alle endlich sind.
- Bei einem ϵ -NFA gibt es auch unendliche Berechnungspfade, die aber einfach zu eliminieren sind.
- Bei einem PDA kann es wegen ϵ -Übergängen auch unendliche Berechnungen \rightarrow_M geben, zB $\delta(q, \epsilon, Z) = (q, ZZ)$:

$$(q, w, Z) \rightarrow_M (q, w, ZZ) \rightarrow_M (q, w, ZZZ) \rightarrow_M \dots$$

Diese sind wegen des möglicherweise wachsenden oder pulsierenden Kellers nicht einfach zu eliminieren.

- Daher ist es a priori unklar, wie man mit einem PDA das Wortproblem entscheidet.

187

Ziel:

Akzeptanz durch Endzustände und leeren Keller gleich mächtig.

Satz 3.50 (Endzustand \rightarrow leerer Keller)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ kann man in linearer Zeit einen PDA $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta')$ konstruieren mit $L_F(M) = L_\epsilon(M')$.

Ziel:

$$(q_0, w, Z_0) \xrightarrow{M}^* (f, \epsilon, \gamma) \Leftrightarrow (q'_0, w, Z'_0) \xrightarrow{M'}^* (q, \epsilon, \epsilon)$$

188

Beweisskizze:

M' (mit leerem Keller) simuliert M (mit Endzustand). Zusätzlich:

- Sobald M einen Endzustand erreicht, darf M' den Keller leeren (im neuen Zustand \bar{q}).
- Um zu verhindern, dass der Keller von M' leer wird, ohne dass M in einem Endzustand ist, führen wir ein neues Kellersymbol Z'_0 ein.

$$\begin{aligned} Q' &:= Q \uplus \{\bar{q}, q'_0\} \\ \Gamma' &:= \Gamma \uplus \{Z'_0\} \end{aligned}$$

Wir erweitern δ zu δ' :

$$\begin{aligned} \delta'(q'_0, \epsilon, Z'_0) &= \{(q_0, Z_0 Z'_0)\} \\ \delta'(q, b, Z) &= \delta(q, b, Z) && \text{für } q \in Q \setminus F, b \in \Sigma \cup \{\epsilon\}, Z \in \Gamma \\ \delta'(f, a, Z) &= \delta(f, a, Z) && \text{für } f \in F, a \in \Sigma, Z \in \Gamma \\ \delta'(f, \epsilon, Z) &= \delta(f, \epsilon, Z) \cup \{(\bar{q}, \epsilon)\} && \text{für } f \in F, Z \in \Gamma' \\ \delta'(\bar{q}, \epsilon, Z) &= \{(\bar{q}, \epsilon)\} && \text{für } Z \in \Gamma' \end{aligned} \quad \square$$

189

Satz 3.51 (Leerer Keller \rightarrow Endzustand)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$ kann man in linearer Zeit einen PDA $M' = (Q', \Sigma, \Gamma', q'_0, Z'_0, \delta', F)$ konstruieren mit $L_\epsilon(M) = L_F(M')$.

Beweisskizze:

M' (mit Endzustand) simuliert M (mit leerem Keller). Zusätzlich:

- M' schreibt am Anfang ein neues Zeichen Z'_0 auf den Keller.
- Sobald M auf dem Keller Z'_0 findet, ist der Keller eigentlich leer, und M' geht in den (neuen) Endzustand f .

$$\begin{aligned} Q' &:= Q \uplus \{q'_0, f\} \\ \Gamma' &:= \Gamma \uplus \{Z'_0\} \\ F &:= \{f\} \end{aligned}$$

$$\begin{aligned} \delta'(q'_0, \epsilon, Z'_0) &= \{(q_0, Z_0 Z'_0)\} \\ \delta'(q, b, Z) &= \delta(q, b, Z) && \text{für } q \in Q, b \in \Sigma \cup \{\epsilon\}, Z \in \Gamma \\ \delta'(q, \epsilon, Z'_0) &= \{(f, Z'_0)\} && \text{für } q \in Q \end{aligned} \quad \square$$

190

Die folgenden Sätze erleichtern Beweise über Berechnungen von PDAs.

Lemma 3.52 (Erweiterungslemma)

$$(q, u, \alpha) \rightarrow_M^n (q', u', \alpha') \implies (q, uv, \alpha\beta) \rightarrow_M^n (q', u'v, \alpha'\beta)$$

Beweis:

Nach Definition von \rightarrow_M gilt

$$\begin{aligned} (q_i, u_i, \alpha_i) &\rightarrow_M (q_{i+1}, u_{i+1}, \alpha_{i+1}) \implies \\ (q_i, u_i v, \alpha_i \beta) &\rightarrow_M (q_{i+1}, u_{i+1} v, \alpha_{i+1} \beta) \end{aligned}$$

Mit Induktion über n folgt die Behauptung. □

Gilt die Umkehrung, zB

$$(q_1, aa, XZ) \rightarrow_M^* (q_3, \epsilon, YZ) \implies (q_1, aa, X) \rightarrow_M^* (q_3, \epsilon, Y) ?$$

191

Keller



Eingabe

192

Satz 3.53 (Zerlegungssatz)

Wenn $(q, w, Z_{1\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$ dann gibt es u_i, p_i, n_i , so dass

$$(p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w = u_{1\dots k}$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n . Basis trivial.

Schritt: Eine Berechnung der Länge $n + 1$ hat die Form

$$(q, bw', Z_{1\dots k}) \rightarrow_M (p, w', Y_{1\dots l} Z_{2\dots k}) \rightarrow_M^n (q', \epsilon, \epsilon)$$

IA $\Rightarrow \exists v_j, r_j, m_j$ ($j = 1, \dots, l$), $\exists u_i, p_i, n_i$ ($i = 2, \dots, k$) mit

$$(r_{j-1}, v_j, Y_j) \rightarrow_M^{m_j} (r_j, \epsilon, \epsilon) \quad (p_{i-1}, u_i, Z_i) \rightarrow_M^{n_i} (p_i, \epsilon, \epsilon)$$

und $w' = v_{1\dots l} u_{2\dots k}$, $r_0 = p$, $r_l = p_1$, $p_k = q'$, $\sum m_j + \sum n_i = n$.

Mit Erweiterungslemma: $(r_{j-1}, v_{j\dots l}, Y_{j\dots l}) \rightarrow_M^{m_j} (r_j, v_{j+1\dots l}, Y_{j+1\dots l})$

$$\Rightarrow (r_0, v_{1\dots l}, Y_{1\dots l}) \rightarrow_M^{n_1-1} (r_l, \epsilon, \epsilon) \text{ wobei } n_1 := 1 + \sum m_j.$$

Aus $(q, bv_{1\dots l}, Z_1) \rightarrow_M (p, v_{1\dots l}, Y_{1\dots l})$ folgt $(p_0, u_1, Z_1) \rightarrow_M^{n_1} (p_1, \epsilon, \epsilon)$ wobei $p_0 := q$, $u_1 := bv_{1\dots l}$, $p_1 := r_l$.

Damit auch $w = bw' = bv_{1\dots l} u_{2\dots k} = u_{1\dots k}$ und $\sum n_i = n + 1$. □

193

3.9 Äquivalenz von PDAs und CFGs

Satz 3.54 (CFG \rightarrow PDA)

Zu jeder CFG G kann man einen PDA M konstruieren, der mit leerem Stack akzeptiert, so dass

$$L_\epsilon(M) = L(G).$$

Konstruktion:

Zuerst bringen wir alle Produktionen von G in die Form

$$A \rightarrow bB_1 \dots B_k$$

wobei $b \in \Sigma \cup \{\epsilon\}$. Methode: Für jedes $a \in \Sigma$

- ① füge ein neues A_a zu V hinzu,
- ② ersetze a rechts in P durch A_a (außer am Kopfende),
- ③ füge eine neue Produktion $A_a \rightarrow a$ hinzu.

194

Alle Produktionen in $G = (V, \Sigma, P, S)$ haben jetzt die Form

$$A \rightarrow bB_1 \dots B_k$$

Der PDA wird wie folgt definiert:

$$M := (\{q\}, \Sigma, V, q, S, \delta)$$

wobei

$$(A \rightarrow b\beta) \in P \implies \delta(q, b, A) \ni (q, \beta)$$

also für alle $b \in \Sigma \cup \{\epsilon\}$ und $A \in V$:

$$\delta(q, b, A) := \{(q, \beta) \mid (A \rightarrow b\beta) \in P\}$$

195

Lemma 3.55

Für alle $u, v \in \Sigma^*$ und $\gamma \in V^*$ und $A \in V$ gilt:

$$A \xrightarrow{G} u\gamma \text{ mit Linksableitung gdw } (q, uv, A) \xrightarrow{M} (q, v, \gamma)$$

Beweis:

Mit Induktion über n . Basis $n = 0$ trivial. Schritt:

$$\begin{aligned} & A \xrightarrow{G}^{n+1} u\gamma \\ \Leftrightarrow & \exists (B \rightarrow b\beta) \in P, w, \alpha. A \xrightarrow{G}^n wB\alpha \xrightarrow{G} wb\beta\alpha = u\gamma \\ & \quad (\text{d.h. } wb = u, \beta\alpha = \gamma) \\ \Leftrightarrow & (q, wbv, A) \xrightarrow{M} (q, bv, B\alpha) \wedge (q, bv, B\alpha) \xrightarrow{M} (q, v, \beta\alpha) \\ \Leftrightarrow & (q, uv, A) \xrightarrow{M}^{n+1} (q, v, \gamma) \end{aligned}$$

□

196

$$A \xrightarrow{G} u\gamma \text{ mit Linksableitung gdw } (q, uv, A) \xrightarrow{M} (q, v, \gamma)$$

Satz 3.56

$$L(G) = L_\epsilon(M)$$

Beweis:

$$u \in L(G)$$

$$\Leftrightarrow S \xrightarrow{G} u \text{ mit Linksableitung}$$

$$\Leftrightarrow (q, u, S) \xrightarrow{M} (q, \epsilon, \epsilon)$$

$$\Leftrightarrow u \in L_\epsilon(M)$$

□

197

Satz 3.57 (PDA \rightarrow CFG)

Zu jedem PDA $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta)$, der mit leerem Keller akzeptiert, kann man eine CFG G konstruieren mit $L(G) = L_\epsilon(M)$.

Konstruktion: $G := (V, \Sigma, P, S)$ mit

$V := Q \times \Gamma \times Q \cup \{S\}$ wobei wir die Tripel mit $[., ., .]$ notieren

und P die folgenden Produktionen enthält:

- $S \rightarrow [q_0, Z_0, q]$ für alle $q \in Q$
- Für alle $\delta(q, b, Z) \ni (r_0, Z_1 \dots Z_k)$ und für alle $r_1, \dots, r_k \in Q$:

$$[q, Z, r_k] \rightarrow b[r_0, Z_1, r_1][r_1, Z_2, r_2] \dots [r_{k-1}, Z_k, r_k]$$

Idee: $[q, Z, r_k] \xrightarrow{G} w$ gdw $(q, w, Z) \xrightarrow{M} (r_k, \epsilon, \epsilon)$

Die r_1, \dots, r_k sind potenzielle Zwischenzustände beim Akzeptieren der Teilwörter von $bu_1 \dots u_k = w$, die zu Z_1, \dots, Z_k gehören.
(Zerlegungssatz!)

198

Lemma 3.58

$$[q, Z, p] \xrightarrow{G} w \text{ gdw } (q, w, Z) \xrightarrow{M} (p, \epsilon, \epsilon)$$

Beweis:

„ \Rightarrow “: Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Ableitung $[q, Z, p] \xrightarrow{G} w$ muss von folgender Form sein:

$$\begin{array}{lcl} [q, Z, r_k] & \xrightarrow{G} & b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ & \xrightarrow{G}^{n-1} & bu_1 \dots u_k = w \end{array}$$

mit $r_k = p$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$,

$[r_{i-1}, Z_i, r_i] \xrightarrow{G}^{n_i} u_i$ ($i = 1, \dots, k$) und $\sum n_i = n - 1$.

$$\text{IA} \Rightarrow (r_{i-1}, u_i, Z_i) \xrightarrow{M}^{n_i} (r_i, \epsilon, \epsilon)$$

Erweiterungslemma

$$\Rightarrow (r_{i-1}, u_i \dots u_k, Z_i \dots Z_k) \xrightarrow{M}^{n_i} (r_i, u_{i+1} \dots u_k, Z_{i+1} \dots Z_k)$$

$$\Rightarrow (q, w, Z) \xrightarrow{M} (r_0, u_1 \dots u_k, Z_1 \dots Z_k) \xrightarrow{M}^{n-1} (r_k, \epsilon, \epsilon)$$

□

199

Beweis (Forts.):

„ $[q, Z, p] \xrightarrow{G} w \Leftarrow (q, w, Z) \xrightarrow{M} (p, \epsilon, \epsilon)$ “: Mit Induktion über n .

IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n .

Die Berechnung $(q, w, Z) \xrightarrow{M} (p, \epsilon, \epsilon)$ muss von dieser Form sein:

$$(q, w, Z) \xrightarrow{M} (r_0, w', Z_1 \dots Z_k) \xrightarrow{M}^{n-1} (p, \epsilon, \epsilon)$$

mit $w = bw'$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$.

Nach dem Zerlegungssatz muss es r_i , u_i und n_i geben mit

$$(r_{i-1}, u_i, Z_i) \xrightarrow{M}^{n_i} (r_i, \epsilon, \epsilon) \quad (i = 1, \dots, k)$$

und $w' = u_1 \dots u_k$, $\sum n_i = n - 1$.

$$\text{IA} \Rightarrow [r_{i-1}, Z_i, r_i] \xrightarrow{G}^{n_i} u_i$$

$$\begin{aligned} \Rightarrow [q, Z, p] & \xrightarrow{G} b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ & \xrightarrow{G}^{n-1} bu_1 \dots u_k = w \end{aligned}$$

□

200

Damit haben wir bewiesen:

Satz 3.59

Eine Sprache ist kontextfrei gdw sie von einem Kellerautomaten akzeptiert wird.

201

3.10 Deterministische Kellerautomaten

Definition 3.60

Ein PDA heißt **deterministisch (DPDA)** gdw für alle $q \in Q$, $a \in \Sigma$ und $Z \in \Gamma$

$$|\delta(q, a, Z)| + |\delta(q, \epsilon, Z)| \leq 1$$

Beispiel 3.61

Der folgende DPDA mit $\Sigma = \{0, 1, \$\}$, $\Gamma = \{Z_0, 0, 1\}$ akzeptiert die Sprache

202

Definition 3.62

Eine CFL heißt **deterministisch (DCFL)** gdw sie von einem DPDA akzeptiert wird.

Man kann zeigen: Die CFL $\{ww^R \mid w \in \{0,1\}^*\}$ ist nicht deterministisch.

Da man jeden DFA leicht mit einem DPDA simulieren kann:

Fakt 3.63

Jede reguläre Sprache ist eine DCFL.

203

Eine Sprache erfüllt die **Präfix Bedingung** gdw wenn sie keine zwei Wörter enthält, so dass das eine ein **echtes Präfix** des anderen ist.

Beispiel 3.64

- Die Sprache a^* erfüllt die Präfix Bedingung nicht.
- Die Sprache $\{w\$w^R \mid w \in \{0,1\}^*\}$ erfüllt die Präfix Bedingung.

Lemma 3.65

$$\exists \text{DPDA } M. L = L_\epsilon(M) \Leftrightarrow$$

$\exists \text{DPDA } M. L = L_F(M)$ und L erfüllt die Präfix Bedingung

Beweis: Übung!

Es gibt also einen DPDA M mit $L_F(M) = L(a^*)$ aber keinen DPDA mit $L_\epsilon(M) = L(a^*)$.

204

Satz 3.66

Die Klasse der DCFLs ist unter Komplement abgeschlossen.

Beweis: Komplex. Siehe zB Erk&Priese oder Kozen.

Da die CFLs *nicht* unter Komplement abgeschlossen sind:

Korollar 3.67

Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in $\text{CFL} \setminus \text{DCFL}$:

Sei $L_{ww} := \{ww \mid w \in \{a, b\}^*\}$.

Dann ist $L := \{a, b\}^* \setminus L_{ww}$ eine CFL aber keine DCFL.

L wird von folgender CFG erzeugt (ohne Beweis):

$$\begin{aligned} S &\rightarrow AB \mid BA \mid A \mid B \\ A &\rightarrow CAC \mid a \quad B \rightarrow CBC \mid b \quad C \rightarrow a \mid b \end{aligned}$$

Wäre L eine DCFL, dann auch $\overline{L} = L_{ww}$ (wegen Satz 3.66).

Aber mit dem Pumping Lemma kann man zeigen,
dass L_{ww} keine CFL ist [HMU], und daher erst recht keine DCFL.

205

Lemma 3.68

Die Klasse der DCFLs ist weder unter Schnitt noch unter Vereinigung abgeschlossen.

Beweis:

Erinnerung: CFLs nicht unter Schnitt abgeschlossen:

$$\begin{aligned} L_1 &:= \{a^i b^i c^j \mid i, j \in \mathbb{N}\} \text{ ist kontextfrei} \\ L_2 &:= \{a^i b^j c^j \mid i, j \in \mathbb{N}\} \text{ ist kontextfrei} \\ L_1 \cap L_2 &= \{a^i b^i c^i \mid i \in \mathbb{N}\} \text{ ist nicht kontextfrei} \end{aligned}$$

Aber L_1 und L_2 sind sogar DCFLs.

Da $L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$ können die DCFLs auch nicht unter Vereinigung abgeschlossen sein. □

206

Lemma 3.69

Jede DCFL ist nicht inhärent mehrdeutig, dh sie wird von einer nicht-mehrdeutigen Grammatik erzeugt.

Beweisidee: Die Konversion PDA → CFG erzeugt aus einem DPDA eine nicht-mehrdeutige CFG. [HMU]

Satz 3.70

Das Wortproblem für DCFLs ist in linearer Zeit lösbar.

Mehr Information: Vorlesungen zum Übersetzerbau

207

3.11 Tabellarischer Überblick

Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Regulär	ja	ja	ja	ja	ja
DCFL	nein	nein	ja	nein	nein
CFL	nein	ja	nein	ja	ja

Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
DFA	$O(n)$	ja	ja	ja
DPDA	$O(n)$	ja	ja	nein(*)
CFG	$O(n^3)$	ja	nein(*)	nein(*)

Sénizergues (1997), Stirling (2001)

(*) Vorschau

208

3.12 Die Chomsky-Hierarchie

Definition 3.71

Eine **Grammatik** G ist ein 4-Tupel (V, Σ, P, S) , wobei P eine endliche Teilmenge von $(V \cup \Sigma)^+ \times (V \cup \Sigma)^+$ ist, dh jede Produktion ist von der Form $\alpha \rightarrow \beta$.

G ist vom

Typ 0

Typ 1 falls $|\alpha| \leq |\beta|$

Typ 2 falls G vom Typ 1 ist und $\alpha \in V$

Typ 3 falls G vom Typ 2 ist und $\beta \in \Sigma \cup \Sigma V$.

Zusätzlich erlaubt man noch $S \rightarrow \epsilon$.

Offensichtlich gilt:

$$\text{Typ 3} \subset \text{Typ 2} \subset \text{Typ 1} \subset \text{Typ 0}$$

209

Grammatiken, Maschinen und Sprachklassen:

Typ 3	Rechtslineare Grammatik Endlicher Automat
Typ 2	kontextfreie Grammatik Kellerautomat
Typ 1	Kontextsensitive Grammatik Linear beschränkter Automat
Typ 0	Chomsky-Grammatik, Phrasenstrukturgrammatik Turingmaschine

Satz 3.72

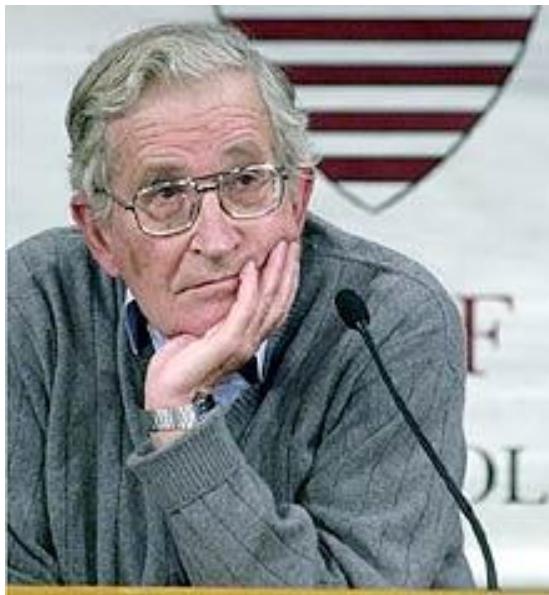
$$L(\text{Typ 3}) \subset L(\text{Typ 2}) \subset L(\text{Typ 1}) \subset L(\text{Typ 0})$$

210



Noam Chomsky.

Three Models for the Description of Language. Transactions on Information Theory, 113-124, 1956.



Noam Chomsky (* 7. Dezember 1928) ist Professor für Linguistik am MIT und ein bedeutender Sprachwissenschaftler. Er entwickelte die Chomsky-Hierarchie. Neben seiner linguistischen Arbeit gilt Chomsky als einer der bedeutendsten Intellektuellen Nordamerikas und ist als scharfer Kritiker der US-amerikanischen Außenpolitik bekannt. Seine politische Heimat ist der Anarchosyndikalismus.
[Wikipedia]

211

Kapitel II Berechenbarkeit, Entscheidbarkeit, Komplexität

Überblick:

- Was kann man berechnen? Dh:
Welche Funktionen kann man in endlicher Zeit berechnen?
- Mit welchen Sprachen/Maschinen?
- Welche Eigenschaften von Programmen sind entscheidbar?
ZB Termination? Code-Erreichbarkeit? Überlauf?
- Was kann man in polynomieller Zeit berechnen?
Mit welchen Maschinen?

212

4. Berechenbarkeit und Entscheidbarkeit

4.1 Der Begriff der Berechenbarkeit

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist intuitiv berechenbar, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, \dots, n_k) \in \mathbb{N}^k$

- nach endlich vielen Schritten mit Ergebnis $f(n_1, \dots, n_k)$ hält, falls $f(n_1, \dots, n_k)$ definiert ist,
- und nicht terminiert, falls $f(n_1, \dots, n_k)$ nicht definiert ist.

Was bedeutet „Algorithmus“? Assembler? C? Java? OCaml?
Macht es einen Unterschied?

Achtung: Berechenbarkeit setzt zwei verschiedene Dinge in Beziehung:

- Algorithmen, dh endliche Wörter.
- Mathematische Funktionen, dh Mengen von Paaren.

ZB beschreibt $x \mapsto x^2$ die Funktion

$$\{(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), \dots\}$$

213

Terminologie: Eine Funktion $f : A \rightarrow B$ ist

total gdw $f(a)$ für alle $a \in A$ definiert ist.

partiell gdw $f(a)$ auch undefiniert sein kann.

echt partiell gdw sie nicht total ist.

Beispiel 4.1

Der Algorithmus

```
input(n);  
while true do ;
```

berechnet die überall undefinierte Funktion, dh $\emptyset \subset \mathbb{N} \rightarrow \mathbb{N}$.

214

Beispiel 4.2

Ist die Funktion

$$f_1(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_1(31415) = 1$ aber $f_1(315) = 0$)

Ja, denn π kann iterativ auf beliebig viele Dezimalstellen genau berechnet werden.

215

Beispiel 4.3

Ist die Funktion

$$f_2(n) := \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar? (Bsp: $f_2(415) = 1$) Unbekannt!

Durch schrittweise Approximation und Suche in der Dezimalbruchentwicklung von π kann man feststellen, dass n vorkommt.
Aber wie stellt man fest, dass n nicht vorkommt?

Nichttermination statt 0!

Vielleicht gibt es aber einen (noch zu findenden) mathematischen Satz, der genaue Aussagen über die in π vorkommenden Ziffernfolgen macht.

216

Beispiel 4.4

Ist die Funktion

$$f_3(n) := \begin{cases} 1 & \text{falls mindestens } n \text{ mal hintereinander irgendwo in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ eine } 0 \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

berechenbar?

Ja, denn

- entweder kommt 0^n für beliebig große n vor,
dann ist $f_3(n) = 1$ für alle n ,
- oder es gibt eine längste vorkommende Sequenz 0^m ,
dann ist $f_3(n) = 1$ für $n \leq m$ und $f_3(n) = 0$ sonst.

Beide Funktionen sind berechenbar.

Dies ist ein **nicht-konstruktiver Beweis**:

Wir wissen, es gibt einen Algorithmus, der f_3 berechnet,
aber wir wissen nicht, welcher.

217

Satz 4.5

Es gibt nicht-berechenbare Funktionen in $\mathbb{N} \rightarrow \{0, 1\}$.

Beweis:

Es gibt nur abzählbar viele Algorithmen, aber überabzählbar viele
Funktionen in $\mathbb{N} \rightarrow \{0, 1\}$ (Beweis wie zu Satz 1.8).

□

218

Verschiedene Formalisierungen des Begriffs der Berechenbarkeit:

- Turing-Maschinen ([Turing](#) 1936)
- λ -Kalkül ([Church](#) 1936)
- μ -rekursive Funktionen
- Markov-Algorithmen
- Registermaschinen
- Awk, Basic, C, Dylan, Eiffel, Fortran, Java, Lisp, Modula, Oberon, Pascal, Python, Ruby, Simula, \TeX , ...-Programme
- while-Programme
- goto-Programme
- DNA-Algorithmen
- Quantenalgorithmen
- ...

219

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

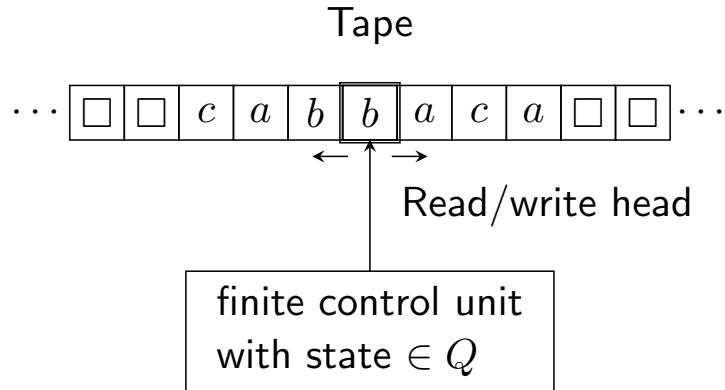
Churchsche These / Church-Turing These

Der formale Begriff der Berechenbarkeit mit Turing-Maschinen (bzw λ -Kalkül etc) stimmt mit dem intuitiven Berechenbarkeitsbegriff überein.

Ist formal prinzipiell nicht beweisbar.

220

4.2 Turingmaschinen



221

Definition 4.6

Eine **Turingmaschine (TM)** ist ein 7-Tupel

$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ so dass

- Q ist eine endliche Menge von **Zuständen**.
- Σ ist eine endliche Menge, das **Eingabealphabet**.
- Γ ist eine endliche Menge, das **Bandalphabet**, mit $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$ ist die **Übergangsfunktion**.
- δ darf partielle sein.
- $q_0 \in Q$ ist der **Startzustand**.
- $\square \in \Gamma \setminus \Sigma$ ist das **Leerzeichen**.
- $F \subseteq Q$ ist die Menge der **akzeptierenden oder Endzustände**.

Eine **nichtdeterministische** Turingmaschine hat eine Übergangsfunktion $\delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$.

222

Intuitiv bedeutet $\delta(q, a) = (q', b, d)$:

- Wenn sich M im Zustand q befindet,
- und auf dem Band a liest,
- so geht M im nächsten Schritt in den Zustand q' über,
- überschreibt a mit b ,
- und bewegt danach den Schreib-/Lesekopf nach **rechts** (falls $d = R$), nach **links** (falls $d = L$), oder **nicht** (falls $d = N$).

223

Definition 4.7

Eine **Konfiguration** einer Turingmaschine ist ein Tripel $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$.

Dies modelliert

- Bandinhalt: $\dots \square \alpha \beta \square \dots$
- Zustand: q
- Kopf auf dem ersten Zeichen von $\beta \square$

Die **Startkonfiguration** der Turingmaschine bei Eingabe $w \in \Sigma^*$ ist (ϵ, q_0, w) .

224

Die Berechnung der TM M wird als Relation \rightarrow_M auf Konfigurationen formalisiert. Falls $\delta(q, \text{first}(\beta)) = (q', c, d)$:

$$(\alpha, q, \beta) \rightarrow_M \begin{cases} (\alpha, q', c, \text{rest}(\beta)) & \text{falls } d = N \\ (\alpha c, q', \text{rest}(\beta)) & \text{falls } d = R \\ (\text{butlast}(\alpha), q', \text{last}(\alpha) c, \text{rest}(\beta)) & \text{falls } d = L \end{cases}$$

wobei

$$\begin{array}{ll} \text{first}(aw) = a & \text{first}(\epsilon) = \square \\ \text{rest}(aw) = w & \text{rest}(\epsilon) = \epsilon \\ \text{last}(wa) = a & \text{last}(\epsilon) = \square \\ \text{butlast}(wa) = w & \text{butlast}(\epsilon) = \epsilon \end{array}$$

für $a \in \Gamma$ und $w \in \Gamma^*$.

Falls M nichtdeterministisch ist: $\delta(q, \text{first}(\beta)) \ni (q', c, d)$

225

Beispiel 4.8 (Unär +1)

$$M = (\{q, f\}, \{1\}, \{1, \square\}, \delta, q, \square, \{f\})$$

$$\begin{aligned} \delta(q, \square) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

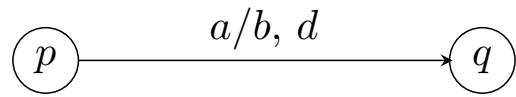
- Beispieldurchlauf:

$$(\epsilon, q, 11) \rightarrow_M$$

- Welche Eingaben führen von q nach f ?

226

Eine graphische Notation für $\delta(p, a) = (q, b, d)$:



227

Beispiel 4.9 (Binär +1)

ZB 1011 \mapsto 1100

$$M = (\{q_0, q_1, q_2, q_f\}, \{0, 1\}, \{0, 1, \square\}, \delta, q_0, \square, \{q_f\})$$

wobei

$$\begin{array}{lll} \delta(q_0, 0) = (q_0, 0, R) & \delta(q_1, 1) = (q_1, 0, L) & \delta(q_2, 0) = (q_2, 0, L) \\ \delta(q_0, 1) = (q_0, 1, R) & \delta(q_1, 0) = (q_2, 1, L) & \delta(q_2, 1) = (q_2, 1, L) \\ \delta(q_0, \square) = (q_1, \square, L) & \delta(q_1, \square) = (q_f, 1, N) & \delta(q_2, \square) = (q_f, \square, R) \end{array}$$

Beispiellauf:

$$\begin{aligned} (\epsilon, q_0, 101) &\rightarrow_M (1, q_0, 01) \rightarrow_M (10, q_0, 1) \rightarrow_M (101, q_0, \epsilon) \rightarrow_M \\ (10, q_1, 1\square) &\rightarrow_M (1, q_1, 00\square) \rightarrow_M \\ (\epsilon, q_2, 110\square) &\rightarrow_M (\epsilon, q_2, \square110\square) \rightarrow_M \\ (\square, q_f, 110\square) & \end{aligned}$$

228

Definition 4.10

Eine Turingmaschine M akzeptiert die Sprache

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F, \alpha, \beta \in \Gamma^*. (\epsilon, q_0, w) \xrightarrow{M}^* (\alpha, q, \beta)\}$$

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heißt Turing-berechenbar gdw es eine Turingmaschine M gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt

$$\begin{aligned} f(n_1, \dots, n_k) = m &\iff \\ \exists r \in F. (\epsilon, q_0, bin(n_1)\#bin(n_2)\#\dots\#bin(n_k)) \\ &\xrightarrow{M}^* (\square \dots \square, r, bin(m)\square \dots \square) \end{aligned}$$

wobei $bin(n)$ die Binärdarstellung der Zahl n ist.

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ heißt Turing-berechenbar gdw es eine Turingmaschine M gibt, so dass für alle $u, v \in \Sigma^*$ gilt

$$f(u) = v \iff \exists r \in F. (\epsilon, q_0, u) \xrightarrow{M}^* (\square \dots \square, r, v \square \dots \square)$$

Zum Halten/Terminieren von TM

OE(!) können wir festlegen, dass eine TM in der Konfiguration (α, q, β) hält falls $q \in F$. Dann gibt es keine Konfiguration (α', q', β') mit $(\alpha, q, \beta) \xrightarrow{M} (\alpha', q', \beta')$.

Ist δ partiell, so kann eine TM auch halten, bevor sie einen Endzustand erreicht.

Daher können wir auch annehmen, dass $\delta(q, a)$ undefiniert (bzw \emptyset) ist falls $q \in F$.

Satz 4.11

Zu jeder nichtdeterministischen TM N gibt es eine deterministische TM M mit $L(N) = L(M)$.

Beweis:

M durchsucht den Baum der Berechnungen von N in Breitensuche, beginnend mit der Startkonfiguration (ϵ, q_0, w) , eine Ebene nach der anderen.

Gibt es in dem Baum (auf Ebene n) eine Konfigurationen mit Endzustand, so wird diese (nach Zeit $O(c^n)$) gefunden. □

231

Satz 4.12

Die von Turingmaschinen akzeptierten Sprachen sind genau die Typ-0-Sprachen der Chomsky Hierarchie.

Beweis:

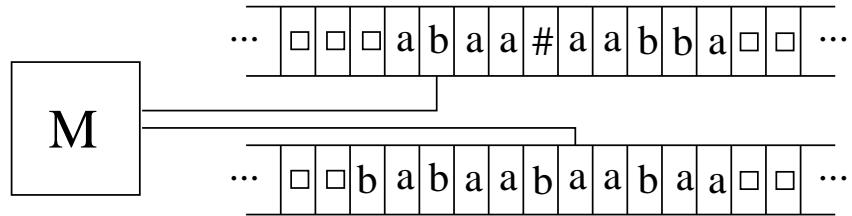
Wir beschreiben nur die Beweisidee. (Mehr Details: [Schöning]).

„ \Rightarrow “: Grammatikregeln können direkt die Rechenregeln einer TM simulieren.

„ \Leftarrow “: Die (nichtdeterministische) TM versucht von ihrer Eingabe aus das Startsymbol der Grammatik zu erreichen, indem sie die Produktionen der Grammatik von rechts nach links anwendet, (nichtdeterministisch) an jeder möglichen Stelle. □

232

Eine beliebte Modellvariante ist die k -Band-Turingmaschine:



Die k Köpfe sind völlig unabhängig voneinander:

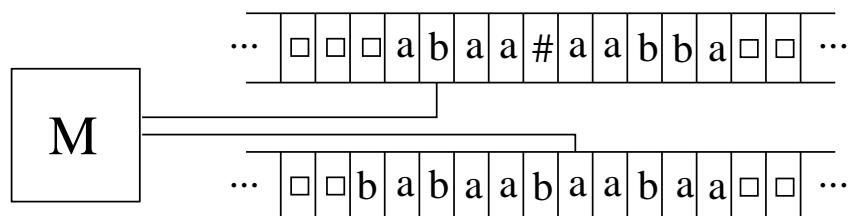
$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, N\}^k$$

233

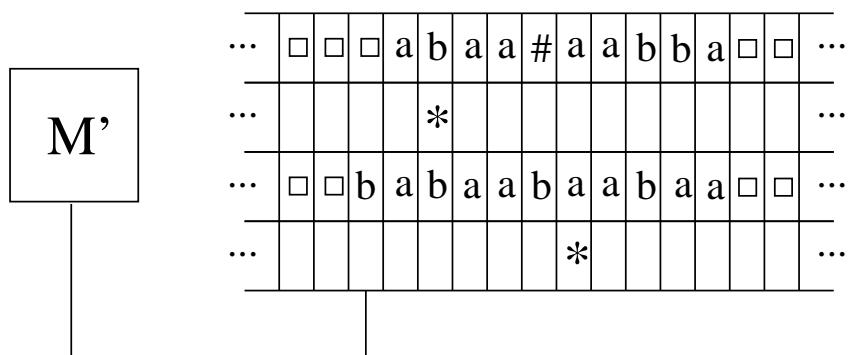
Satz 4.13

Jede k -Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Beweisidee: Aus



wird



234

Beweisskizze:

- $\Gamma' := (\Gamma \times \{\star, \square\})^k$
- M' simuliert einen M -Schritt durch mehrere Schritte:
 - startet mit Kopf links von allen \star .
 - geht nach rechts bis alle \star überschritten sind, und merkt sich dabei (in Q') die Zeichen über jedem \star .
 - hat jetzt alle Information, um δ_M anzuwenden.
 - geht nach links über alle \star hinweg und führt dabei δ_M aus.

Beobachtung:

n Schritte von M lassens sich durch
 $O(n^2)$ Schritte von M' simulieren.

Denn nach n Schritten von M trennen $\leq 2n$ Felder linkesten und rechtesten Kopf. Obige Simulation eines M -Schritts braucht daher $O(n)$ M' -Schritte. Simulation von n Schritten: $O(n^2)$ Schritte.

235

4.3 Programmieren von Mehrband-TM

Die folgenden Basismaschinen sind leicht programmierbar:

- Band $i :=$ Band $i + 1$
- Band $i :=$ Band $i - 1$
- Band $i := 0$
- Band $i :=$ Band j

236

Seien $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_i, \square, F_i)$, $i = 1, 2$.

Die **sequentielle Komposition** (Hintereinanderschaltung) von M_1 und M_2 bezeichnen wir mit

$$\longrightarrow M_1 \longrightarrow M_2 \longrightarrow$$

Sie ist wie folgt definiert:

$$M := (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, \square, F_2)$$

wobei (oE) $Q_1 \cap Q_2 = \emptyset$ und

$$\delta := \delta_1 \cup \delta_2 \cup \{(f_1, a) \mapsto (q_2, a, N) \mid f_1 \in F_1, a \in \Gamma_1\}$$

237

Sind f_1 und f_2 Endzustände von M so bezeichnet

$$\begin{array}{ccccc} \longrightarrow & M & \xrightarrow{f_1} & M_1 & \longrightarrow \\ & & \downarrow f_2 & & \\ & & M_2 & & \\ & & \downarrow & & \end{array}$$

eine **Fallunterscheidung**, dh eine TM, die vom Endzustand f_1 von M nach M_1 übergeht, und von f_2 aus nach M_2 .

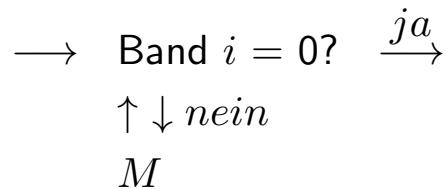
Die folgende TM nennen wir „Band=0?“ (bzw „Band $i = 0?$ “):

$$\begin{aligned} \delta(q_0, 0) &= (q_0, 0, R) \\ \delta(q_0, \square) &= (ja, \square, L) \\ \delta(q_0, a) &= (nein, a, N) \quad \text{für } a \neq 0, \square \end{aligned}$$

wobei *ja* und *nein* Endzustände sind.

238

Analog zur Fallunterscheidung kann man auch eine TM für eine **Schleife** konstruieren



die sich wie `while Band $i \neq 0$ do M` verhält.

Moral: Mit TM kann man imperativ programmieren:

```
:=
;
if
while
```

239

4.4 LOOP-, WHILE- und GOTO-Berechenbarkeit

LOOP, WHILE	\equiv strukturierte Programme mit <code>for/while</code> -Schleifen
GOTO	\equiv Assembler

Syntax von LOOP-Programmen:

$$P \rightarrow X := X + C \\ | \quad X := X - C \\ | \quad P; P \\ | \quad \text{IF } X = 0 \text{ DO } P \text{ ELSE } Q \text{ END} \\ | \quad \text{LOOP } X \text{ DO } P \text{ END}$$

wobei X eine der Variablen x_0, x_1, \dots
 und C eine der Konstanten $0, 1, \dots$ sein kann.

Beispiel 4.14

`LOOP x_2 DO $x_1 := x_0+1$ END`

240

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von LOOP-Programmen (informell):

$x_i := x_j + n$ Neuer Wert von x_i ist $x_j + n$.

$x_i := x_j - n$ Neuer Wert von x_i ist $x_j \dot{-} n$.

$P_1; P_2$ Führe zuerst P_1 und dann P_2 aus.

LOOP x_i DO P END Führe P genau n mal aus, wobei n der Anfangswert von x_i ist. Zuweisungen an x_i in P ändern die Anzahl n der Schleifendurchläufe nicht.

Beispiel 4.15

LOOP x_0 DO $x_1 := x_1 + 1$ END simuliert

Zu Beginn der Ausführung stehen die Eingaben in x_1, \dots, x_k . Alle anderen Variablen sind 0. Die Ausgabe wird in x_0 berechnet.

241

Definition 4.16

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **LOOP-berechenbar** gdw es ein LOOP-Programm P gibt, so dass für alle $n_1, \dots, n_k \in \mathbb{N}$:

P , gestartet mit n_1, \dots, n_k in x_1, \dots, x_k (0 in den anderen Var.) terminiert mit $f(n_1, \dots, n_k)$ in x_0 .

Lemma 4.17

Alle LOOP-berechenbaren Funktionen sind total.

Beweis mit Induktion über die Erzeugung/Struktur der LOOP-Programme.

Sind alle totalen Funktionen LOOP-berechenbar?

242

Syntaktische Abkürzungen („Zucker“):

- $x_i := x_j \equiv x_i := x_j + 0$
- $x_i := n \equiv x_i := x_j + n$
(wobei an x_j nirgends zugewiesen wird)
- $x_i := x_j + x_k \equiv x_i := x_j; \text{LOOP } x_k \text{ DO } x_i := x_i + 1$
(falls $i \neq k$)
- $x_i := x_j * x_k \equiv x_i := 0; \text{LOOP } x_k \text{ DO } x_i := x_i + x_j$
(falls $i \neq j, k$)
- DIV, MOD, ...
- $x := \text{komplexer Ausdruck}$
- IF $x = 0$ THEN P END

243

WHILE-Programme sind LOOP-Programme erweitert um WHILE-Schleifen:

$$P \rightarrow \text{WHILE } X \neq 0 \text{ DO } P \text{ END}$$

Die Semantik ist wie üblich.

Fakt 4.18

WHILE-Schleifen können LOOP-Schleifen simulieren.

Fakt 4.19

LOOP-Schleifen können WHILE-Schleifen nicht simulieren.

244

Definition 4.20

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \dots, n_k \in \mathbb{N}$:

P , gestartet mit n_1, \dots, n_k in x_1, \dots, x_k (0 in den anderen Var.)

- terminiert mit $f(n_1, \dots, n_k)$ in x_0 , falls $f(n_1, \dots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \dots, n_k)$ undefiniert ist.

Turingmaschinen können WHILE-Programme simulieren:

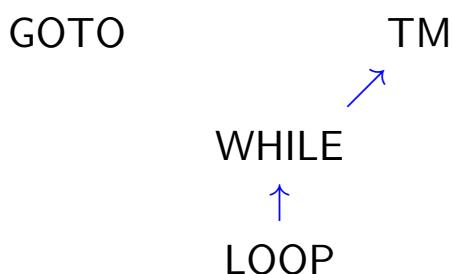
Satz 4.21 (WHILE \rightarrow TM)

Jede WHILE-berechenbare Funktion ist auch Turing-berechenbar.

Beweis:

Jede Programmvariable wird auf einem eigenen Band gespeichert. Wir haben bereits gezeigt: Alle Konstrukte der WHILE-Sprache können von einer Mehrband-TM simuliert werden, und eine Mehrband-TM kann von einer 1-Band TM simuliert werden. \square

245



Übersetzungen

246

Ein **GOTO-Programm** ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$$

(wobei alle Marken verschieden und optional sind)

Mögliche Anweisungen A_i sind:

- $x_i := x_j + n$
- $x_i := x_j - n$
- GOTO M_i
- IF $x_i = n$ GOTO M_j
- HALT

Die Semantik ist wie erwartet.

Fakt 4.22 (WHILE \rightarrow GOTO)

Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden.

247

Satz 4.23 (GOTO \rightarrow WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Beweis: Simuliere $M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$ durch

```

pc := 1;
WHILE pc ≠ 0 DO
    IF pc = 1 THEN P1 ELSE
        :
    IF pc = k THEN Pk ELSE pc := 0
END

```

wobei $A_i \mapsto P_i$ wie folgt definiert ist:

$x_i := x_j +/− n$	\mapsto	$x_i := x_j +/− n; pc := pc+1$
GOTO M_i	\mapsto	$pc := i$
IF $x_j = 0$ GOTO M_i	\mapsto	IF $x_j = 0$ THEN $pc:=i$ ELSE $pc:=pc+1$ END
HALT	\mapsto	$pc := 0$

□

248

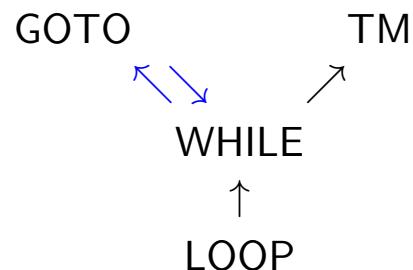
Korollar 4.24

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 4.25 (Kleenesche Normalform)

Jedes WHILE-Programm ist zu einem WHILE-Programm mit genau einer WHILE-Schleife äquivalent.

249



Übersetzungen

250

Satz 4.26 ($\text{TM} \rightarrow \text{GOTO}$)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung einer TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ in ein GOTO-Programm. Zeichen und Zeichenreihen werden als Zahlen kodiert.

$$Q = \{q_0, \dots, q_k\} \quad \Gamma = \{a_0 (= \square), \dots, a_n\}$$

Eine Konfiguration

$$(a_{i_p} \dots a_{i_1}, \ q_l, \ a_{j_1} \dots a_{j_q})$$

wird durch die Programmvariablen x, y, z wie folgt repräsentiert:

$$x = (i_p \dots i_1)_b, \quad y = (j_q \dots j_1)_b, \quad z = l$$

wobei $(i_p \dots i_1)_b$ die Zahl $i_p \dots i_1$ zur Basis $b := n + 1$ ist:

$$x = \sum_{r=1}^p i_r b^{r-1}$$

251

Der Kern des GOTO-Programms ist die iterierte Simulation von δ :

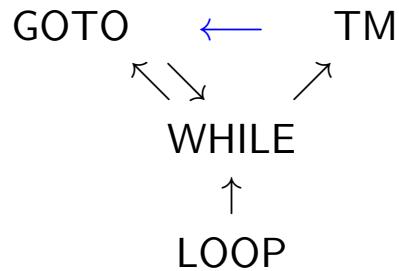
```

 $M:$       IF  $z \in F$  GOTO  $M_{end};$ 
           $a := y \bmod b;$ 
          IF  $z = 0$  AND  $a = 0$  GOTO  $M_{00};$ 
          IF  $z = 0$  AND  $a = 1$  GOTO  $M_{01};$ 
          ...
          IF  $z = k$  AND  $a = n$  GOTO  $M_{kn};$ 
 $M_{00}:$     $P_{00};$  GOTO  $M;$ 
 $M_{01}:$     $P_{01};$  GOTO  $M;$ 
          ...
 $M_{kn}:$     $P_{kn};$  GOTO  $M$ 
```

wobei P_{ij} die Simulation von $\delta(q_i, a_j) = (q_r, a_s, d)$ ist. Für $d = L$:

$z := r;$	Zustand aktualisieren
$y := y \bmod b;$	Löschen von a_j
$y := b*y + s;$	Schreiben von a_s
$y := b*y + (x \bmod b);$	Bewegung L (I): Einfügen von a_{i_1} in y
$x := x \bmod b$	Bewegung L (II): Löschen von a_{i_1} aus x

252



Übersetzungen

253

4.5 Primitiv rekursive Funktionen

Klassen von Programmiersprachen

Imperativ	Funktional
C, Java, ...	OCaml, Lisp, ...
TM, WHILE, GOTO	μ -rekursiv
LOOP	Primitiv rekursiv

Wir betrachten Funktionen $\mathbb{N}^k \rightarrow \mathbb{N}$, $k \geq 0$.

Wir identifizieren $\mathbb{N}^0 \rightarrow \mathbb{N}$ mit \mathbb{N} und $c()$ mit c .

Definition der primitiv rekursiven Funktionen:

Fixe Basisfunktionen zB $s(x) = x + 1$

Funktionskomposition zB $f(x, y) = g(x, h(x, y))$

Fixe Art der Rekursion zB $f(0) = 1$
 $f(n + 1) = n * f(n)$

254

Definition 4.27 (Basisfunktionen)

- Die konstante Funktion 0
- Die Nachfolgerfunktion $s(n) = n + 1$
- Die Projektionsfunktionen $\pi_i^k : \mathbb{N}^k \rightarrow \mathbb{N}$, $1 \leq i \leq k$:

$$\pi_i^k(x_1, \dots, x_k) = x_i$$

Definition 4.28

Die Komposition von g und h_1, \dots, h_k erzeugt die Funktion f

$$f(\bar{x}) = g(h_1(\bar{x}), \dots, h_k(\bar{x}))$$

wobei $\bar{x} = (x_1, \dots, x_n)$ und

$$\begin{aligned} f &: \mathbb{N}^n \rightarrow \mathbb{N} \\ g &: \mathbb{N}^k \rightarrow \mathbb{N} \\ h_i &: \mathbb{N}^n \rightarrow \mathbb{N} \quad (i = 1, \dots, k) \end{aligned}$$

255

Definition 4.29

Das Schema der primitiven Rekursion erzeugt aus g und h die Funktion f

$$\begin{aligned} f(0, \bar{x}) &= g(\bar{x}) \\ f(m + 1, \bar{x}) &= h(f(m, \bar{x}), m, \bar{x}) \end{aligned}$$

wobei $\bar{x} = (x_1, \dots, x_n)$ und

$$\begin{aligned} f &: \mathbb{N}^{n+1} \rightarrow \mathbb{N} \\ g &: \mathbb{N}^n \rightarrow \mathbb{N} \\ h &: \mathbb{N}^{n+2} \rightarrow \mathbb{N} \end{aligned}$$

256

Definition 4.30 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \rightarrow \mathbb{N}$, $k \geq 0$:

- Die Basisfunktionen 0 , s , und π_i^k sind primitiv rekursiv.
- Sind g und h_i primitiv rekursiv, dann auch ihre Komposition

$$f(\bar{x}) = g(h_1(\bar{x}), \dots, h_k(\bar{x}))$$

- Sind g und h primitiv rekursiv, dann auch die mit primitiver Rekursion definierte Funktion

$$\begin{aligned} f(0, \bar{x}) &= g(\bar{x}) \\ f(m+1, \bar{x}) &= h(f(m, \bar{x}), m, \bar{x}) \end{aligned}$$

Lemma 4.31

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau der primitiv-rekursiven Funktionen. □

257

Beispiel 4.32

Alle Konstanten $n \in \mathbb{N}$ sind PR: $1 = s(0)$, $2 = s(1)$, ...

Addition ist PR:

$$\begin{aligned} h(r, x, y) &= s(\pi_1^3(r, x, y)) \\ add(0, y) &= \pi_1^1(y) \\ add(x + 1, y) &= h(add(x, y), x, y) \end{aligned}$$

Lesbarer, aber nicht dem **syntaktischen PR-Format** entsprechend:

$$\begin{aligned} add(0, y) &= y \\ add(x + 1, y) &= s(add(x, y)) \end{aligned}$$

Streng genommen also kein Nachweis, dass Addition PR ist.

258

Beispiel 4.33

Multiplikation ist PR:

$$\begin{aligned} k(y) &= 0 \\ h(r, x, y) &= add(\pi_1^3(r, x, y), \pi_3^3(r, x, y)) \\ mult(0, y) &= k(y) \\ mult(x + 1, y) &= h(mult(x, y), x, y) \end{aligned}$$

Lesbarer, aber nicht dem **syntaktischen PR-Format** entsprechend:

$$\begin{aligned} mult(0, y) &= 0 \\ mult(x + 1, y) &= add(mult(x, y), y) \end{aligned}$$

In Zukunft: $+$ und $*$ statt add und $mult$.

259

Definition 4.34

Wir bezeichnen f als eine **erweiterte Komposition** der Funktionen g_1, \dots, g_k falls

$$f(x_1, \dots, x_n) = t$$

so dass t ein Ausdruck ist, der nur aus den Funktionen g_1, \dots, g_k und den Variablen x_1, \dots, x_n besteht.

Beispiel: $f(x, y) = g_1(x, g_2(y, g_3(x)))$

Lemma 4.35

Eine erweiterte Komposition von PR Funktionen ist wieder PR.

Beweis:

Mit Induktion über Aufbau/Größe von t . □

Beispiel:

$$\begin{aligned} h_1(x, y) &= g_3(\pi_1^2(x, y)) \\ h_2(x, y) &= g_2(\pi_2^2(x, y), h_1(x, y)) \\ f(x, y) &= g_1(\pi_1^2(x, y), h_2(x, y)) \end{aligned}$$

260

Das erweiterte Schema der primitiven Rekursion erlaubt

$$\begin{aligned}f(0, \bar{x}) &= t_0 \\f(m + 1, \bar{x}) &= t\end{aligned}$$

wobei

- t_0 enthält nur PR Funktionen und die x_i ,
- t enthält nur $f(m, \bar{x})$, PR Funktionen, m und die x_i .

Lemma 4.36

Das erweiterte Schema der primitiven Rekursion führt nicht aus PR hinaus.

Beweis:

Mit Hilfe der erweiterten Komposition. □

Moral:

Primitive Rekursion erlaubt

$$f(m + 1, \bar{x}) = \dots f(m, \bar{x}) \dots$$

261

Beispiel 4.37

Die Vorgängerfunktion ist PR:

$$\begin{aligned}\textit{pred}(0) &= 0 \\ \textit{pred}(x + 1) &= x\end{aligned}$$

Die modifizierte Differenz ist PR:

$$\begin{aligned}x \dot{-} 0 &= x \\ x \dot{-} (y + 1) &= \textit{pred}(x \dot{-} y)\end{aligned}$$

262

Definition 4.38

Sei $P(x)$ ein Prädikat, d.h. ein logischer Ausdruck, der in Abhängigkeit von $x \in \mathbb{N}_0$ den Wert **true** oder **false** liefert. Dann können wir P in natürlicher Weise eine Funktion

$$\hat{P} : \mathbb{N} \rightarrow \{0, 1\}$$

zuordnen: $\hat{P}(x) = 1$ gdw $P(x) = \text{true}$.

Wir nennen P **primitiv rekursiv** genau dann, wenn \hat{P} primitiv rekursiv ist.

263

Ist P primitiv rekursiv, dann auch der **beschränkte max-Operator**

$$\max\{x \leq n \mid P(x)\} =: q(n)$$

wobei $\max \emptyset := 0$.

$$\begin{aligned} q(0) &= 0 \\ q(n+1) &= q(n) + \hat{P}(n+1) * ((n+1) \dot{-} q(n)) \\ &= \begin{cases} n+1 & \text{falls } P(n+1) \\ q(n) & \text{sonst} \end{cases} \end{aligned}$$

264

Ist P primitiv rekursiv, dann auch der beschränkte Existenzquantor

$$\exists x \leq n. P(x) =: Q(x)$$

denn:

$$\begin{aligned}\hat{Q}(0) &= \hat{P}(0) \\ \hat{Q}(n+1) &= \hat{Q}(n) + \hat{P}(n+1) * (1 - \hat{Q}(n)) \\ &= \begin{cases} 1 & \text{falls } P(n+1) \\ \hat{Q}(n) & \text{sonst} \end{cases}\end{aligned}$$

4.6 PR = LOOP

Hauptproblem bei LOOP → PR:

Kodierung aller Variablen eines LOOP Programms in einer Zahl.

Satz 4.39

Die Cantorsche Paarungsfunktion

$$c(x,y) := \binom{x+y+1}{2} + x = (x+y)(x+y+1)/2 + x$$

ist eine Bijektion zwischen \mathbb{N}^2 und \mathbb{N} .

	0	1	2	3	...
0	0	2	5	9	...
1	1	4	8	13	...
2	3	7	12	18	...
3	6	11	17	24	...
:	:	:	:	:	:
:	:	:	:	:	:

Die Funktion $x \mapsto \binom{x}{2}$ ist PR:

$$\binom{0}{2} = 0$$

$$\binom{n+1}{2} = \binom{n}{2} + n$$

Mit Komposition ist auch c PR:

$$c(x, y) = \binom{x+y+1}{2} + x$$

267

Mit c kodiert man $k+1$ Tupel:

$$\langle n_0, n_1, \dots, n_k \rangle := c(n_0, c(n_1, \dots, c(n_k, 0) \dots))$$

Wir brauchen die Umkehrfunktionen p_1 und p_2 von c :

$$p_1(c(x, y)) = x \quad p_2(c(x, y)) = y$$

Damit kann man Projektionsfunktionen auf Tupeln definieren:

$$d_0(n) := p_1(n)$$

$$d_1(n) := p_1(p_2(n))$$

⋮

$$d_k(n) := p_1(\underbrace{p_2 \dots p_2}_{k}(n) \dots)$$

Sind p_1, p_2 PR, so auch d_0, \dots, d_k .

268

Satz 4.40

Die Umkehrfunktionen von c sind PR definierbar:

$$\begin{aligned} p_1(n) &= \max\{x \leq n \mid \exists y \leq n. c(x, y) = n\} \\ p_2(n) &= \max\{y \leq n \mid \exists x \leq n. c(x, y) = n\} \end{aligned}$$

Beweis:

Alle Funktionen in der Definition der p_i sind PR, auch der Gleichheitstest (Übung!).

Die p_i sind die Umkehrfunktionen von c denn:

- Es gibt zu jedem n eindeutige x und y mit $c(x, y) = n$.
(Bijektivität von c)
- $x \leq c(x, y)$ und $y \leq c(x, y)$.

Damit findet die beschränkte Suche immer die gesuchten x und y .

□

269

Satz 4.41 (PR = LOOP)

Die primitiv rekursiven sind genau die LOOP-berechenbaren Funktionen.

Beweis:

LOOP \rightarrow PR:

Sei $f : \mathbb{N}^m \rightarrow \mathbb{N}$ LOOP-berechenbar.

Dann gibt es ein LOOP-Programm P (mit Variablen x_0, \dots, x_k), das f berechnet.

Mit Induktion über die Struktur von P konstruieren wir eine PR Funktion g_P mit

$$g_P(\underbrace{\langle a_0, \dots, a_k \rangle}_{\text{Belegung der Variablen beim Start von } P}) = \underbrace{\langle b_0, \dots, b_k \rangle}_{\text{Belegung der Variablen am Ende von } P}$$

270

Beweis (Forts.):

- P ist $x_i := x_j \pm c$:

$$g_P(x) = \langle d_0(x), \dots, d_{i-1}(x), d_j(x) \pm c, d_{i+1}(x), \dots, d_k(x) \rangle$$

- P ist $Q; R$: $g_P(x) = g_R(g_Q(x))$
- P ist LOOP x_i DO Q END:

$$\begin{aligned} h(0, x) &= x \\ h(n+1, x) &= g_Q(h(n, x)) \\ g_P(x) &= h(d_i(x), x) \end{aligned}$$

$g_P(x)$ berechnet den Zustand nach $d_i(x)$ Iterationen von Q .

Berechnet P die Funktion $f : \mathbb{N}^m \rightarrow \mathbb{N}$, dann ist f PR denn

$$f(x_1, \dots, x_m) = d_0(g_P(\langle 0, x_1, \dots, x_m, \underbrace{0, \dots, 0}_{k-m} \rangle)).$$

271

Beweis (Forts.): PR \rightarrow LOOP

Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.

Mit Induktion über die Erzeugung von f konstruieren wir ein LOOP-Programm P_f , das f berechnet:

- 0:

$$x_0 := 0$$

- Nachfolger-Funktion:

$$x_0 := x_1 + 1$$

- Projektions-Funktionen, zB $\pi_2^3(x_1, x_2, x_3) = x_2$:

$$x_0 := x_2$$

- Komposition, zB $f(x_1, x_2) = g(h_1(x_1, x_2), h_2(x_1, x_2))$:

$$P_{h_1}; x_{15} := x_0; P_{h_2}; x_1 := x_{15}; x_2 := x_0; P_g$$

Funktions-Komposition wird simuliert durch
Hintereinanderausführung (;) wobei Argumente und
Zwischenergebnisse in separaten Variablen
zwischengespeichert werden müssen.

272

Beweis (Forts.):

- Primitive Rekursion:

$$\begin{aligned} f(0, \bar{x}) &= g(\bar{x}) \\ f(y + 1, \bar{x}) &= h(f(y, \bar{x}), y, \bar{x}) \end{aligned}$$

Folgendes LOOP-Programm berechnet $f(y, \bar{x})$:

```
r := g( $\bar{x}$ );
k := 0;
LOOP y DO
    r := h(r, k,  $\bar{x}$ )
    k := k + 1;
END
x0 := r
```

wobei wir nach Induktionsannahme LOOP-Programme zur Berechnung von g und h konstruieren können.
(Schönings Programm falsch!)



273

Reversible Kodierung endlicher Zahlenfolgen als Zahlen:

$\{0, \dots, k\}^*$ Kodiere (i_1, \dots, i_n) als Zahl $i_1 \dots i_n$ zur Basis k .

\mathbb{N}^n Mit iterierter Paarfunktion $c: \langle i_1, \dots, i_n \rangle$

NB n muss beim Dekodieren bekannt sein denn zB
 $1 = c(0, 1) = c(0, c(0, 1))$.

\mathbb{N}^* Auch mit $\langle \dots \rangle$ reversibel kodierbar (Übung)

\mathbb{N}^* Kodiere (i_1, \dots, i_n) als $2^{i_1} 3^{i_2} \cdots p_n^{i_n}$
wobei p_n die n -te Primzahl ist.
Dekodierung = Primzahlzerlegung

274

4.7 Die μ -rekursiven Funktionen

- Mit PR kann man nur beschränkt suchen: $n, \dots, 0$.
- Die *unbeschränkte* Suche $0, \dots$ erfordert so etwas wie

$$f(n) = \dots f(n+1) \dots$$

- Der μ -Operator formalisiert diese Art der Suche.
- Damit erhält man alle berechenbaren Funktionen.
- Andere Such- bzw Rekursionsschemata sind (im Prinzip!) nicht notwendig.

Notation: $f(n) = \perp$ bedeutet „ $f(n)$ ist undefiniert.“

275

Definition 4.42 (μ -Operator)

Sei $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion.
Die durch Anwendung des μ -Operators entstehende Funktion
 $\mu f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist definiert durch:

$$\bar{x} \mapsto \begin{cases} \min\{n \in \mathbb{N} \mid f(n, \bar{x}) = 0\} & \text{falls} \\ & \text{ein solches } n \text{ existiert und} \\ & f(m, \bar{x}) \neq \perp \text{ für alle } m \leq n \\ \perp & \text{sonst} \end{cases}$$

Intuitiv: $\mu f(\bar{x}) = \text{find}(0, \bar{x})$
 $\text{find}(n, \bar{x}) = \text{if } f(n, \bar{x}) = 0 \text{ then } n \text{ else } \text{find}(n+1, \bar{x})$

Beispiel 4.43

Ist $f(n, x) = x \dot{(n+n)}$
dann ist $\mu f(x) =$

276

Definition 4.44 (μR)

Die Menge der μ -rekursiven Funktionen ist die kleinste Teilmenge aller (nicht notwendigerweise totalen) Funktionen, die die Basisfunktionen $(0, +1, \pi_i^k)$ enthält und alle Funktionen, die man hieraus durch (evtl. wiederholte) Anwendung von Komposition, primitiver Rekursion oder des μ -Operators erzeugen kann.

Satz 4.45 ($\mu R = \text{WHILE}$)

Die μ -rekursiven sind genau die WHILE-berechenbaren Funktionen.

Beweis: als Ergänzung des Beweises von $\text{PR} = \text{LOOP}$.

$\mu R \rightarrow \text{WHILE}$:

Fall μf : Nach IA gibt es ein WHILE-Programm für f .

Dann ist μf auch WHILE-berechenbar:

```
x0 := 0; y := f(0, x);  
WHILE y ≠ 0 DO x0 := x0 + 1; y := f(x0, x) END
```

277

Beweis (Forts.):

$\text{WHILE} \rightarrow \mu R$:

Fall P ist WHILE $x_i \neq 0$ DO Q END

Nach IA gibt es eine Funktion g_Q die Q berechnet.

Dann ist g_P wie folgt definierbar:

$$\begin{aligned} h(0, x) &= x \\ h(n+1, x) &= g_Q(h(n, x)) \\ h_i(n, x) &= d_i(h(n, x)) \end{aligned}$$

$h_i(n, x)$ ist der Wert von x_i nach n Iterationen von Q .

Dh $\mu h_i(x)$ ist die Anzahl der Iterationen von Q bis $x_i = 0$, dh bis P terminiert.

$$g_P(x) = h(\mu h_i(x), x)$$

□

278

Durch die Transformation

$$\mu R \rightarrow \text{WHILE mit einer Schleife (Kleene)} \rightarrow \mu R$$

genügt also immer ein μ -Operator:

Korollar 4.46 (Kleene)

Für jede n -stellige μ -rekursive Funktionen f gibt es zwei $n + 1$ -stellige PR Funktionen h und h' , so dass

$$f(\bar{x}) = h(\mu h'(\bar{x}), \bar{x})$$

279

4.8 Die Ackermann-Funktion

$$\begin{aligned} a(0, n) &= n + 1 \\ a(m + 1, 0) &= a(m, 1) \\ a(m + 1, n + 1) &= a(m, a(m + 1, n)) \end{aligned}$$

- Dies ist keine PR Definition.
- Aber: $\not\Rightarrow a$ ist nicht PR
- Ziel: a ist berechenbar, total, aber nicht PR

Fakt 4.47

Die Ackermann-Funktion ist (OCaml-)berechenbar.

280

Beispiel:

$$\begin{aligned}
 a(2, 2) &= \\
 a(1, a(2, 1)) &= \\
 a(1, a(1, a(2, 0))) &= \\
 a(1, a(1, a(1, 1))) &= \\
 a(1, a(1, a(0, a(1, 0)))) &= \\
 a(1, a(1, a(0, a(0, 1)))) &= \\
 a(1, a(1, a(0, 2))) &= \\
 a(1, a(1, 3)) &= \\
 a(1, a(0, a(1, 2))) &= \\
 a(1, a(0, a(0, a(1, 1)))) &= \\
 a(1, a(0, a(0, a(0, a(1, 0))))) &= \\
 a(1, a(0, a(0, a(0, a(0, 1))))) &= \\
 a(1, a(0, a(0, a(0, 2)))) &= \\
 a(1, a(0, a(0, 3))) &= \\
 a(1, a(0, 4)) &= \\
 a(1, 5) &= \\
 a(0, a(1, 4)) &= \\
 a(0, a(0, a(1, 3))) &= \\
 a(0, a(0, a(0, a(1, 2)))) &= \\
 a(0, a(0, a(0, a(0, a(1, 1))))) &= \\
 a(0, a(0, a(0, a(0, a(0, a(1, 0)))))) &= \\
 a(0, a(0, a(0, a(0, a(0, a(0, 1)))))) &= \\
 a(0, a(0, a(0, a(0, a(0, a(0, 2)))))) &= \\
 a(0, a(0, a(0, a(0, a(0, 3))))) &= \\
 a(0, a(0, a(0, 4))) &= \\
 a(0, a(0, 5)) &= \\
 a(0, 6) &=
 \end{aligned}$$

7

Scherzfrage: $a(6, 6) = ?$

281

Der Beginn:

$$\begin{aligned}
 a(0, n) &= n + 1 \\
 a(1, n) &= 2 + (n + 3) - 3 \\
 a(2, n) &= 2(n + 3) - 3 \\
 a(3, n) &= 2^{n+3} - 3 \\
 a(4, n) &= \underbrace{2^{\overbrace{\dots}^2}}_{n+3} - 3 \\
 a(5, n) &= ???
 \end{aligned}$$

Mit Induktion über n :

$$a(m + 1, n) = \underbrace{a(m, a(m, \dots a(m, 1) \dots)}_{n+1}$$

Bsp: $a(1, n) = a(0, \dots a(0, 1) \dots) = (+1)^{n+1}(1) = n + 2$

282

Ackermann als Familie von Funktionen A_m : $A_m(n) := a(m, n)$.
Nun gilt:

$$\begin{aligned} A_0(n) &= s(n) \\ A_{m+1}(n) &= A_m^{n+1}(1) = \underbrace{A_m(\dots A_m(1)\dots)}_{n+1} \end{aligned}$$

Beobachtung: alle A_m sind total und PR (mit Ind. über m)

Mit Currying (z.B. in Haskell):

$$\begin{aligned} a\ 0 &\quad n = n+1 \\ a\ (m+1)\ n &= \text{iter}\ (n+1)\ (a\ m)\ 1 \\ \text{iter}\ 0 &\quad f\ x = \\ \text{iter}\ (n+1)\ f\ x &= \end{aligned}$$

Ackermann ist mit primitiver Rekursion höherer Stufe definierbar.

283

Man kann auch direkt zeigen:

Lemma 4.48

Die Ackermann-Funktion ist total.

Beweis:

Mit Induktion über m : $\forall n \in \mathbb{N}. a(m, n) \neq \perp$ (1)

- Basis: $a(0, n) = n + 1 \neq \perp$
- Schritt: $a(m + 1, n) \neq \perp$ (2)

Beweis mit Induktion über n :

- $a(m + 1, 0) = a(m, 1) \neq \perp$ wg (1)
- $a(m + 1, n + 1) = a(m, a(m + 1, n)) = a(m, k) \neq \perp$
wg (2) und (1)

□

284

Lemma 4.49

Für jede PR Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ gibt es ein $t \in \mathbb{N}$, so dass

$$\forall \bar{x} \in \mathbb{N}^k. f(\bar{x}) < a(t, \max \bar{x}).$$

Beweis:

Mit Induktion über den Aufbau der Definition von f . □

Prinzip: $t \approx$ Länge der Definition von f .

Details: [Felscher. Berechenbarkeit]

Satz 4.50

Die Ackermann-Funktion ist nicht PR.

Beweis:

Indirekt. Angenommen a wäre doch PR. Dann ist auch f PR:

$$f(n) := a(n, n)$$

Nach obigem Lemma gibt es t mit $f(n) < a(t, n)$ für alle n .

$$f(t) < a(t, t) = f(t) \quad \textcolor{red}{\checkmark}$$



Oberflächlich intuitiv:

Die Ackermann-Funktion wächst schneller als alle PR Funktionen.

Genauer:

Die Funktion $n \mapsto a(n, n)$ wächst schneller als alle PR Funktionen.

Intuitiver Grund:

- Für fixes t ist $n \mapsto a(t, n)$ PR, denn dies ist A_t .
- A_t braucht aber eine PR Definition der Länge $O(t)$.
- Um $n \mapsto a(n, n)$ PR zu berechnen, müsste die Länge der PR Definition dynamisch mit der Eingabe wachsen.

Da die Ackermann-Funktion total, berechenbar und nicht PR ist:

Korollar 4.51

*Die PR Funktionen sind eine **echte** Teilkasse der berechenbaren totalen Funktionen.*

287

Die berechenbaren Funktionen

berechenbar = TM = WHILE = GOTO = μR

total (zB Ackermann)

LOOP = PR

288

4.9 Entscheidbarkeit und das Halteproblem

Ziel: Es ist **unentscheidbar** ob ein Programm terminiert.

Definition 4.52

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt **entscheidbar** gdw ihre **charakteristische Funktion**

$$\chi_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ 0 & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

Eine Eigenschaft/Problem $P(x)$ heißt entscheidbar gdw $\{x \mid P(x)\}$ entscheidbar ist.

Fakt 4.53

*Die entscheidbaren Mengen sind abgeschlossen unter Komplement:
Ist A entscheidbar, dann auch \overline{A} .*

289

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

- Sei $\Gamma = \{a_0, \dots, a_k\}$ und $Q = \{q_0, \dots, q_n\}$.
- $\delta(q_i, a_j) = (q_{i'}, a_{j'}, d)$ wird kodiert als

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m)$$

wobei $bin : \mathbb{N} \rightarrow \{0, 1\}^*$ die Binärkodierung einer Zahl ist und $m = 0/1/2$ falls $d = L/R/N$.

- Kodierung von δ : Konkatenation der Kodierungen aller $\delta(., .) = (., ., .)$, in beliebiger Reihenfolge.
- Kodierung von $\{0, 1, \#\}^*$ in $\{0, 1\}^*$:

$$\begin{array}{rcl} 0 & \mapsto & 00 \\ 1 & \mapsto & 01 \\ \# & \mapsto & 11 \end{array}$$

290

Die Kodierung $\text{TM} \rightarrow \{0, 1\}^*$ ist nicht surjektiv,
dh nicht jedes Wort über $\{0, 1\}^*$ kodiert eine TM.
Sei \hat{M} eine beliebige feste TM.

Definition 4.54

Die zu einem Wort $w \in \{0, 1\}^*$ gehörige TM M_w ist

$$M_w := \begin{cases} M & \text{falls } w \text{ Kodierung von } M \text{ ist} \\ \hat{M} & \text{sonst} \end{cases}$$

Die Kodierung von syntaktischen Objekten (Programmen, Formeln, etc) als Zahlen nennt man **Gödelisierung** und die Zahlen **Gödelnummern**.

291

Definition 4.55

$M[w]$ ist Abk. für „Maschine M mit Eingabe w “
 $M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 4.56 (Spezielles Halteproblem)

Gegeben: Ein Wort $w \in \{0, 1\}^*$.

Problem: Hält M_w bei Eingabe w ?

Als Menge:

$$K := \{w \in \{0, 1\}^* \mid M_w[w]\downarrow\}$$

292

Satz 4.57

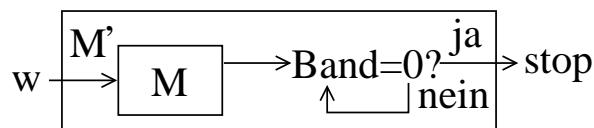
Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.
Dann ist auch f berechenbar:

$$f(w) := \begin{cases} 1 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

Berechnet die TM M χ_K so berechnet M' f :



Dann gibt es ein w' mit $M_{w'} = M'$.

$$\begin{aligned} f(w') = \perp &\Leftrightarrow \chi_K(w') = 1 \\ &\Leftrightarrow M_{w'}[w'] \downarrow \Leftrightarrow M'[w'] \downarrow \Leftrightarrow f(w') = 1 \quad \text{↯} \quad \square \end{aligned}$$

293

Definition 4.58 ((Allgemeines) Halteproblem)

Gegeben: Wörter $w, x \in \{0, 1\}^*$.

Problem: Hält M_w bei Eingabe x ?

Als Menge:

$$H := \{w \# x \mid M_w[x] \downarrow\}$$

Satz 4.59

Das Halteproblem H ist nicht entscheidbar.

Beweis:

Wäre H entscheidbar, dann trivialerweise auch K :

$$\chi_K(w) = \chi_H(w, w)$$

□

294

Definition 4.60 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ gibt mit

$$\forall w \in \Sigma^*. w \in A \Leftrightarrow f(w) \in B$$

Wir schreiben dann $A \leq B$.

Intuition:

- B ist mindestens so schwer zu lösen wie A .
- Ist A unlösbar, dann auch B .
- Ist B lösbar, dann erst recht A .

295

Lemma 4.61

Falls $A \leq B$ und B ist entscheidbar, so ist auch A entscheidbar.

Beweis:

Es gelte $A \leq B$ mittels f und χ_B sei berechenbar.

Dann ist $\chi_B \circ f$ berechenbar und $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \quad \square$$

Korollar 4.62

Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Beispiel 4.63

Da $K \leq H$ (mit Reduktion $f(w) := w\#w$) und K unentscheidbar ist, ist auch H unentscheidbar.

296

Satz 4.64

Das Halteproblem auf leerem Band, H_0 , ist unentscheidbar.

$$H_0 := \{w \in \{0, 1\}^* \mid M_w[\epsilon] \downarrow\}$$

Beweis:

Wir zeigen $K \leq H_0$ mit einer Funktion $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$.
 $f(w)$ ist die Gödelnummer folgender TM:

Überschreibe die Eingabe mit w ; führe M_w aus.

Dh f berechnet aus w die Kodierung w_1 einer TM, die w schreibt, und gibt die Kodierung von " $w_1; w$ " zurück.

Damit ist f total und berechenbar.

Es gilt:

$$w \in K \Leftrightarrow M_w[w] \downarrow \Leftrightarrow M_{f(w)}[\epsilon] \downarrow \Leftrightarrow f(w) \in H_0$$

□

297

Fazit:

Es gibt keine allgemeine algorithmische Methode, um zu entscheiden, ob ein Programm terminiert.

Die Unentscheidbarkeit vieler Fragen über die Ausführung von Programmen folgt durch Reduktion des Halteproblems:

- Kann ein WHILE-Programm mit einer bestimmten Eingabe einen bestimmten Programmpunkt erreichen?
Der Spezialfall Programmpunkt=Programmende ist das Halteproblem.
- Kann Variable x_7 bei einer bestimmten Eingabe je den Wert 2^{32} erreichen?

Reduktion: Ein Programm P hält gdw während der Ausführung von

$$P; \quad x_7 := 2^{32}$$

Variable x_7 den Wert 2^{32} erreicht.
(OE: x_7 kommt in P nicht vor)

298

Quiz

Ist es entscheidbar, ob eine TM

- ① mehr als 314 Zustände hat?
- ② bei Eingabe ϵ mehr als 314 Schritte macht?
- ③ bei *irgendeiner* Eingabe mehr als 314 Schritte macht?
- ④ bei *allen* Eingaben mehr als 314 Schritte macht?
- ⑤ bei Eingabe ϵ ihren Kopf mehr als 314 Felder von der 0-Position entfernen kann?
- ⑥ bei Eingabe ϵ einen bestimmten Zustand erreichen kann?
- ⑦ *irgendeine* Eingabe akzeptieren kann?

299

Es müssen nicht immer TM sein:

Satz 4.65 (Matiyasevich 1970, Hilberts 10. Problem)

Es ist unentscheidbar, ob ein Polynom in n Variablen mit ganzzahligen Koeffizienten eine ganzzahlige Nullstelle hat ($\in \mathbb{Z}^n$).

Beweis:

$$H \leq H_{10}$$



300

Bemerkungen

- Nicht alle unentscheidbaren Probleme sind gleich schwer
- ZB gilt: Das Äquivalenzproblem

$$Eq := \{u \# v \mid M_u \text{ berechnet die gleiche Funktion wie } M_v\}$$

ist schwerer als das Halteproblem:

$$H \leq Eq \quad \text{aber} \quad Eq \not\leq H$$

- Es gibt sogar eine unendliche Folge

$$A_0 \leq A_1 \leq A_2 \leq \dots$$

aber $A_{i+1} \not\leq A_i$.

301

4.10 Semi-Entscheidbarkeit

Definition 4.66

Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt **semi-entscheidbar (s-e)** gdw

$$\chi'_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ \perp & \text{falls } x \notin A \end{cases}$$

berechenbar ist.

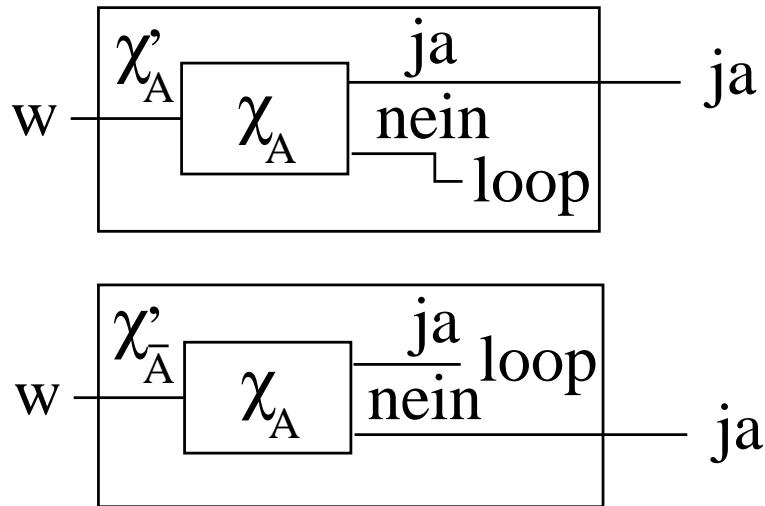
302

Satz 4.67

Eine Menge A ist entscheidbar gdw sowohl A als auch \overline{A} s-e sind.

Beweis:

„ \Rightarrow “: Wandle TM für χ_A in TM für χ'_A und $\chi'_{\overline{A}}$ um:



303

Beweis (Forts.):

„ \Leftarrow “:

Wandle TM M_1 für χ'_A und TM M_2 für $\chi'_{\overline{A}}$ in TM für χ_A um:

```

input( $x$ );
for  $s := 0, 1, 2, \dots$  do
  if  $M_1[x]$  hält in  $s$  Schritten then output(1); halt fi ;
  if  $M_2[x]$  hält in  $s$  Schritten then output(0); halt fi
```

Formulierung mit Parallelismus:

```

input( $x$ );
führen  $M_1[x]$  und  $M_2[x]$  parallel aus;
hält  $M_1$ , gib 1 aus, hält  $M_2$ , gib 0 aus.
```

□

Lemma 4.68

Ist $A \leq B$ und ist B s-e, so ist auch A s-e.

Beweis: Übung

304

Definition 4.69

Eine Menge A heißt **rekursiv aufzählbar** (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \rightarrow A$ gibt, so dass

$$A = \{f(0), f(1), f(2), \dots\}$$

Bemerkung:

- Es dürfen Elemente doppelt auftreten ($f(i) = f(j)$ für $i \neq j$)
- Die Reihenfolge ist beliebig.

Warnung: Rekursiv aufzählbar \neq abzählbar!

- Rekursiv aufzählbar \implies abzählbar
- Aber nicht umgekehrt:
 $\overline{K} \subseteq \{0, 1\}^*$ ist abzählbar aber nicht rekursiv aufzählbar (s.u.)

305

Lemma 4.70

Eine Menge A ist rekursiv aufzählbar gdw sie semi-entscheidbar ist.

Beweis:

Der Fall $A = \emptyset$ ist trivial. Sei $A \neq \emptyset$.

„ \Rightarrow “:

Sei A rekursive aufzählbar mit f . Dann ist A semi-entscheidbar:

```
input(x);
for i := 0, 1, 2, ... do
  if f(i) = x then output(1); halt fi
```

„ \Leftarrow “: Wir nehmen an $A \subseteq \mathbb{N}$.

Sei A semi-entscheidbar durch (zB) GOTO-Programm P .

Problem: $P[i]$ muss nicht halten und darf daher nur „zeitbeschränkt“ ausgeführt werden.

Gesucht: Paare (i, j) so dass $P[i]$ nach j Schritten hält.

Idee: Bijektion $\mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$, dh Umkehrung von c .

306

Beweis (Forts.):

Sei $d \in A$ beliebig.

Folgender Algorithmus berechnet eine Aufzählung von A :

```
input( $n$ );  
if  $P[p_1(n)]$  hält nach  $p_2(n)$  Schritten then output( $p_1(n)$ )  
else output( $d$ ) fi
```

Korrektheit:

Der Algorithmus hält immer und liefert immer ein Element aus A .

Vollständigkeit: Sei $a \in A \subseteq \mathbb{N}$.

Dann hält $P[a]$ nach einer endlichen Zahl k von Schritten.

Dann liefert die Eingabe $n =$ die Ausgabe a . □

Folgende Aussagen sind äquivalent:

- A ist semi-entscheidbar
- A ist rekursiv aufzählbar
- χ'_A ist berechenbar
- $A = L(M)$ für eine TM M
- A ist Definitionsbereich einer berechenbaren Funktion
- A ist Wertebereich einer berechenbaren Funktion

Satz 4.71

Die Menge $K = \{w \mid M_w[w] \downarrow\}$ ist semi-entscheidbar.

Beweis:

Die Funktion χ'_K ist wie folgt Turing-berechenbar:

Bei Eingabe w simuliere die Ausführung von $M_w[w]$;
gib 1 aus. □

- Hier haben wir benutzt, dass man einen Interpreter/Simulator für Turingmaschinen als Turingmaschine programmieren kann.
- Ein solcher Interpreter wird oft eine **Universelle Turingmaschine (U)** genannt.

Korollar 4.72

\overline{K} ist nicht semi-entscheidbar.

Semi-Entscheidbarkeit ist nicht abgeschlossen unter Komplement.

309

4.11 Die Sätze von Rice und Shapiro

Die von der TM M_w berechnete Funktion bezeichnen wir mit φ_w .
Wir betrachten implizit nur einstellige Funktionen.

Satz 4.73 (Rice)

Sei F eine Menge berechenbarer Funktionen.

Es gelte weder $F = \emptyset$ noch $F = \text{alle ber. Funkt.}$ („ F nicht trivial“)

Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

Nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.

Beispiel 4.74

Es ist unentscheidbar ob ein Programm

- irgendwo hält. ($F = \{\varphi_w \mid \exists x. M_w[x] \downarrow\}$)
- überall hält. ($F = \{\varphi_w \mid \forall x. M_w[x] \downarrow\}$)
- bei Eingabe 42 Ausgabe 42 produziert.

310

Warnung

Es ist entscheidbar ob ein Programm

- länger als 5 Zeilen ist.
- eine Zuweisung an die Variable x_{17} enthält.

Im Satz von Rice geht es um die von einem Programm berechnete Funktion (Semantik), nicht um den Programmtext (Syntax).

311

Beweis:

Wir zeigen $C_F := \{w \in \{0,1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \perp) \notin F$.

Wähle $h \in F \neq \emptyset$ beliebig; sei u Kodierung einer TM mit $\varphi_u = h$.

Reduziere K auf C_F ($K \leq C_F$) mit $f : \{0,1\}^* \rightarrow \{0,1\}^*$:

$f(w)$ ist die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band;
schreibe $w \# w$ auf die Eingabe und führe U aus;
führe M_u auf x aus.

Damit gilt $\varphi_{f(w)} = \begin{cases} h & \text{falls } M_w[w] \downarrow \\ \Omega & \text{sonst} \end{cases}$ und damit

$$w \in K \Leftrightarrow M_w[w] \downarrow \stackrel{(*)}{\Leftrightarrow} \varphi_{f(w)} \in F \Leftrightarrow f(w) \in C_F$$

$$(*) : \begin{cases} M_w[w] \downarrow \Rightarrow \varphi_{f(w)} = h \in F \\ \varphi_{f(w)} \in F \Rightarrow \varphi_{f(w)} = h \Rightarrow M_w[w] \downarrow \end{cases}$$

312

Beweis (Forts.):

Fall 2: $\Omega \in F$.

Wähle berechenbares $h \notin F$.

Zeige analog, dass $\overline{K} \leq C_F$. □

313

Satz 4.75 (Rice-Shapiro)

Sei F eine Menge berechenbarer Funktionen.

Ist $C_F := \{w \mid \varphi_w \in F\}$ semi-entscheidbar,

so gilt für alle berechenbaren f :

$f \in F \Leftrightarrow$ es gibt eine endlich Teilfunktion $g \subseteq f$ mit $g \in F$.

Beweis:

„ \Rightarrow “ mit Widerspruch.

Sei $f \in F$, so dass für alle endlichen $g \subseteq f$ gilt $g \notin F$.

Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. ↯

314

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t simuliere t Schritte von $M_w[w]$.

Hält diese Berechnung in $\leq t$ Schritten, gehe in eine endlos Schleife, sonst berechne $f(t)$.

Wir zeigen

$$w \in \overline{K} \Leftrightarrow h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w] \downarrow \implies \varphi_{h(w)} = f \in F \implies h(w) \in C_F$
- Falls $w \notin \overline{K}$ dann hält $M_w[w]$ nach t Schritten.
Damit gilt: $\varphi_{h(w)}$ ist f eingeschränkt auf $\{0, \dots, t - 1\}$.
Nach Annahme folgt $\varphi_{h(w)} \notin F$, dh $h(w) \notin C_F$.

315

Beweis (Forts.):

„ \Leftarrow “ mit Widerspruch.

Sei f berechenbar, sei $g \subseteq f$ endlich mit $g \in F$, aber sei $f \notin F$.

Wir zeigen $\overline{K} \leq C_F$ womit C_F nicht semi-entscheidbar ist. ↴

Reduktion $\overline{K} \leq C_F$ mit $h : \{0, 1\}^* \rightarrow \{0, 1\}^*$:

$h(w)$ ist die Kodierung folgender TM:

Bei Eingabe t , teste ob t im *endlichen* Def.ber. von g ist.

Wenn ja, berechne $f(t)$,

sonst simuliere $M_w[w]$ und berechne dann $f(t)$.

Wir zeigen

$$w \in \overline{K} \Leftrightarrow h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w] \downarrow \implies \varphi_{h(w)} = g \in F \implies h(w) \in C_F$
- $w \notin \overline{K} \implies M_w[w] \downarrow \implies \varphi_{h(w)} = f \notin F \implies h(w) \notin C_F$

□

316

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e \implies

$f \in F \Leftrightarrow$ es gibt endliche Funkt. $g \subseteq f$ mit $g \in F$.

Ein Programm heißt **terminierend** gdw es für alle Eingaben hält.

Korollar 4.76

- Die Menge der terminierenden Programme ist nicht semi-entscheidbar.
- Die Menge der nicht-terminierenden Programme ist nicht semi-entscheidbar.

Beweis:

- $F :=$ Menge aller berechenbaren totalen Funktionen.
Sei $f \in F$. Jede endliche $g \subseteq f$ ist echt partiell, dh $g \notin F$.
Also kann C_F nicht semi-entscheidbar sein.
- $F :=$ Menge aller berechenbaren nicht-totalen Funktionen.
Sei f total und berechenbar. Damit $f \notin F$.
Aber jede endliche $g \subseteq f$ ist in F .
Also kann C_F nicht semi-entscheidbar sein. \square

317

Grenzen automatischer Terminationsanalyse von Programmen

- Termination ist unentscheidbar (Rice):
Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
**Es gibt kein Zertifizierungs-Programm,
das alle terminierenden Programme erkennt.**
- Nicht-Termination ist nicht semi-entscheidbar (Rice-Shapiro):
**Es gibt keinen perfekten *Bug Finder*,
der alle nicht-terminierenden Programme erkennt.**

Aber es gibt mächtige heuristische Verfahren, die für
relativ viele Programme aus der Praxis (Gerätetreiber)

- Termination beweisen können, oder
- Gegenbeispiele finden können.

318

Ab hier viele Beweise wg Länge nicht auf Folien sondern Tafel.
Siehe Schöning.

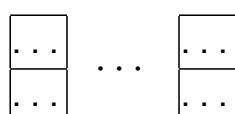
319

4.12 Das Postsche Korrespondenzproblem

Gegeben beliebig viele Kopien der 3 „Spielkarten“

001	10	0
00	11	010

gibt es dann eine Folge dieser Karten



so dass oben und unten das gleiche Wort steht?

320

Definition 4.77 (Postsche Korrespondenzproblem, Post's Correspondence Problem, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \dots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \dots, i_n \in \{1, \dots, k\}$, $n > 0$, mit $x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}$?

Dann nennen wir i_1, \dots, i_n eine Lösung des Problems $(x_1, y_1), \dots, (x_k, y_k)$.

Beispiel 4.78

- Hat $(1, 111), (10111, 10), (10, 0)$ eine Lösung?
- Hat $(b, ca), (a, ab), (ca, a), (abc, c)$ eine Lösung?
- Hat $(101, 01), (101, 010), (010, 10)$ eine Lösung?
- Hat $(10, 101), (011, 11), (101, 011)$ eine Lösung?
- Hat $(1000, 10), (1, 0011), (0, 111), (11, 0)$ eine Lösung?

321



Emil Post.

A Variant of a Recursively Unsolvable Problem.

Bulletin American Mathematical Society, 1946.

Emil Leon Post, 1897 (Polen) – 1954 (NY).

In einem Brief an Kurt Gödel, 1938:

*For fifteen years I carried around
the thought of astounding the
mathematical world with my
unorthodox ideas.*

...

*As for any claims I might make
perhaps the best I can say is
that I would have proved Gödel's
Theorem in 1921 — had I been
Gödel.*



322

Lemma 4.79

Das PCP ist semi-entscheidbar.

Beweis:

Zähle die möglichen Lösungen der Länge nach auf, und probiere jeweils, ob es eine wirkliche Lösung ist. \square

Wir zeigen nun:

$$H \leq MPCP \leq PCP$$

wobei

Definition 4.80 (Modifiziertes PCP, MPCP)

Gegeben: wie beim PCP

Problem: Gibt es eine Lösung i_1, \dots, i_n mit $i_1 = 1$?

Satz 4.81

$$MPCP \leq PCP$$

323

Beweis:

Für $w = a_1 \dots a_n$:

$$\begin{aligned}\overline{w} &:= \#a_1\#a_2\#\dots\#a_n\# \\ \overleftarrow{w} &:= a_1\#a_2\#\dots\#a_n\# \\ \overrightarrow{w} &:= \#a_1\#a_2\#\dots\#a_n\#\end{aligned}$$

$$f((x_1, y_1), \dots, (x_k, y_k)) := ((\overline{x_1}, \overrightarrow{y_1}), (\overleftarrow{x_1}, \overrightarrow{y_1}), \dots, (\overleftarrow{x_k}, \overrightarrow{y_k}), (\$, \$\$))$$

324

Satz 4.82

$$H \leq MPCP$$

325

Beweis:

- $(\#, \#q_0 u \#)$
- (a, a) für alle $a \in \Gamma \cup \{\#\}$
- $(qa, q'a')$ falls $\delta(q, a) = (q', a', N)$
 $(qa, a'q')$ falls $\delta(q, a) = (q', a', R)$
 $(bqa, q'ba')$ falls $\delta(q, a) = (q', a', L)$, für alle $b \in \Gamma$
- $(\#, \square \#), (\#, \# \square)$
- $(aq, q), (qa, q)$ für alle $q \in F, a \in \Gamma$
- $(q \#\#, \#)$ für alle $q \in F$

326

Aus $H \leq PCP$ folgt direkt

Korollar 4.83

Das PCP ist **unentscheidbar**.

Korollar 4.84

Das PCP ist auch für $\Sigma = \{0, 1\}$ unentscheidbar

Beweis:

Wir nennen dies das 01-PCP und zeigen $PCP \leq 01\text{-PCP}$.

Sei $\Sigma = \{a_1, \dots, a_m\}$ das Alphabet des gegebenen PCPs.

Abbildung auf ein 01-PCP: $\widehat{a_j} := 01^j$
 $\widehat{a_{j_1} \dots a_{j_n}} := \widehat{\widehat{a_{j_1}} \dots \widehat{a_{j_n}}}$

Dann hat $(x_1, y_1), \dots, (x_k, y_k)$ eine Lösung gdw

$(\widehat{x_1}, \widehat{y_1}), \dots, (\widehat{x_k}, \widehat{y_k})$ eine Lösung hat.

„ \Rightarrow “ klar, „ \Leftarrow “ folgt da $\widehat{\cdot} : \Sigma^* \rightarrow \{0, 1\}^*$ injektive ist:

$$\begin{aligned}\widehat{x_{i_1} \dots x_{i_n}} &= \widehat{y_{i_1} \dots y_{i_n}} \quad \Rightarrow \\ \widehat{\widehat{x_{i_1} \dots x_{i_n}}} &= \widehat{y_{i_1} \dots y_{i_n}} \quad \Rightarrow \quad x_{i_1} \dots x_{i_n} = y_{i_1} \dots y_{i_n}\end{aligned}$$

□

327

Bemerkungen

- Das PCP ist entscheidbar falls $|\Sigma| = 1$
- Das PCP ist entscheidbar falls $k \leq 2$.
- Das PCP ist unentscheidbar falls $k \geq 7$.
- Für $k = 3, \dots, 6$ ist noch offen, ob das PCP unentscheidbar ist.

328

4.13 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
 $L(A) = \emptyset, L(A) = L(B), \dots$
- Für TMs ist fast nichts entscheidbar:
 $L(M) = \emptyset, L(M_1) = L(M_2), \dots$
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$),
 und manches **unentscheidbar**: $L(G_1) = L(G_2), \dots$

Satz 4.85

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

- ① Ist $L(G_1) \cap L(G_2) = \emptyset$?
- ② Ist $|L(G_1) \cap L(G_2)| = \infty$?
- ③ Ist $L(G_1) \cap L(G_2)$ kontextfrei?
- ④ Ist $L(G_1) \subseteq L(G_2)$?
- ⑤ Ist $L(G_1) = L(G_2)$?

329

Beweis:

$K = (x_1, y_1), \dots, (x_k, y_k)$ wird abgebildet auf G_1

$$\begin{array}{lcl} S & \rightarrow & A\$B \\ A & \rightarrow & a_1Ax_1 \mid \dots \mid a_kAx_k \\ A & \rightarrow & a_1x_1 \mid \dots \mid a_kx_k \\ B & \rightarrow & y_1^R Ba_1 \mid \dots \mid y_k^R Ba_k \\ B & \rightarrow & y_1^R a_1 \mid \dots \mid y_k^R a_k \end{array}$$

und G_2 :

$$\begin{array}{lcl} S & \rightarrow & a_1Sa_1 \mid \dots \mid a_kSa_k \mid T \\ T & \rightarrow & 0T0 \mid 1T1 \mid \$ \end{array}$$

330

Beweis (Forts.):

□

331

Satz 4.86

Für eine CFG G sind folgende Probleme unentscheidbar:

- ① Ist G mehrdeutig?
- ② Ist $L(G)$ regulär?
- ③ Ist $L(G)$ deterministisch (DCFL)?

Beweis:

1. G mehrdeutig: Sei $G_3 := „G_1 \cup G_2“$.

K lösbar $\Leftrightarrow L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_3)$ ist mehrdeutig

„ \Rightarrow “: Syntaxbäume von G_1 und G_2 disjunkt

„ \Leftarrow “: G_1 und G_2 sind DCFG und damit nicht mehrdeutig

Damit ist $K \mapsto G_3$ Reduktion für $PCP \leq$ Mehrdeutigkeit.

2/3. $L(G)$ regulär/deterministisch: Sei $G_4 := „G'_1 \cup G'_2“$.

K unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset \Rightarrow L(G_4) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

K lösbar $\Rightarrow L(G_1) \cap L(G_2)$ nicht CFL $\Rightarrow \overline{L(G_4)}$ nicht reg/det

$\Rightarrow L(G_4)$ nicht reg/det.

Damit ist $K \mapsto G_4$ Reduktion für $PCP \leq L(G)$ reg/det. □

332

Satz 4.87

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Beweis:

Im Beweis des vorherigen Satzes hatten wir eine Reduktion
 $K \mapsto G_4$ mit

$$K \text{ unlösbar} \Leftrightarrow L(G_4) = \Sigma^*.$$

Sei $\Sigma = \{a_1, \dots, a_n\}$. Dann reduziert $K \mapsto (G_4, (a_1| \dots |a_n)^*)$ das PCP auf das Problem $L(G) = L(\alpha)$, □

333

5. Komplexitätstheorie

- Was ist mit beschränkten Mitteln (Zeit&Platz) berechenbar?
- Wieviel Zeit&Platz braucht man, um ein bestimmtes Problem/Sprache zu entscheiden?
- **Komplexität eines Problems**, nicht eines Algorithmus.
- **Obere Schranken**: durch Angabe eines Algorithmus
Bsp: $w \in L(G)$ für CFGs ist in Zeit $O(|w|^3)$ lösbar, mit CYK.
- **Untere Schranken**: schwierig ...
Bsp: Palindrom-Test auf einer 1-Band TM braucht Zeit $\Theta(n^2)$
Nicht hier.
- Einfluss des Maschinenmodells:
Deterministisch oder Nichtdeterministisch

334

Polynomielle und exponentielle Komplexität

Taktfrequenz: 1GHz

Zeit	Problemgröße n					
	10	20	30	40	50	60
n	.01 ms	.02 ms	.03 ms	.04 ms	.05 ms	.06 ms
n^2	.1 ms	.4 ms	.9 ms	1.6 ms	2.5 ms	3.6 ms
n^5	100ms	0.003s	0.02s	0.1 s	0.3 s	0.7 s
2^n	1 ms	0.001s	1 s	18 m	13 T	36 J
3^n	59 ms	3 s	2 T	385 J	$2 \cdot 10^7$ J	10^{12} J

ms=Mikrosek., s=Sek., m=Minute, T=Tag, J=Jahr

Problemgröße lösbar in fester Zeit: Speedup

Komplexität	n	n^2	n^5	2^n	3^n
1 MHz Prozessor	N_1	N_2	N_3	N_4	N_5
1 GHz Prozessor	$1000 N_1$	$32 N_2$	$4 N_3$	$N_4 + 10$	$N_5 + 6$

335

Zwei zentrale Komplexitätsklassen:

P = die von einer DTM in polynomieller Zeit lösbar Probleme
 = die „leichten“ Probleme (*feasible problems*)

NP = die von einer NTM in polynomieller Zeit lösbar Probleme
 In exponentieller Zeit lösbar (s. Simulation NTM durch DTM)

Zentrale Frage:

ist P = NP ?

Ist wichtig

- nicht weil Intel den NPentium angekündigt hat
- sondern weil viele *praktisch relevante* Such- und Optimierungsprobleme in NP liegen
- und alle schnell lösbar wären wenn eines schnell lösbar wäre.

336

Überblick:

- ① Die Komplexitätsklassen P und NP
- ② NP-Vollständigkeit
- ③ SAT: Ein NP-vollständiges Problem
- ④ Weitere NP-vollständige Probleme

337

5.1 Die Komplexitätsklasse P

Berechnungsmodelle:

DTM = deterministische Mehrband-TM

NTM = nichtdeterministische Mehrband-TM

Definition 5.1

$\text{time}_M(w)$:= Anzahl der Schritte bis die DTM $M[w]$ hält
 $\in \mathbb{N} \cup \{\infty\}$

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Funktion.

Die Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$\text{TIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{DTM } M. A = L(M) \wedge \forall w \in \Sigma^*. \text{time}_M(w) \leq f(|w|)\}$

Merke:

- n ist implizit die Länge der Eingabe
- Die DTM entscheidet die Sprache A in $\leq f(n)$ Schritten

338

Wir betrachten nur Polynome mit Koeffizienten $a_k, \dots, a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \dots + a_1 n + a_0$$

Definition 5.2

$$\mathsf{P} := \bigcup_{\substack{p \text{ Polynom}}} \mathsf{TIME}(p(n))$$

Damit gilt auch

$$\mathsf{P} = \bigcup_{k \geq 0} \mathsf{TIME}(O(n^k))$$

wobei

$$\mathsf{TIME}(O(f)) := \bigcup_{g \in O(f)} \mathsf{TIME}(g)$$

339

Beispiel 5.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathsf{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G \text{ ist CFG} \wedge w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G \text{ ist Graph} \wedge w \text{ ist Pfad in } G\} \in \mathsf{P}$
- $\{bin(p) \mid p \text{ Primzahl}\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$.

Bemerkungen

- $O(n \log n) \subset O(n^2)$
- $n^{\log n}, 2^n \notin O(n^k)$ für alle k
- Beweis $A \notin \mathsf{P}$ meist schwierig

340

- Warum P und nicht (zB) $O(n^{17})$?
 - Um robust bzgl Maschinenmodell zu sein:
 - 1-Band DTM braucht $O(t^2)$ Schritte
 - um t Schritte einer k -Band DTM zu simulieren.
 - Fast alle bekannten „vernünftigen“ Maschinenmodelle lassen sich von einer DTM in polynomieller Zeit simulieren.
 - Offen: Quantencomputer
- Warum TM? Historisch.
 - Ebenfalls möglich: (zB) WHILE.
 - Aber zwei mögliche Kostenmodelle:
 - Uniform** $x_i := x_j + n$ kostet 1 Zeiteinheit.
 - Logarithmisch** $x_i := x_j + n$ kostet $\log x_j$ Zeiteinheiten.

341

5.2 Die Komplexitätsklasse NP

Intuitiv:

- NP die Klasse der Sprachen, die von einer NTM in polynomieller Zeit akzeptiert werden.
- Dh: Eine Sprache A liegt in NP gdw es ein Polynom $p(n)$ und eine NTM M gibt, so dass
 - ① $L(M) = A$ und
 - ② für alle $w \in A$ kann $M[w]$ in $\leq p(|w|)$ Schritten akzeptieren, dh einen Endzustand erreichen.

342

Definition 5.4

$$ntime_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} & \text{falls } w \in L(M) \\ \text{bis NTM } M[w] \text{ akzeptiert} \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \rightarrow \mathbb{N}$ eine totale Funktion.

$$\begin{aligned} \text{NTIME}(f(n)) &:= \{A \subseteq \Sigma^* \mid \exists \text{NTM } M. A = L(M) \wedge \forall w \in \Sigma^*. ntime_M(w) \leq f(|w|)\} \\ \text{NP} &:= \bigcup_{p \text{ Polynom}} \text{NTIME}(p(n)) \end{aligned}$$

Bemerkungen:

- $P \subseteq NP$
- Seit etwa 1970 ist offen ob $P = NP$.

343

Bemerkungen zur Definition von NP:

Akzeptierende NTM M

- braucht für $w \notin L(M)$ nicht zu halten.
- kann für $w \in L(M)$ auch beliebig lange Berechnungsgfolgen haben.

Äquivalente Definition NP' von NP:

Die NTM $M[w]$ muss nach maximal $p(|w|)$ Schritten halten.

$NP' \subseteq NP$: Klar.

$NP \subseteq NP'$: Falls $A \in NP$ mit Polynom p und NTM M , so kann man NTM M' konstruieren mit $L(M') = A$, so dass $M'[w]$ immer innerhalb von polynomieller Zeit hält.

- ① Eingabe w
- ② Schreibe $p(|w|)$ auf ein getrenntes Band („timer“)
- ③ Simuliere $M[w]$, aber dekrementiere timer nach jedem Schritt.
- ④ Wird timer=0, ohne dass M gehalten hat, halte in einem nicht-Endzustand („timeout“)

344

Beispiel 5.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.
- Ein Graph hat einen Euler-Kreis gdw er zusammenhängend ist, und jeder Knoten geraden Grad hat.
- Beide Eigenschaften sind in polynomieller Zeit von einer DTM überprüfbar.
- $\implies \{G \mid G \text{ hat Euler-Kreis}\} \in P$

345

Beispiel 5.6

- Ein Hamilton-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jeden Knoten genau einmal enthält.
- HAMILTON := $\{G \mid G \text{ hat Hamilton-Kreis}\} \in NP$ mit folgendem Algorithmus der Art „Rate und prüfe“:
 - ① Rate eine Permutation der Knoten des Graphen.
 - ② Prüfe, ob diese Permutation ein Hamilton-Kreis ist.

Das Raten ist in polynomieller Zeit von einer NTM machbar.

Das Prüfen ist in polynomieller Zeit von einer DTM machbar.

Vermutung: HAMILTON $\notin P$, da man keinen substantiell besseren Algorithmus kennt, als alle Permutationen auszuprobieren.

346

Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob eine Lösungsvorschlag eine Lösung ist.

Definition 5.7

Sei M eine DTM mit $L(M) \subseteq \{w\#c \mid w \in \Sigma^*, c \in \Delta^*\}$.

- Falls $w\#c \in L(M)$, so heißt c Zertifikat für w .
- M ist ein polynomiell beschränkter Verifikator für die Sprache $\{w \in \Sigma^* \mid \exists c \in \Delta^*. w\#c \in L(M)\}$
falls es ein Polynom p gibt, so dass $\text{time}_M(w\#c) \leq p(|w|)$.

NB:

In Zeit $p(n)$ kann M maximal die ersten $p(n)$ Zeichen von c lesen.
Daher genügt für w ein Zertifikat der Länge $\leq p(|w|)$.

Beispiel 5.8 (HAMILTON)

Zertifikat: Knotenpermutation. In polynomieller Zeit verifizierbar.

347

Beispiel 5.9 (RUCKSACK)

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $c \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = c$?

$$\text{RUCKSACK} := \{ \text{bin}(a_1) \# \dots \# \text{bin}(a_n) \# \text{bin}(c) \mid \exists R \subseteq \{1, \dots, n\}. \sum_{i \in R} a_i = c \}$$

$\text{RUCKSACK} \in \text{NP}$:

Zertifikat: Indexmenge R . In polynomieller Zeit verifizierbar.

Merke: Der Nachweis $A \in \text{NP}$ ist meistens einfach.

348

Satz 5.10

$A \in NP$ gdw es gibt polynomiell beschränkten Verifikator für A .

Beweis:

„ \Rightarrow “:

Sei $A \in NP$. Dh es gibt NTM N , die A in Zeit $p(n)$ akzeptiert. Ein Zertifikat für $w \in A$ ist die Folge der benutzten Transitionen $\delta(q, a) \ni (q', a', d)$ einer akzeptierenden Berechnungsfolge von $N[w]$ mit $\leq p(n)$ Schritten.

Ein polynomiell beschränkter Verifikator für $L(N)$:

- ① Eingabe $w\#c$
- ② Simuliere $N[w]$, gesteuert durch die Transitionen in c .
- ③ Überprüfe dabei, ob die Transition in c jeweils zu N und zur augenblicklichen Konfiguration von N passt.
- ④ Akzeptiere, falls c in einen Endzustand führt.

349

Beweis (Forts.):

„ \Leftarrow “:

Sei M ein polynomiell (durch p) beschränkter Verifikator für A .

Wir bauen eine polynomiell beschränkte NTM N mit $L(N) = A$:

- ① Eingabe w .
- ② Schreibe hinter w zuerst $\#$ und dann ein nichtdeterministisch gewähltes Wort $c \in \Delta^*$, $|c| = p(|w|)$:
for $i := 1, \dots, p(|w|)$ **do**
schreibe ein Zeichen aus Δ und gehe nach rechts
- ③ Führe M aus (mit Eingabe $w\#c$).

Nach Annahme gilt $time_M(w\#c) \leq p(|w|)$.

Damit braucht $N[w]$ $O(p(|w|))$ Schritte. □

350

Fazit:

- P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.
- NP sind die Sprachen, bei denen ein Zertifikat für $w \in L$ schnell verifiziert/überprüft werden kann.

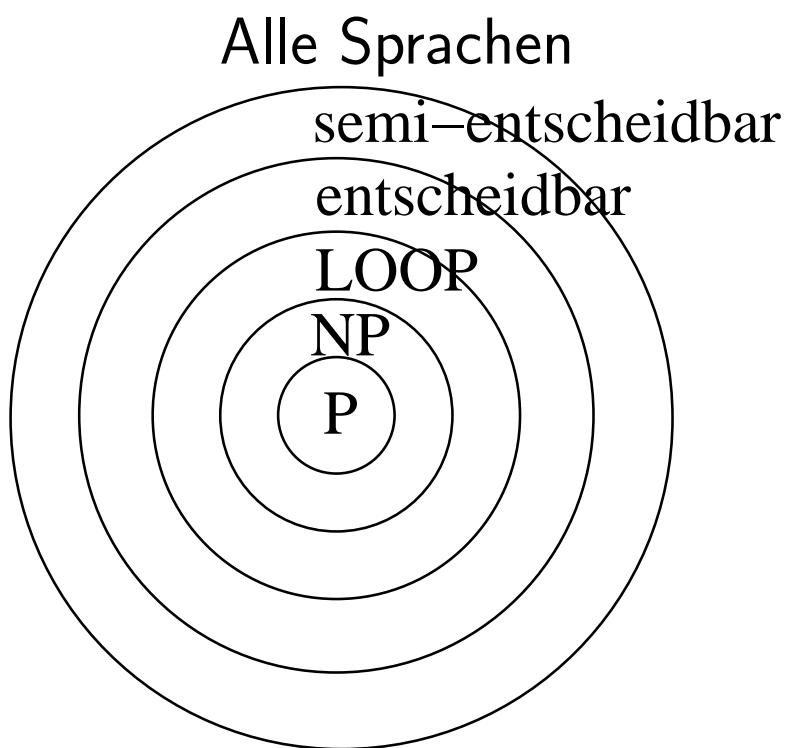
Intuition:

Es ist leichter, eine Lösung zu verifizieren als zu finden.

Aber:

Noch wurde von keiner Sprache bewiesen, dass sie in $\text{NP} \setminus \text{P}$ liegt.

351



Nur die P-NP-Grenze ist offen.

352

Satz 5.11

Ist $A \in TIME(f)$ und ist f LOOP-berechenbar, dann ist A LOOP-entscheidbar, dh χ_A ist LOOP-berechenbar.

Beweis:

Sei M eine DTM mit $L(M) = A$ und $time_M(w) \leq f(|w|)$.

$M \mapsto \text{GOTO} \mapsto \text{WHILE}$:

```
pc := 1;
WHILE pc ≠ 0 DO
    IF pc = 1 THEN P1 ELSE
        ...
        IF pc = k THEN Pk ELSE pc := 0
    END
```

wobei jeder Schleifendurchlauf einen Schritt von M simuliert.

Es kann maximal $f(|w|)$ viele Schleifendurchläufe geben.

Ersetze WHILE $pc \neq 0$ DO durch $i := f(|w|)$; LOOP i DO.

Berechnung wie vorher, aber evtl mit zusätzlichen Durchläufen mit $pc = 0$. □

353

Da + und * LOOP-berechenbar sind, folgt:

Korollar 5.12

Alle Sprachen in P sind LOOP-entscheidbar.

Da sich jede polynomiell zeitbeschränkte NTM (Zeit $O(p(n))$) in eine exponentiell zeitbeschränkte DTM (Zeit $O(c^{p(n)})$) umwandeln lässt und $c^{p(n)}$ LOOP-berechenbar ist, folgt:

Korollar 5.13

Alle Sprachen in NP sind LOOP-entscheidbar.

354

5.3 NP-Vollständigkeit

- ① Polynomielle Reduzierbarkeit \leq_p
- ② NP-vollständige Probleme = härteste Probleme in NP,
alle anderen Probleme in NP darauf polynomiell reduzierbar
- ③ Satz: SAT ist NP-vollständig

355

Definition 5.14

Sei $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$.

Dann heißt A polynomiell reduzierbar auf B , $A \leq_p B$,
gdw es eine totale und von einer DTM in polynomieller Zeit
berechenbare Funktion $f : \Sigma^* \rightarrow \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in A \iff f(w) \in B$$

Da $q(p(n))$ ein Polynom ist falls p und q Polynome sind:

Lemma 5.15

Die Relation \leq_p ist transitiv.

356

Lemma 5.16

Die Klassen P und NP sind unter polynomieller Reduzierbarkeit nach unten abgeschlossen:

$$A \leq_p B \in P/NP \implies A \in P/NP$$

Beweis:

- Sei $A \leq_p B$ mittels f , die von DTM M_f berechnet wird.
Polynom p begrenzt Rechenzeit von M_f .
- Sei $B \in P$ mittels DTM M .
Polynom q begrenzt Rechenzeit von M .

Damit ist $M_f; M$ polynomiell zeitbeschränkt in $|w|$:

- $M_f[w]$ macht $\leq p(|w|)$ Schritte.
- Ausgabe $f(w)$ von M_f hat Länge $\leq |w| + p(|w|)$.
- M macht $\leq q(|f(w)|) \leq q(|w|+p(|w|))$ Schritte (q monoton)

Daher macht $(M_f; M)[w]$ maximal $p(|w|) + q(|w| + p(|w|))$ Schritte, ein Polynom in $|w|$.

Analog: $A \leq_p B \in NP \implies A \in NP$

□

357

Ein Problem ist **NP-hart**,

wenn es mindestens so hart wie alles in NP ist:

Definition 5.17

Eine Sprache L heißt **NP-hart** gdw $A \leq_p L$ für alle $A \in NP$.

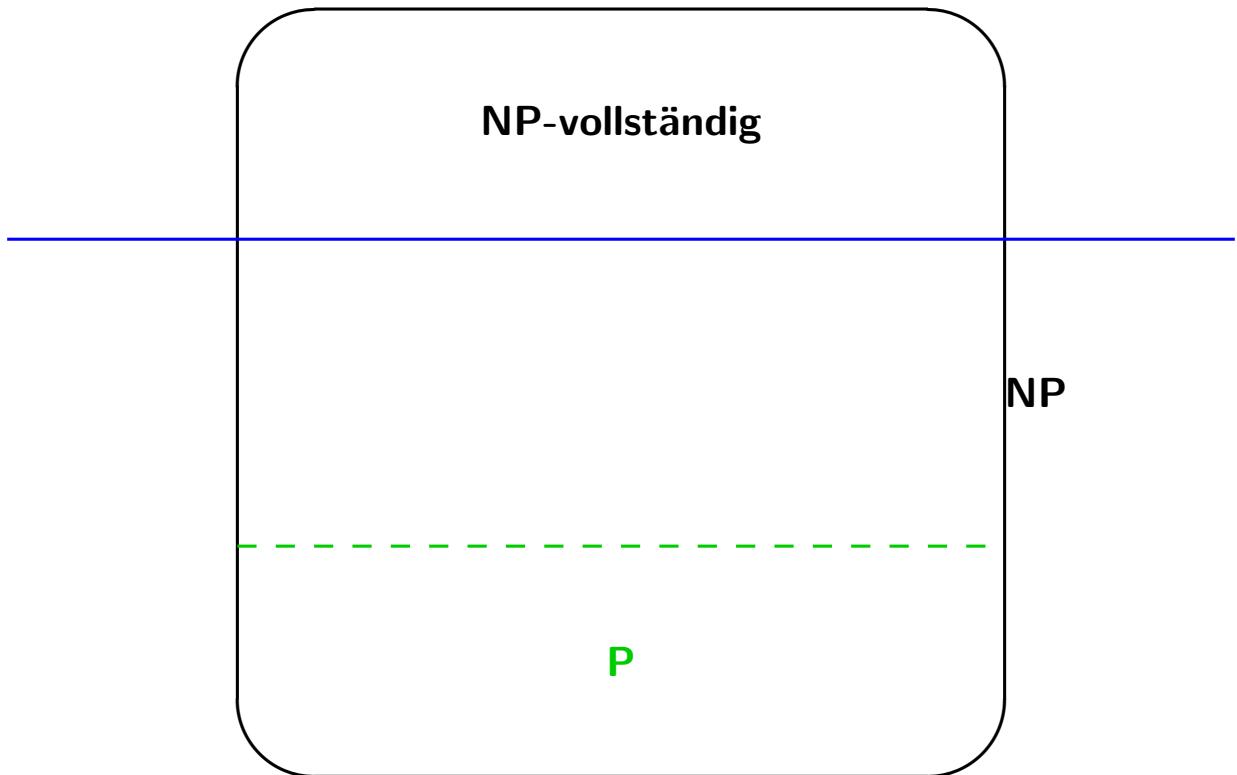
Definition 5.18

Eine Sprache L heißt **NP-vollständig** gdw
 L NP-hart ist und $L \in NP$.

NP-vollständige Probleme sind die schwierigsten Probleme in NP:
alle Probleme in NP sind polynomiell auf sie reduzierbar.

358

NP-hart



359

Wie man $P \stackrel{?}{=} NP$ lösen kann:

Lemma 5.19

Es gilt $P = NP$ gdw ein NP-vollständiges Problem in P liegt.

Beweis:

„ \Rightarrow “: Falls $P = NP$, so liegt jedes NP-vollständige Problem in P .

„ \Leftarrow “: Sei L ein NP-vollständiges Problem in P .

Dann gilt $P \supseteq NP$:

Ist $A \in NP$, so gilt $A \leq_p L$ (da L NP-hart)
und nach Lemma 5.16 $A \in P$ (da $L \in P$). □

Starke Vermutung:

- $P \neq NP$
- dh kein NP-vollständiges Problem ist in P .

Aber gibt es überhaupt NP-vollständige Probleme?

360

Aussagenlogik

Syntax der Aussagenlogik:

Formeln: $F \rightarrow \neg F \mid (F \wedge F) \mid (F \vee F) \mid X$

Variablen: $X \rightarrow x \mid y \mid z \mid \dots$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Konvention: man darf äußerste Klammern weglassen
und \wedge bindet stärker als \vee : $x \wedge y \vee z$ ist Abk. für $(x \wedge y) \vee z$

Semantik der Aussagenlogik:

- Eine **Belegung** ist eine Funktion von Variablen auf $\{0, 1\}$.
Bsp: $\sigma = \{x \mapsto 0, y \mapsto 1, z \mapsto 0, \dots\}$
- Belegungen werden mittels Wahrheitstabellen auf Formeln erweitert. Bsp: $\sigma((\neg x \wedge y) \vee (x \wedge \neg z)) = 1$
- Eine Formel F ist **erfüllbar** gdw es eine Belegung σ gibt mit $\sigma(F) = 1$

361

SAT

Gegeben: Eine aussagenlogische Formel F .

Problem: Ist F erfüllbar?

Praktische Bedeutung von SAT:

Äquivalenztest von Schaltungen (und vieles mehr)

Fakt 5.20

Zwei Formeln F_1 und F_2 sind genau dann äquivalent ($F_1 \leftrightarrow F_2$), wenn $(F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$ nicht erfüllbar ist.

Lemma 5.21

$SAT \in NP$

Beweis:

Belegungen sind Zertifikate, die in polynomieller Zeit geprüft werden können: Es gibt eine DTM, die bei Eingabe einer Formel F und einer Belegung σ für die Variablen in F , in polynomieller Zeit $\sigma(F)$ berechnet. □

362

Satz 5.22 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da $SAT \in NP$, bleibt noch zu zeigen, SAT ist NP-hart.

Sei $A \in NP$. Wir zeigen $A \leq_p SAT$.

Da $A \in NP$ gibt es NTM M mit $A = L(M)$ und

Polynom p mit $ntime_M(w) \leq p(|w|)$.

Reduktion bildet $w = x_1 \dots x_n \in \Sigma^*$ auf eine Formel F ab.

In polynomieller Zeit. So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Die Variablen beschreiben das *mögliche* Verhalten von $M[w]$:

$zust_{t,q}$	$t = 0, \dots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q
$pos_{t,i}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots, p(n)$	$pos_{t,i} = 1 \Leftrightarrow$ Kopfposition nach t Schritten ist i
$band_{t,i,a}$	$t = 0, \dots, p(n)$ $i = -p(n), \dots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \Leftrightarrow$ Bandinhalt nach t Schritten auf Bandposition i ist Zeichen a

363

Beweis (Forts.):

Sei $Q = \{q_1, \dots, q_k\}$ und $\Gamma = \{a_1, \dots, a_l\}$.

$$F := R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

$$R := \bigwedge_t [G(zust_{t,q_1}, \dots, Zust_{t,q_k}) \wedge G(pos_{t,-p(n)}, \dots, pos_{t,p(n)}) \wedge \bigwedge_i G(band_{t,i,a_1}, \dots, band_{t,i,a_l})]$$

$$A := Zust_{0,q_1} \wedge pos_{0,1} \wedge \bigwedge_{j=1}^n band_{0,j,x_j} \wedge \bigwedge_{j=-p(n)}^0 band_{0,j,\square} \wedge \bigwedge_{j=n+1}^{p(n)} band_{0,j,\square}$$

364

Beweis (Forts.):

$$\begin{aligned}
 T_1 &:= \bigwedge_{t,q,i,a} [zust_{t,q} \wedge pos_{t,i} \wedge band_{t,i,a} \\
 &\quad \rightarrow \bigvee_{(q',a',y) \in \delta(q,a)} (zust_{t+1,q'} \wedge pos_{t+1,i+y} \wedge band_{t+1,i,a'})] \\
 T_2 &:= \bigwedge_{t,i,a} ((\neg pos_{t,i} \wedge band_{t,i,a}) \rightarrow band_{t+1,i,a}) \\
 E &:= \bigvee_t \bigvee_{q \in F} zust_{t,q}
 \end{aligned}$$

$$G(v_1, \dots, v_r) := (\bigvee_{i=1}^r v_i) \wedge \left(\bigwedge_{i=1}^{r-1} \bigwedge_{j=i+1}^r \neg(v_i \wedge v_j) \right)$$

□

365

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT reduziert werden:

Sei F eine Formel mit den Variablen x_1, \dots, x_k :

```

if  $F \notin \text{SAT}$  then output("nicht lösbar")
else
  for  $i := 1$  to  $k$  do
    if  $F[x_i := 0] \in \text{SAT}$  then  $b := 0$  else  $b := 1$ 
    output( $x_i = b$ )
     $F := F[x_i := b]$ 
  
```

wobei $F[x := b] = F$ mit x ersetzt durch b .

Entscheidung von SAT in Zeit $O(f(n))$

\implies Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$
(falls es eine gibt.)

$f(n)$ polynomiell $\implies n \cdot (f(n) + n)$ polynomiell
 $f(n)$ exponentiell $\implies n \cdot (f(n) + n)$ exponentiell

366

Bemerkungen:

- Die Reduzierung der Lösungsberechnung auf SAT ist eine rein theoretische Konstruktion.
- Sie zeigt, dass man sich auf SAT beschränken kann, wenn man nur an polynomiell/exponentiell interessiert ist.
- Alle bekannten Entscheidungsverfahren für SAT berechnen auch eine Lösung.

367

Von NP-hart zu „NP-leicht“

- Bis vor 10 Jahren:

NP-vollständig = Todesurteil

- In den letzten 10 Jahren:

Spektakuläre Fortschritte bei *Implementierung* von
SAT-Lösern (*SAT-solver*): <http://satcompetition.org>
Stand der Kunst: Erfüllbar: bis 10^5 Variablen
Unerfüllbar: bis 10^3 Variablen

- Jetzt:

NP-vollständig = Hoffnung durch SAT

- Paradigma:

SAT (Logik!) als universelle Sprache
zur Kodierung kombinatorischer Probleme

Reduktion auf SAT manchmal schneller als problemspezifische Löser! Und fast immer einfacher.

368

Beispiel: Reduktion von 3-Färbbarkeit (3COL) auf SAT.

3COL

Gegeben: Ein ungerichteter Graph

Problem: Gibt es eine Färbung der Knoten, so dass keine benachbarten Knoten gleich gefärbt sind?

Lineare Reduktion von Graph $G = (V, E)$ auf SAT:

- Variablen = $V \times \{1, 2, 3\}$. Notation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Instanz von SAT:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Bemerkungen

- Zeigt $3COL \leq_p SAT$ und damit $3COL \in NP$.
- **Zeigt nicht, dass $3COL$ NP-vollständig ist.**

369

Ähnlich direkt polynomiell auf SAT reduzierbar:

- HAMILTON
- SUDOKU
- ...

370

Eine Anwendung von k -Färbbarkeit: Registerverteilung

Kann man in einem Programmstück n Variablen so auf k Register verteilen, dass jede Variable so lange in einem Register bleibt, wie sie lebendig ist?

Variable ist an einem Punkt **lebendig**

gdw sie später noch gelesen wird, ohne vorher überschrieben worden zu sein.

Reduktion auf k -Färbbarkeit:

Variable = Knoten

u und v verbunden = $u \neq v$ und es gibt einen Programmpunkt, an dem u und v lebendig sind

Farbe = Register

k -Färbung = Zuordnung eines Registers zu jeder Variablen

Sowohl k -Färbbarkeit als auch Registerverteilung ist für $k \geq 3$ NP-vollständig.

Mehr Information: Vorlesung *Programmoptimierung*

371

Weitere Anwendungen von SAT:

Bounded Model Checking

Hardware Entscheide, ob eine Schaltung mit Zustand für **alle** Eingaben innerhalb von 10 Zyklen ein bestimmtes Verhalten (nicht) hat.

Software Entscheide, ob ein Programm für **alle** Eingaben innerhalb von 10 Schritten ein bestimmtes Verhalten (nicht) hat.
Variablen müssen auf sehr kleine Wertebereiche eingeschränkt werden!

372

5.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

- Zeige $B \in \text{NP}$ (meist trivial)
- Zeige $A \leq_p B$ für ein NP-vollständiges Problem A .

Lemma 5.23

Ist A NP-vollständig, so ist $B \in \text{NP}$ ebenfalls NP-vollständig, falls $A \leq_p B$.

Beweis:

Folgt direkt aus der Transitivität von \leq_p :

B ist NP-hart, denn für $C \in \text{NP}$ gilt $C \leq_p A \leq_p B$. □

373

Warum will man wissen, dass ein Problem B NP-vollständig ist?

Um sicher zu sein, dass

- ein polynomieller Algorithmus für B ein Durchbruch wäre
- und daher wahrscheinlich nicht existiert.

Praktische Instanzen von B könnten trotzdem (zB mit SAT) „effizient“ lösbar sein.

374

Definition 5.24

- Eine Formel ist in **Konjunktiver Normalform (KNF)** gdw sie eine Konjunktion von **Klauseln** ist: $K_1 \wedge \dots \wedge K_n$
- Eine **Klausel** ist eine Disjunktion von **Literalen**: $L_1 \vee \dots \vee L_m$
- Ein **Literal** ist eine (evtl. negierte) Variable.
- Eine Formel ist in **3KNF** gdw jede Klausel ≤ 3 Literale enthält.

Dh eine KNF ist ein Konjunktion von Disjunktionen von (evtl negierten) Variablen.

Bsp: $(x_9 \vee \neg x_2) \wedge (\neg x_7 \vee x_1 \vee x_6)$

3KNF-SAT

Gegeben: Ein Formel in 3KNF

Problem: Ist die Formel erfüllbar?

Offensichtlich gilt 3KNF-SAT \in NP.

Aber vielleicht ist 3KNF-SAT einfacher als SAT?

375

Satz 5.25

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen $SAT \leq_p 3\text{KNF-SAT}$ mit einer polynomiellen Reduktion $F \mapsto F'$ so dass F' in 3KNF ist und

$$F \text{ ist erfüllbar} \Leftrightarrow F' \text{ ist erfüllbar}$$

NB F und F' sind **erfüllbarkeitsäquivalent**, aber nicht notwendigerweise auch äquivalent.

1. Transformiere F in **Negations-Normalform (NNF)** durch fortgesetzte Anwendung der de Morganschen Gesetze

$$\begin{aligned}\neg(A \wedge B) &= \neg A \vee \neg B \\ \neg(A \vee B) &= \neg A \wedge \neg B \\ \neg\neg A &= A\end{aligned}$$

von links nach rechts. F_1 ist Resultat.

376

Beweis (Forts.):

Für F_1 gilt: \neg nur noch direkt vor Variablen.

2. Betrachte F_1 als Baum, wobei die Literale Blätter sind.
Ordne jedem inneren Knoten eine neue Variable $\in \{y_0, y_1, \dots\}$ zu.
Ordne dabei der Wurzel von F_1 die Variable y_0 zu.
3. Betrachte die y_i als Abkürzung für die Teilbäume, an deren Wurzeln sie stehen

$$y_i = \begin{array}{c} \circ_i \\ / \quad \backslash \\ l_i \quad r_i \end{array}$$

wobei $\circ_i \in \{\wedge, \vee\}$ und l_i, r_i ein Literal oder eine Variable y_j ist.
Beschreibe F_1 Knoten für Knoten:

$$y_0 \wedge (y_0 \leftrightarrow (l_0 \circ_0 r_0)) \wedge (y_1 \leftrightarrow (l_1 \circ_1 r_1)) \dots =: F_2$$

377

Beweis (Forts.):

F_1 erf. $\implies F_2$ erf.: y_i bekommt Wert seines Teilbaums.

F_2 erf. $\implies F_1$ erf.: klar

4. Transformiere jede Äquivalenz in 3KNF:

$$\begin{aligned} (a \leftrightarrow (b \vee c)) &\mapsto (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c) \\ (a \leftrightarrow (b \wedge c)) &\mapsto (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c) \end{aligned}$$

Ergebnis ist F' .

Jeder Schritt ist in polynomieller Zeit in $|F|$ berechenbar.
Bei Transformation in NNF nimmt mit jedem Schritt

Summe der $|G|$ für alle Teilformeln $\neg G$ von F

ab. Daher erreicht man die NNF in $\leq |F|^2$ Schritten. □

378

Da jede Formel in 3KNF auch in KNF ist:

Korollar 5.26

KNF-SAT ist NP-vollständig.

Kann man wie folgt die NP-Vollständigkeit von KNF-SAT zeigen?

Man zeigt $SAT \leq_p KNF-SAT$
indem man jede Formel in KNF transformiert.

Satz 5.27

2KNF-SAT $\in P$

Ohne Beweis

379

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \dots, i_k \in \{1, \dots, n\}$ mit $M = T_{i_1} \cup \dots \cup T_{i_k}$?

Beispiel 5.28

$$\begin{array}{lll} T_1 & = & \{1, 2\} \\ T_3 & = & \{3, 4\} \\ M & = & \{1, 2, 3, 4, 5\} \end{array} \quad \begin{array}{lll} T_2 & = & \{1, 3\} \\ T_4 & = & \{3, 5\} \\ k & = & 3 \end{array}$$

Lösung:

Anwendung: Zulieferer

M Menge der Teile, die eine Firma einkaufen muss

T_i Menge der Teile, die Zulieferer i anbietet

Kann die Firma ihre Bedürfnisse mit k Zulieferern abdecken?

Fakt 5.29

$MÜ \in NP$.

380

Satz 5.30

MÜ ist NP-vollständig.

Beweis:

Wir zeigen KNF-SAT \leq_p MÜ.

Sei $F = K_1 \wedge \dots \wedge K_m$ in KNF, mit Variablen x_1, \dots, x_k .

$$M := \{1, \dots, m, m+1, \dots, m+k\}$$

$$T_i := \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\}$$

$$T'_i := \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\}$$

F ist erfüllbar gdw

M wird durch k der Teilmengen $T_1, \dots, T_k, T'_1, \dots, T'_k$ überdeckt

□

381

Das Minimierungsproblem

Gegeben: Teilmengen $T_1, \dots, T_n \subseteq M$ einer endlichen Menge M

Problem: Finde das kleinste k , so dass M von k der Teilmengen überdeckt wird.

kann auf das Entscheidungsproblem reduziert werden:

Finde kleinstes k durch binäre Suche im Intervall $[1, n]$ mit $O(\log n)$ Aufrufen von MÜ.

Kann man MÜ in Zeit $O(f(n))$ entscheiden, dann kann man das kleinste k in Zeit $O(f(n) \cdot \log n)$ berechnen.

$$\begin{aligned} f(n) \text{ polynomiell} &\implies f(n) \cdot \log n \text{ polynomiell} \\ f(n) \text{ exponentiell} &\implies f(n) \cdot \log n \text{ exponentiell} \end{aligned}$$

382

Die Berechnung einer **Lösung**

Gegeben: Teilmengen $\vec{T} := T_1, \dots, T_n \subseteq M$ einer endlichen Menge M , und eine Zahl $k \leq n$.

Problem: Finde eine Überdeckung von M durch k der Teilmengen.

kann auf das Entscheidungsproblem reduziert werden:

if $(\vec{T}, M, k) \notin \text{MÜ}$ **then** output(“nicht lösbar”)

else

$\ddot{U} := \emptyset$

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

if $(\vec{T} - T_i, M, k) \in \text{MÜ}$

then $\vec{T} := \vec{T} - T_i$

else $\ddot{U} := \ddot{U} \cup \{T_i\}$

383

CLIQUE

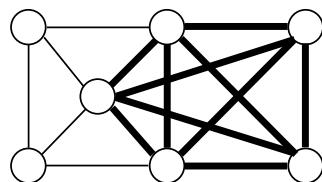
Gegeben: Ungerichteter Graph $G = (V, E)$ und Zahl $k \in \mathbb{N}$.

Problem: Besitzt G eine „Clique“ der Größe mindestens k ?

Dh eine Teilmenge $V' \subseteq V$ mit $|V'| \geq k$

und alle $u, v \in V'$ mit $u \neq v$ sind benachbart.

Beispiel mit 5-Clique:



Anwendung von Clique-Berechnung:

Knoten = Prozess

Kante = Zwei Prozesse sind parallel ausführbar

\Rightarrow Clique ist Gruppe von parallelisierbaren Prozessen

Fakt 5.31

$\text{CLIQUE} \in NP$

384

Satz 5.32

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT \leq_p CLIQUE:

Eine Formel F in 3KNF-SAT (oE: genau 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1), (1,2), (1,3), \dots, (k,1), (k,2), (k,3)\} \\ E &= \{\{(i,j), (p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

F ist erfüllbar durch eine Belegung $\sigma \iff$

Es gibt in jeder Klausel i ein Literal z_{ij_i} mit $\sigma(z_{ij_i}) = 1$

\iff

Die Literale $z_{1j_1}, \dots, z_{kj_k}$ sind paarweise nicht komplementär

\iff

Die Knoten $(1, j_1), \dots, (k, j_k)$ sind paarweise benachbart

$\iff G$ hat eine Clique der Größe k . □

385

RUCKSACK

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \dots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 5.33

RUCKSACK ist NP-vollständig.

Beweis:

3KNF-SAT \leq_p RUCKSACK :

Sei $F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$, wobei

$$z_{ij} \in \{x_1, \dots, x_n\} \cup \{\neg x_1, \neg x_2, \dots, \neg x_n\}$$

D.h. $m =$ Anzahl der Klauseln und n Anzahl der vorkommenden Variablen.

Wir geben jetzt Zahlen $\vec{a} = a_1, \dots, a_k$ und b an. b ist (etwa im Dezimalsystem)

$$b = \underbrace{44 \dots 444}_m \underbrace{11 \dots 11}_n$$

□

386

PARTITION

Gegeben: Zahlen $a_1, \dots, a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \dots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 5.34

PARTITION ist NP-vollständig.

Beweis: RUCKSACK \leq_p PARTITION

Beispiel: $\vec{a} = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen 7, 9, 7, 15, dh $R = \{2, 4, 5, 7\}$

RUCKSACK \rightarrow PARTITION:

$$M := \sum_{i=1}^n a_i = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$$

Das resultierende PARTITONS-Problem: 12, 7, 4, 9, 7, 3, 15, 20, 39

Lösung: {7, 9, 7, 15, 20} und {12, 4, 3, 39}, dh $I = \{2, 4, 5, 7, 8\}$.

Reduktion im Allgemeinen:

$$a_1, a_2, \dots, a_k, b \mapsto a_1, a_2, \dots, a_k, M - b + 1, b + 1$$

□

387

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ a_1, a_2, \dots, a_n .

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 5.35

BIN PACKING ist NP-vollständig.

Beweis: PARTITION \leq_p BIN PACKING

$$(a_1, \dots, a_n) \mapsto (b, k, 2a_1, \dots, 2a_n)$$

wobei $b := \sum_{i=1}^k a_i$ und $k := 2$.

□

388

HAMILTON

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

Satz 5.36

HAMILTON ist NP-vollständig.

389

TRAVELLING SALESMAN (TSP)

Gegeben: Eine $n \times n$ Matrix $M_{ij} \in \mathbb{N}$ von „Entfernungen“ und eine Zahl $k \in \mathbb{N}$

Problem: Gibt es eine „Rundreise“ (Hamilton-Kreis) der Länge $\leq k$?

Satz 5.37

TSP ist NP-vollständig.

Beweis: HAMILTON \leq_p TSP

$$(\{1, \dots, n\}, E) \mapsto (M, n)$$

wobei

$$M_{ij} := \begin{cases} 1 & \text{falls } \{i, j\} \in E \\ 2 & \text{sonst} \end{cases}$$

□

390

FÄRBBARKEIT (COL)

Gegeben: Ein ungerichteter Graph (V, E) und eine Zahl k .

Problem: Gibt es eine Färbung der Knoten V mit k Farben, so dass keine zwei benachbarten Knoten die gleiche Farbe haben?

Satz 5.38

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

Beweis:

3KNF-SAT \leq_p 3FÄRBBARKEIT □

Satz 5.39

2FÄRBBARKEIT $\in P$

391

Die NP-Bibel, der NP-Klassiker:



Michael Garey and David Johnson.

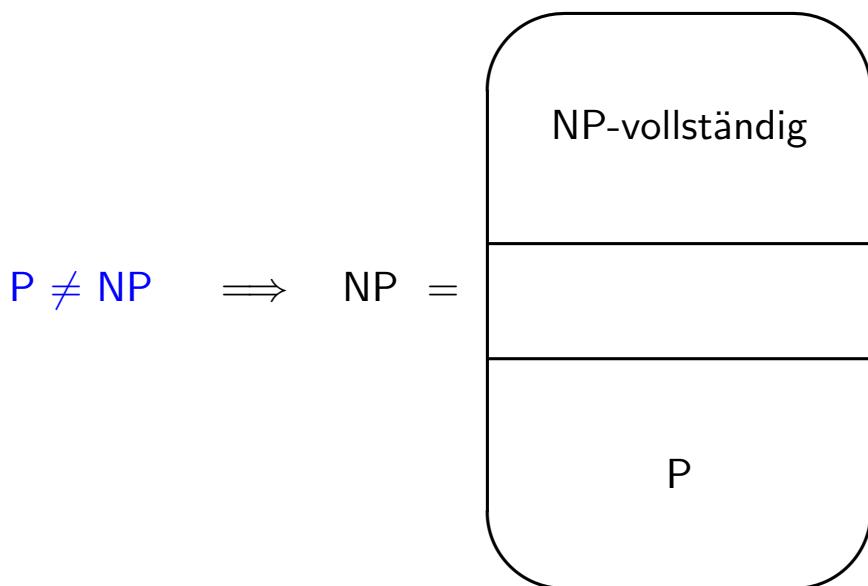
Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

Despite the 23 years that have passed since its publication, I consider Garey and Johnson the single most important book on my office bookshelf.

Lance Fortnow, 2002.

392

Man weiß nicht ob $P = NP$. Aber man weiß (Ladner 1975)



393



Kurt Gödel.

Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik, 1931.

Kurt Gödel
1906 (Brünn) –
1978 (Princeton)



394

5.5 Die Unvollständigkeit der Arithmetik

Syntax der Arithmetik:

Variablen: V $x \ y \ z \ \dots$

Zahlen: N $0 \ 1 \ 2 \ \dots$

Terme: T $V \ N \ (T + T) \ (T * T)$

Formeln: F $(T = T) \ \neg F \ (F \wedge F) \ (F \vee F)$
 $\exists V. F$

Wir betrachten $\forall x. F$ als Abk. für $\neg \exists x. \neg F$.

Definition 5.40

Ein Vorkommen einer Variablen x in einer Formel F ist **gebunden** gdw das Vorkommen in einer Teilformel der Form $\exists x. F'$ oder $\forall x. F'$ in der Teilformel F' liegt.

Ein Vorkommen ist **frei** gdw es nicht gebunden ist.

395

Notation: $F(x_1, \dots, x_k)$ bezeichnet eine Formel F , in der höchstens x_1, \dots, x_k frei vorkommen.

Sind $n_1, \dots, n_k \in \mathbb{N}$ so ist $F(n_1, \dots, n_k)$ das Ergebnis der Substitution von n_1, \dots, n_k für die freien Vorkommen von x_1, \dots, x_k .

Beispiel 5.41

$$\begin{aligned} F(x, y) &= (x = y \wedge \exists x. x = y) \\ F(5, 7) &= (5 = 7 \wedge \exists x. x = 7) \end{aligned}$$

Ein **Satz** ist eine Formel ohne freie Variablen.

Beispiel 5.42

$$\exists x. \exists y. x = y \qquad \exists y. \exists x. x = y$$

Mit **S** bezeichnen wir die Menge der arithmetischen Sätze.

396

Die Menge der **wahren** Sätze der Arithmetik nennen wir W :

Definition 5.43

$(t_1 = t_2) \ W$ gdw t_1 und t_2 den selben Wert haben.

$\neg F \ W$ gdw $F \not\in W$

$(F \wedge G) \ W$ gdw $F \in W$ und $G \in W$

$(F \vee G) \ W$ gdw $F \in W$ oder $G \in W$

$\exists x. F(x) \ W$ gdw es $n \in \mathbb{N}$ gibt, so dass $F(n) \in W$

Fakt 5.44

Für jeden Satz F gilt entweder $F \in W$ oder $\neg F \in W$

NB Ob eine Formel mit freien Variablen wahr ist, kann vom Wert der freien Variablen abhängen:

$$\exists x. x + x = y$$

Daher haben wir Wahrheit nur für Sätze definiert.

397

Definition 5.45

Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ ist **arithmetisch repräsentierbar** gdw es eine Formel $F(x_1, \dots, x_k, y)$ gibt, so dass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \dots, n_k) = m \quad \text{gdw} \quad F(n_1, \dots, n_k, m) \in W$$

Satz 5.46

Jede WHILE-berechenbare Funktion ist arithmetisch repräsentierbar.

398

Satz 5.47

W ist nicht entscheidbar.

Korollar 5.48

W ist nicht semi-entscheidbar.

399

Wir kodieren Beweise als Zahlen.

Definition 5.49

Ein **Beweissystem** für die Arithmetik ist ein entscheidbares Prädikat

$$Bew : \mathbb{N} \times S \rightarrow \{0, 1\}$$

wobei wir $Bew(b, F)$ lesen als „ b ist Beweis für Formel F “.

Ein Beweissystem Bew ist **korrekt** gdw

$$Bew(b, F) \implies F W.$$

Ein Beweissystem Bew ist **vollständig** gdw

$$F W \implies \text{es gibt } b \text{ mit } Bew(b, F).$$

400

Satz 5.50 (Gödel)

Es gibt kein korrektes und vollständiges Beweissystem für die Arithmetik.

Beweis:

Denn mit jedem korrekten und vollständigen Beweissystem kann man $\chi'_W(F)$ programmieren:

```
b := 0
while Bew(b, F) = 0 do b := b + 1
output(1)
```

□

401

5.6 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

Variablen: $V \rightarrow x | y | z | \dots$

Zahlen: $N \rightarrow 0 | 1 | 2 | \dots$

Terme: $T \rightarrow V | N | T + T$

Formeln: $F \rightarrow (T = T) | \neg F | (F \wedge F) | (F \vee F)$
| $\exists V. F$

Wir betrachten $\forall x. F$ als Abk. für $\neg \exists x. \neg F$.

Satz 5.51

Für Sätze der Presburger Arithmetik ist entscheidbar, ob sie wahr sind.

402



Mojzesz Presburger.

Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. 1929.

Mojzesz Presburger 1904 – 1943



The Holocaust Martyrs' and Heroes' Remembrance Authority
Hall of Names · P.O.B. 3477, Jerusalem 91034 www.yadvashem.org



יד ושם

רשות הזיכרון לשואה ולגבורה
היגייל בשמות - ח' ג 3422 - ירושלים

Page of Testimony זע פג

דף עד לרשום והנצחה של הנפטרים בשואה; אנא למלא דף עבור כל נספה בפרד, בכתב ברור ובאותיות דפוס.
Page of Testimony for commemoration of the Jews who perished during the Shoah; please submit a separate form for each victim, in block capitals.

	<p>תקנות זיכרון למשתנין והניצחון - חוק זיכרון 5713-1953 קומץ 5905-55: 2 צו יאנקוקון של דוד ווון ואוחזת אל החותם והוא כולם יכולים להזכיר את שמו.</p> <p>The Martyrs' and Heroes' Remembrance Law 5713-1953 determines in section 2 that: "The task of Yad Vashem is to gather into the homeland material regarding all those members of the Jewish people who laid down their lives, who fought and rebelled against the nazi enemy and his collaborators, and to perpetuate their names and those of the communities, organizations and institutions which were destroyed because they were Jewish".</p>										
	Maiden name:			שם משפחתו לפני הנישואין:			Victim's family name:			שם משפחתו של הנשפט:	
									PRESBURGER		
Previous/other family name:			שם משפחתו קודם/אחר:						שם פרטי (שם חביבה/כינוי):		
									MOGZESZ O MIETEK ?		
Approx. age at death:			Date of birth:			תאריך לידה:			Gender:	שם:	
43			1904						M / F	ת"ז:	
Nationality:	מולדת:	Country:	ארץ:	Region:	חוות:	חוות:	Place of birth:	תורה:			
POLISH		POLAND			?	?	?	מקום לידה:			
Victim's father:	Family name:				שם משפחתי:		First name:		שם פרטי:		
Victim's mother:	Maiden name:				שם כבוי הנישואין:		First name:		שם פרטי:		
Victim's wife/husband:	No. of children	סוכס:	Family status:	סטטוס חברתי:	Maiden name:	שם לפני הנישואין:	First name:	שם פרטי:		אשתה/בעל	
Address:											טלפון:

403

Presburger Arithmetik

1

normale Arithmetik ohne Multiplikation

Arithmetik

: hochgradig unentscheidbar :-(
sogar sogar unvollständig :((

1

Hilberts 10tes Problem Gödels Theorem

Vereinfachte Presburger Formeln:

$$\begin{array}{ccl} \phi & ::= & x + y = z \quad | \quad x = n \quad | \\ & & \phi_1 \wedge \phi_2 \quad | \quad \neg \phi \quad | \\ & & \exists x. \phi \end{array}$$

Bemerkungen

- Durch sukzessive Addition kann man Multiplikation mit Konstanten simulieren.
(Wie?)
- Das Rucksackproblem lässt sich in PA ausdrücken.
(Wie?)

405

Allgemeineres Ziel: PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee: Codiere die Werte der Variablen als Worte ...

213	t	<table border="1"><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	0	1	1	0	1	0	1	0	0	0	1	0	0	0
1	0	1	0	1	0	1	1																											
0	1	0	1	0	1	0	0																											
1	0	0	1	1	0	1	0																											
1	0	0	0	1	0	0	0																											
42	z	<table border="1"><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr></table>	0	1	0	1	0	1	0	0	1	0	0	1	1	0	1	0	1	0	0	0	1	0	0	1	0	1	0	0	0	1	1	0
0	1	0	1	0	1	0	0																											
1	0	0	1	1	0	1	0																											
1	0	0	0	1	0	0	1																											
0	1	0	0	0	1	1	0																											
89	y	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr></table>	1	0	0	1	1	0	1	0	0	1	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
1	0	0	1	1	0	1	0																											
0	1	0	0	1	0	0	1																											
1	0	0	0	0	1	0	0																											
0	0	1	0	0	1	0	0																											
17	x	<table border="1"><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table>	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	1
1	0	0	0	1	0	0	0																											
0	1	0	0	0	1	0	0																											
0	0	1	0	0	0	1	0																											
0	0	0	1	0	0	0	1																											

406

Allgemeineres Ziel:**PSAT**

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

407

Allgemeineres Ziel:**PSAT**

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

408

Allgemeineres Ziel:**PSAT**

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

409

Allgemeineres Ziel:**PSAT**

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

410

Allgemeineres Ziel:**PSAT**

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1 0 1 0 1 0 1 1
42	z	0 1 0 1 0 1 0 0
89	y	1 0 0 1 1 0 1 0
17	x	1 0 0 0 1 0 0 0

411

Allgemeineres Ziel:**PSAT**

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1 0 1 0 1 0 1 1
42	z	0 1 0 1 0 1 0 0
89	y	1 0 0 1 1 0 1 0
17	x	1 0 0 0 1 0 0 0

412

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

413

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

414

Allgemeineres Ziel:

PSAT

Finde Werte in \mathbb{N} für die freien Variablen , so dass ϕ gilt ...

Bemerkung:

Ganzzahlige Optimierung lässt sich als PSAT-Problem ausdrücken.
(Wie?)

Idee:

Codiere die Werte der Variablen als Worte ...

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

415

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

$$\begin{array}{lcl} \phi_1 \wedge \phi_2 & \implies & \overline{\mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2)} \quad (\text{Durchschnitt}) \\ \neg\phi & \implies & \overline{\mathcal{L}(\phi)} \quad (\text{Komplement}) \\ \exists x : \phi & \implies & \pi_x(\mathcal{L}(\phi)) \quad (\text{Projektion}) \end{array}$$

Achtung:

- Ein akzeptiertes Tupel kann immer durch führende Nullen verlängert werden !
- bleibt unter Vereinigung, Durchschnitt und Komplement erhalten !!
- Wie sieht das mit der Projektion aus ?

416

Weg-Projizierung der x -Komponente:

213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0
17	x	1	0	0	0	1	0	0	0

417

Weg-Projizierung der x -Komponente:

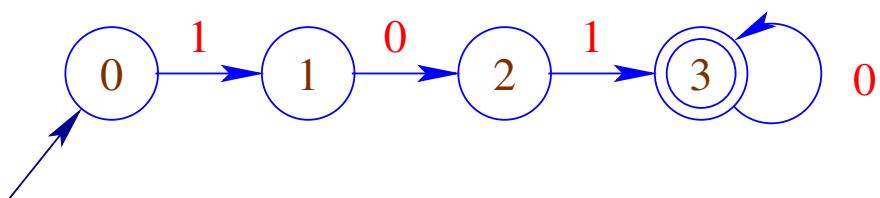
213	t	1	0	1	0	1	0	1	1
42	z	0	1	0	1	0	1	0	0
89	y	1	0	0	1	1	0	1	0

- Nun werden möglicherweise Tupel von Zahlen erst nach Anhängen von geeignet vielen führenden Nullen akzeptiert !
- Der Automat muss so komplettiert werden, dass q bereits akzeptierend ist, wenn von q aus mit Nullen ein akzeptierender Zustand erreicht werden kann.

418

Automaten für Basis-Prädikate:

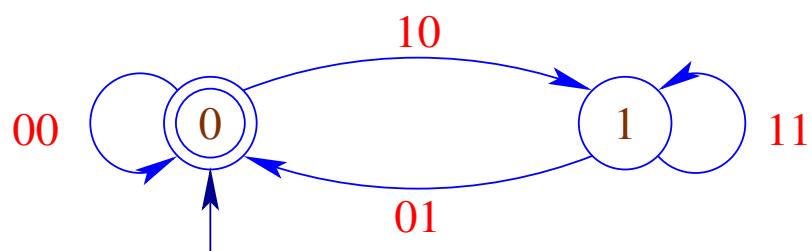
$$x = 5$$



419

Automaten für Basis-Prädikate:

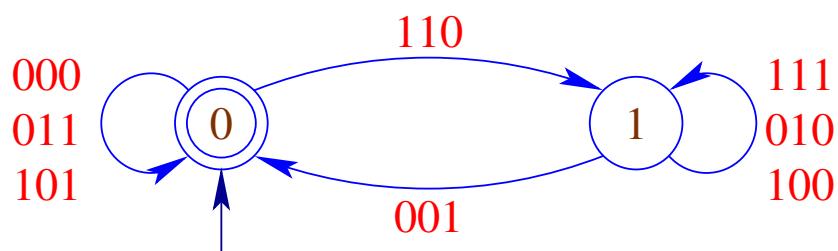
$$x+x = y$$



420

Automaten für Basis-Prädikate:

$$x+y = z$$



421

Ergebnisse:

Ferrante, Rackoff, 1973 : $\text{PSAT} \leq \text{DSPACE}(2^{2^{c \cdot n}})$

Fischer, Rabin, 1974 : $\text{PSAT} \geq \text{NTIME}(2^{2^{c \cdot n}})$

422