Einführung in die Theoretische Informatik

Tobias Nipkow

Fakultät für Informatik
TU München

Sommersemester 2020

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhalt und Gliederung der Vorlesung

- Endliche Automaten und reguläre Ausdrücke
  - Grundlagen der Textanalyse, der lexikalischen Analyse von Programmiersprachen, der Spezifikation und Analyse von Kommunikationsprotokollen, ...
- Kontextfreie Grammatiken
  - Grundlagen der syntaktische Analyse von Programmiersprachen, Parsing, Compilerbau
- Theorie der Berechenbarkeit
  - Untersuchung der Grenzen, was Rechner prinzipiell können.
- Komplexitätstheorie
  - Untersuchung der Grenzen, was Rechner mit begrenzten Ressourcen können.

Universalwerkzeuge der Informatik

Wichtige abstrakte Ziele

Abstraktion von *irrelevanten* Details:
zB nicht x86 sondern Turingmaschine.

Formalisierung durch mathematische Objekte (Mengen, Funktionen, Relationen)

Simulation eines Formalismus durch einen anderen:
zB nicht-deterministische durch deterministische Maschinen

Äquivalenz von Formalismen:
zB endliche Automaten und reguläre Ausdrücke.

Reduktion von einem Problem auf ein anderes:
zB Formeln auf Automaten, ...

Geschichte:

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

Vorkenntnisse und weiterführende Vorlesungen

- Vorkenntnisse:
  - Einführung in die Informatik 1
  - Diskrete Strukturen
- Weiterführende Vorlesungen:
  - Automata und Formal Languages
  - Complexity Theory
  - Logic
  - Model Checking
  - Quantitative Verification
  - Compiler Construction
  - ...

Literatur


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


Dexter Kozen.
Automata and Computability.


Michael Sipser.
Introduction to the Theory of Computation.


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

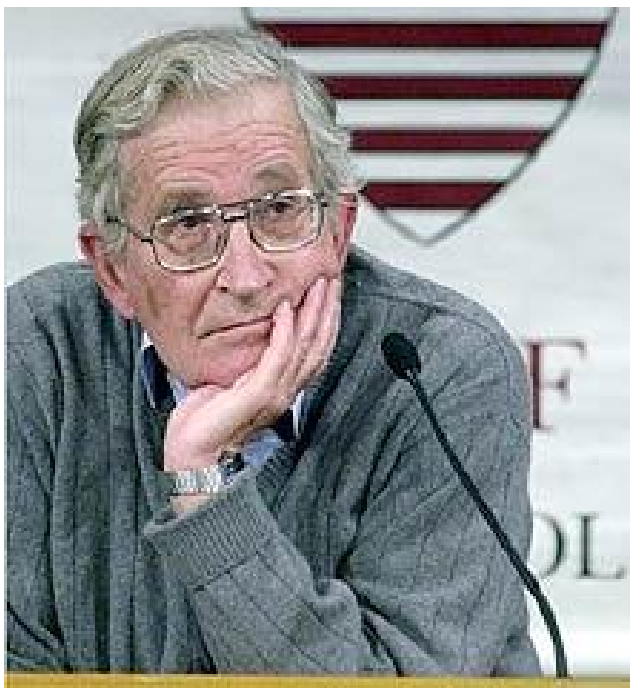

Uwe Schöning.
Theoretische Informatik — kurzgefasst.

Teil 1
Formale Sprachen

1. Historischer Hintergrund

Die Geburt formaler Grammatiken:

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



Noam Chomsky (1928) war (bis 2017) Professor für Linguistik am MIT und ein bedeutender Sprachwissenschaftler. Neben seiner linguistischen Arbeit gilt Chomsky als einer der bedeutendsten Intellektuellen Nordamerikas und ist als scharfer Kritiker der US-amerikanischen Außenpolitik bekannt.
[Quelle: Wikipedia]

- Beispiele von Regeln für den Bau von Sätzen:

$$\begin{aligned} \text{Satz} &\rightarrow \text{Nominalphrase Verbalphrase} \\ \text{Verbalphrase} &\rightarrow \text{Verb Nominalphrase} \\ \text{Nominalphrase} &\rightarrow \text{Artikel Nomen} \end{aligned}$$

- Beispiele von Regeln für den Bau von mathematischen Ausdrücken:

$$\begin{aligned} \text{Expression} &\rightarrow \text{Identifier} \\ \text{Expression} &\rightarrow \text{Expression} + \text{Expression} \\ \text{Expression} &\rightarrow \text{Expression} * \text{Expression} \\ \text{Identifier} &\rightarrow a \end{aligned}$$

- Das Wortproblem: Kann eine gegebene Zeichenkette ("Wort") von den Regeln einer Grammatik erzeugt werden, d.h. ist es eine legale Zeichenkette bzgl. der Grammatik?

  Beispiel: Expression kann $a + a$ erzeugen aber nicht aa
- Recognizer: Programm, welches das Wortproblem für eine gegebene Grammatik löst.
  Recognizer $\rightarrow$ Lexer/Parser $\rightarrow$ Compiler
- Hauptfragen:
  - Für welche Grammatiken gibt es effiziente Recognizer?
  - Gegeben eine Grammatik, wie kann ein Recognizer automatisch konstruiert werden?

2. Grundbegriffe

Definition 2.1

- Ein Alphabet Σ ist eine endliche Menge.
  Bsp: $\{0, 1\}$, ASCII, Unicode.
- Ein Wort/String über Σ is eine endliche Folge von Zeichen aus Σ, zB 010.
- $|w|$ bezeichnet die Länge eines Wortes w.
- Das leere Wort (das einzige Wort der Länge 0) 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: $(ab)^3 = ababab$.
- Σ^* ist die Menge aller Wörter über Σ.
- Eine Teilmenge $L \subseteq \Sigma^*$ ist eine (formale) Sprache.

Beispiel 2.2 (Formale Sprachen)

- Die Menge aller Wörter im Duden (24. Aufl.)
- Die Menge der deutschen Sätze ist keine formale Sprache
- $L_1 = \{\epsilon, ab, abab, ababab, \ldots\} = \{(ab)^n \mid n \in \mathbb{N}\}$
  $(\Sigma_1 = \{a, b\})$
- $L_2 = \{\epsilon, ab, aabb, aaabbb, \ldots\} = \{a^n b^n \mid n \in \mathbb{N}\}$
  $(\Sigma_2 = \{a, b\})$
- $L_3 = \{\epsilon, 1, 100, 1001, 10000, \ldots\} =$
  $\{w \in \{0,1\}^* \mid w$ ist eine binär kodierte Quadratzahl$\}$
  $(\Sigma_3 = \{0, 1\})$
- $\emptyset$
- $\{\epsilon\}$
- ϵ ist keine Sprache

Definition 2.3 (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: $\{ab, b\} \times \{a, bb\} = \{(ab, a), (ab, bb), (b, a), (b, bb)\}$
- $A^n = \{w_1 \cdots w_n \mid w_1, \ldots, w_n \in A\} = \underbrace{A \cdots A}_{n}$

  Bsp: $\{ab, ba\}^2 = \{abab, abba, baab, baba\}$
  Rekursiv: $A^0 = \{\epsilon\}$ und $A^{n+1} = AA^n$
- $A^* = \{w_1 \cdots w_n \mid n \geq 0 \wedge w_1, \ldots, w_n \in A\} = \bigcup_{n \in \mathbb{N}} A^n$
  Bsp: $\{01\}^* = \{\epsilon, 01, 0101, 010101, \ldots\} \neq \{0, 1\}^*$
- $A^+ = AA^* = \bigcup_{n \geq 1} A^n$
  Bsp: $\Sigma^+ =$ Menge aller nicht-leeren Wörter über Σ

Achtung:

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

Einige Rechenregeln:

Lemma 2.4

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

Lemma 2.5

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

Beweis: $A(B \cup C)$

$$
\begin{aligned}
&= \{uv \mid u \in A \wedge v \in B \cup C\} \\
&= \{uv \mid u \in A \wedge (v \in B \vee v \in C)\} \\
&= \{uv \mid u \in A \wedge v \in B \vee u \in A \wedge v \in C\} \\
&= \{uv \mid u \in A \wedge v \in B\} \cup \{uv \mid u \in A \wedge v \in C\} \\
&= AB \cup AC
\end{aligned}
$$

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

Lemma 2.6

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

2.1 Grammatiken

Definition 2.7

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

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

Konventionen:

- $A, B, C, \ldots$ sind Nichtterminale,
- $a, b, c, \ldots$ (und Sonderzeichen wie $+, *, \ldots$) sind Terminale,
- $\alpha, \beta, \gamma, \ldots \in (V \cup \Sigma)^*$
- Produktionen schreiben wir $\alpha \to \beta$ statt $(\alpha, \beta) \in P$.
- Statt $\alpha \to \beta_1, \ldots, \alpha \to \beta_n$ schreiben wir $\alpha \to \beta_1 \mid \cdots \mid \beta_n$

Beispiel 2.8 (Arithmetische Ausdrücke)

- $V = \{\langle \text{Expr} \rangle, \langle \text{Term} \rangle, \langle \text{Faktor} \rangle\}$
- $\Sigma = \{a, b, c, +, *, (,)\}$
- $S = \langle \text{Expr} \rangle$
- Die Menge P enthält folgende Produktionen:

$$
\begin{aligned}
\langle \text{Expr} \rangle &\rightarrow \langle \text{Term} \rangle \\
\langle \text{Expr} \rangle &\rightarrow \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\
\langle \text{Term} \rangle &\rightarrow \langle \text{Factor} \rangle \\
\langle \text{Term} \rangle &\rightarrow \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\
\langle \text{Factor} \rangle &\rightarrow a \mid b \mid c \\
\langle \text{Factor} \rangle &\rightarrow (\langle \text{Expr} \rangle)
\end{aligned}
$$

Definition 2.9

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

$$\alpha \rightarrow_G \alpha'$$

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

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

Beispiel : Für die Grammatik der arithmetischen Ausdrücken gilt

$$a + \langle \text{Term} \rangle + b \quad \rightarrow_G \quad a + \langle \text{Term} \rangle * \langle \text{Factor} \rangle + b$$

Eine Sequenz $\alpha_1 \rightarrow_G \cdots \rightarrow_G \alpha_n$ ist eine Ableitung von α_n aus α_1.

Wenn $\alpha_1 = S$ und $\alpha_n \in \Sigma^*$, dann erzeugt G das Wort α_n.

Die Sprache von G ist die Menge aller Wörter, die von G erzeugt werden. Sie wird mit $L(G)$ bezeichnet.

Beispiel 2.10 (Arithmetische Ausdrücke)

$$
\begin{array}{rcl}
\langle \text{Expr} \rangle & \to & \langle \text{Term} \rangle \\
\langle \text{Expr} \rangle & \to & \langle \text{Expr} \rangle + \langle \text{Term} \rangle \\
\langle \text{Term} \rangle & \to & \langle \text{Factor} \rangle \\
\langle \text{Term} \rangle & \to & \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\
\langle \text{Factor} \rangle & \to & a \mid b \mid c \\
\langle \text{Factor} \rangle & \to & (\langle \text{Expr} \rangle)
\end{array}
$$

Das Startsymbol ist $\langle \text{Expr} \rangle$.

Eine Ableitung:

$$
\begin{aligned}
\langle \text{Expr} \rangle &\to \langle \text{Term} \rangle \to \langle \text{Term} \rangle * \langle \text{Factor} \rangle \\
&\to \langle \text{Factor} \rangle * \langle \text{Factor} \rangle \to a * \langle \text{Factor} \rangle \\
&\to a * (\langle \text{Expr} \rangle) \to a * (\langle \text{Expr} \rangle + \langle \text{Term} \rangle) \\
&\to a * (\langle \text{Term} \rangle + \langle \text{Term} \rangle) \to a * (\langle \text{Factor} \rangle + \langle \text{Term} \rangle) \\
&\to a * (b + \langle \text{Term} \rangle) \to a * (b + \langle \text{Factor} \rangle) \\
&\to a * (b + c)
\end{aligned}
$$

Beispiel 2.11

Zwei Grammatiken für $M = \{a^n b^n c^n \mid n \geq 0\}$?

G_1: $S \to abcS \mid \epsilon$
$ca \to ac$
$ba \to ab$
$cb \to bc$

Es gilt $L(G_1) \neq M$ weil
$S \to abcS \to abcabcS \to abcabc$

Aber $M \subseteq L(G_1)$

Bsp:

$S \to abcS \to abcabcS \to abcabc \to abacbc \to aabcbc \to aabbcc$

G_2: $S \to aBSc \mid \epsilon$
$Ba \to aB$
$Bb \to bb$
$Bc \to bc$

Es gilt: $L(G_2) = M$

Bsp: $S \to aBSc \to aBaBScc \to aBaBcc \to aaBBcc \to$
$aaBbcc \to aabbcc$

Definition 2.12 (Iteration von $\to_G$)

$$\alpha \to_G^0 \alpha$$

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

Reflexive transitive Hülle:

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

Transitive Hülle:

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

Beispiel: $\langle \text{Expr} \rangle \to_G^{11} a * (b + c)$ und somit $\langle \text{Expr} \rangle \to_G^* a * (b + c)$.

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

2.2 Die Chomsky-Hierarchie

Eine Grammatik G ist vom

- Typ 0 immer.
- Typ 1 falls für jede Produktion $\alpha \to \beta$ außer $S \to \epsilon$ gilt $|\alpha| \leq |\beta|$
- Typ 2 falls G vom Typ 1 ist und für jede Produktion $\alpha \to \beta$ gilt $\alpha \in V$.
- Typ 3 falls G vom Typ 2 ist und für jede Produktion $\alpha \to \beta$ außer $S \to \epsilon$ gilt $\beta \in \Sigma \cup \Sigma V$.

Offensichtlich gilt:

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

Grammatiken und Sprachklassen:

Typ 3	Rechtslineare Grammatik	Reguläre Sprachen
Typ 2	Kontextfreie Grammatik	Kontextfreie Sprachen
Typ 1	Kontextsensitive Grammatik	Kontextsens. Sprachen
Typ 0	Phrasenstrukturgrammatik	Rekursiv aufzählbare Sprachen

Satz 2.13

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

Bemerkung Wir benutzen später eine etwas liberalere Definition von kontextfreien Grammatiken, die aber die gleiche Klasse von kontextfreien Sprachen erzeugt.

Das Wortproblem:

Gegeben: eine Grammatik G, ein Wort $w \in \Sigma^*$
Frage: Gilt $w \in L(G)$?

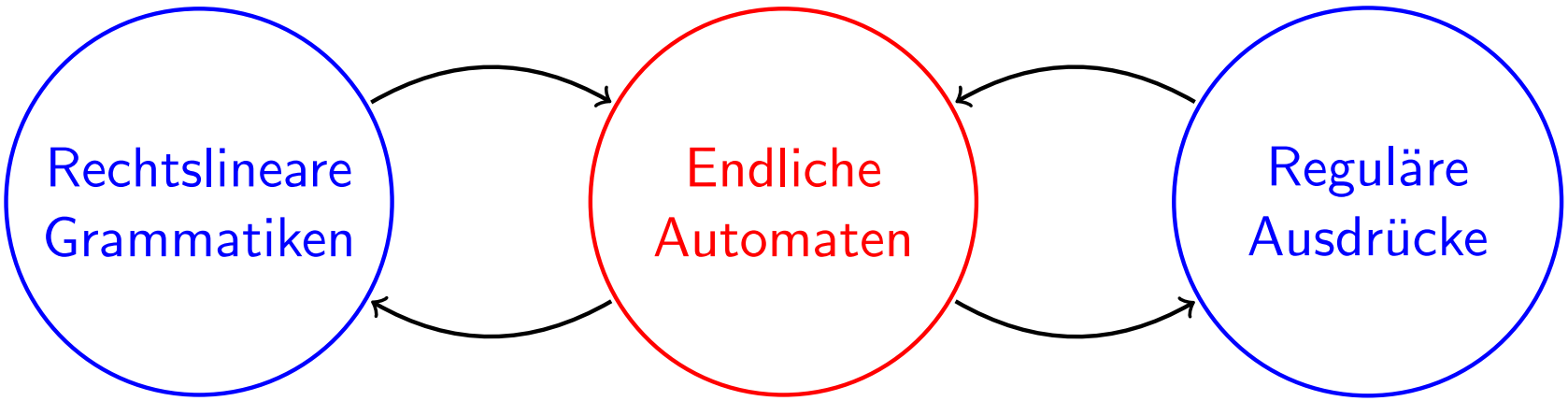
In den kommenden Wochen untersuchen wir das Wortproblem für Grammatiken vom Typ 3 und Typ 2.

Wir studieren Algorithmen, die für eine gegebene Grammatik G einen Automaten A_G konstruieren, und untersuchen ihre Laufzeit.

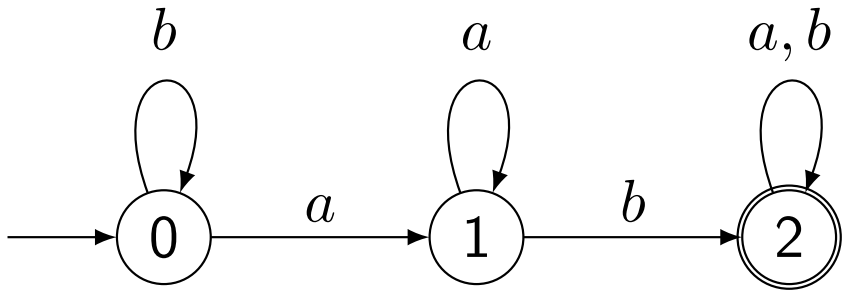
A_G ist eine abstrakte Beschreibung eines Programms zur Lösung des Wortproblems.

Der Algorithmus mit G als Eingabe und A_G als Ausgabe ist eine abstrakte Beschreibung eines Programms für die automatische Synthese von Recognizern (lex, yacc).

3. Reguläre Sprachen



3.1 Deterministische endliche Automaten



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

Akzeptierte Sprache: Menge der Wörter, die vom Startzustand in einen Endzustand führen.

Recognizer, der nur einmal das Wort durchläuft und es in linearer Zeit akzeptiert oder ablehnt.

Definition 3.1

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 $\delta : \ Q \times \Sigma \to Q$,
- einem Startzustand $q_0 \in Q$, und
- einer Menge $F \subseteq Q$ von Endzuständen (akzeptierenden Zust.)

Die von M akzeptierte Sprache ist

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

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

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

Definition 3.2

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

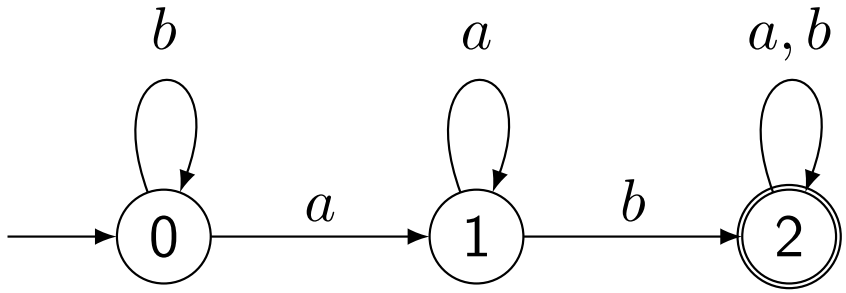
Eigenschaften von $\hat{\delta}$:

- $\hat{\delta}(q, a) = \delta(q, a)$.
- $\hat{\delta}(q, ww') = \delta(\hat{\delta}(q, w), w')$.

Graphische Darstellung

- Endliche Automaten können durch gerichtete und markierte Zustandsgraphen veranschaulicht werden:
  - Knoten $\widehat{=}$ Zuständen
  - Kanten $\widehat{=}$ Übergängen: $p \xrightarrow{a} q \;\widehat{=}\; \delta(p, a) = q$
- Der Anfangszustand wird durch einen Pfeil, Endzustände werden durch doppelte Kreise gekennzeichnet.

Beispiel 3.3

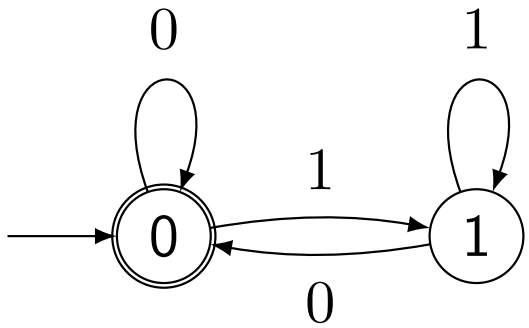


Akzeptierte Sprache = alle Wörter über $\{a, b\}$, die ab enthalten.

- Jedes Wort, das akzeptiert wird, enthält ab.
  Sei w ein Wort, welches akzeptiert wird.
  Betrachte in der Zustandsfolge für w den Zeitpunkt, in dem Zustand 1 zum letzten Mal besucht wird (existiert weil die Folge in Zustand 2 endet).
  Ummitelbar davor wird a gelesen und unmittelbar danach b
- Jedes Wort, das ab enthält, wird akzeptiert.
  Für alle $q \in \{0, 1, 2\}$ gilt: $\hat{\delta}(q, ab) = 2$.
  Für alle $w' \in \{a, b\}^*$ gilt: $\hat{\delta}(2, w') = 2$.
  $\Longrightarrow \hat{\delta}(0, wabw') = \hat{\delta}(\hat{\delta}(\hat{\delta}(0, w), ab), w') = \hat{\delta}(2, w') = 2$

Beispiel 3.4

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

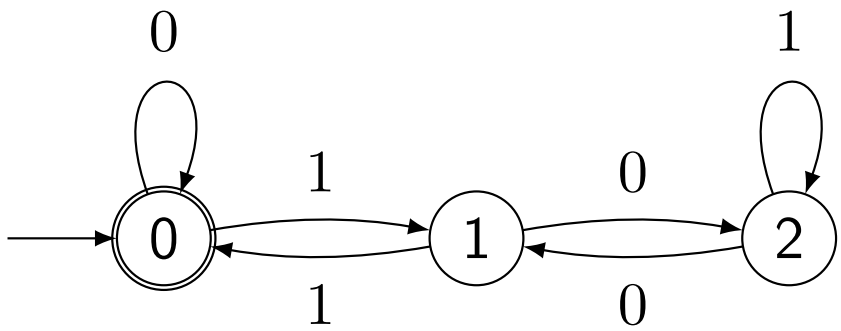


Der DFA akzeptiert genau die $w \in \{0,1\}^*$ so dass $\#w$ gerade ist.

Beweis:

Für alle $w \neq \epsilon$, $\#w$ ist gerade gdw w endet mit 0.
1. $\hat{\delta}(0, w0) = \delta(\hat{\delta}(0,w), 0) = 0 \in F$, daher $w0 \in L(A)$
2. $\hat{\delta}(0, w1) = \delta(\hat{\delta}(0,w), 1) = 1 \notin F$, daher $w1 \notin L(A)$
Für $w = \epsilon$, $\hat{\delta}(0, w) = 0$, daher $\epsilon \in L(A)$ und $\#\epsilon = 0$. □

Beispiel 3.5



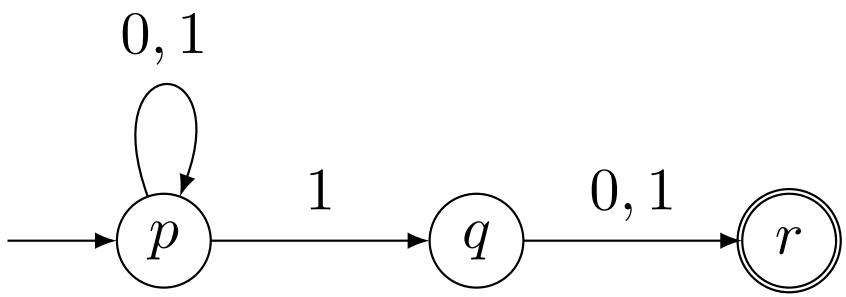
Der DFA akzeptiert genau die Wörter $w \in \{0,1\}^*$ mit: $\#w$ ist ein Vielfaches von 3.

3.2 Nichtdeterministische endliche Automaten (NFAs)

Verallgemeinerung von DFAs: Aus einem Zustand eines NFAs können $0, 1$, oder mehr Transitionen mit derselben Beschriftung ausgehen.

Beispiel 3.6

Erkennung der Binärzahlen, deren vorlezte Ziffer 1 ist:



Wort wird akzepiert gdw es einen Weg zum Endzustand gibt.

Intuitive Vorstellung:

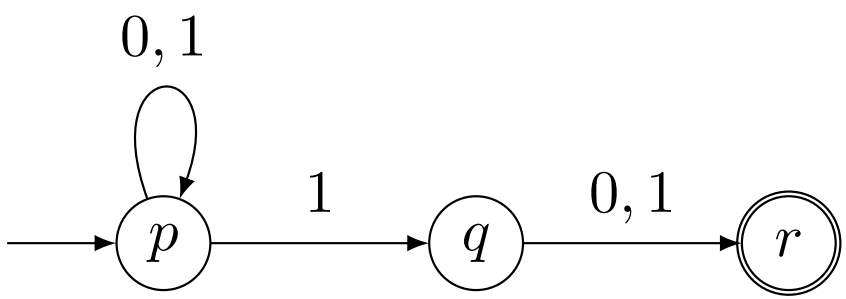
- Der Automat durchsucht alle möglichen Wege
- Der Automat "rät" den richtigen Weg.

Definition 3.7

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
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$
  $\mathcal{P}(Q)$ = Menge aller Teilmengen von $Q = 2^Q$.
  Alternative: Relation $\delta \subseteq Q \times \Sigma \times Q$.

Beispiel



δ	0	1
p	$\{p\}$	$\{p, q\}$
q	$\{r\}$	$\{r\}$
r	$\emptyset$	$\emptyset$

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

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

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

Intuition:

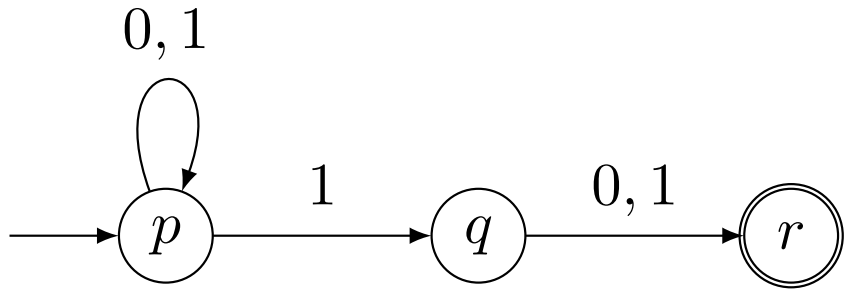
$\hat{\bar{\delta}}(S, w)$ ist Menge aller Zustände,
die sich von einem Zustand in S aus mit w erreichen lassen.

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

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

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

Beispiel



$$
\begin{aligned}
& \hat{\bar{\delta}}(\{p,q\},10) \\
= \quad & \hat{\bar{\delta}}(\bar{\delta}(\{p,q\},1),0) \\
= \quad & \hat{\bar{\delta}}(\delta(p,1) \cup \delta(q,1),0) \\
= \quad & \hat{\bar{\delta}}(\{p,q,r\},0) \\
= \quad & \bar{\delta}(\{p,q,r\},0) \\
= \quad & \delta(p,0) \cup \delta(q,0) \cup \delta(r,0) \\
= \quad & \{p\} \cup \{r\} \cup \emptyset \;=\; \{p,r\}
\end{aligned}
$$

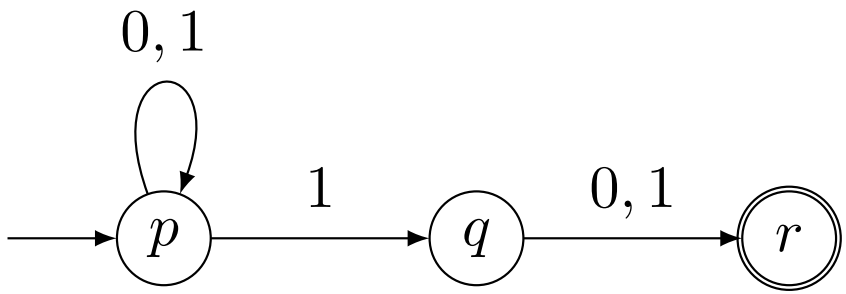
- NFA wird (typischerweise) nicht direkt als Recognizer verwendet
- sonder ist meist Zwischenstation auf dem Weg zum DFA oder kompakte Darstellung einer Sprache

3.3 Äquivalenz von NFA und DFA

Satz 3.8

Für jeden NFA N gibt es einen DFA M mit $L(N) = L(M)$.

Beispiel



Beweis:

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

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

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

Dann gilt:

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

□

Dies nennt man die Potenzmengen- oder Teilmengenkonstruktion.

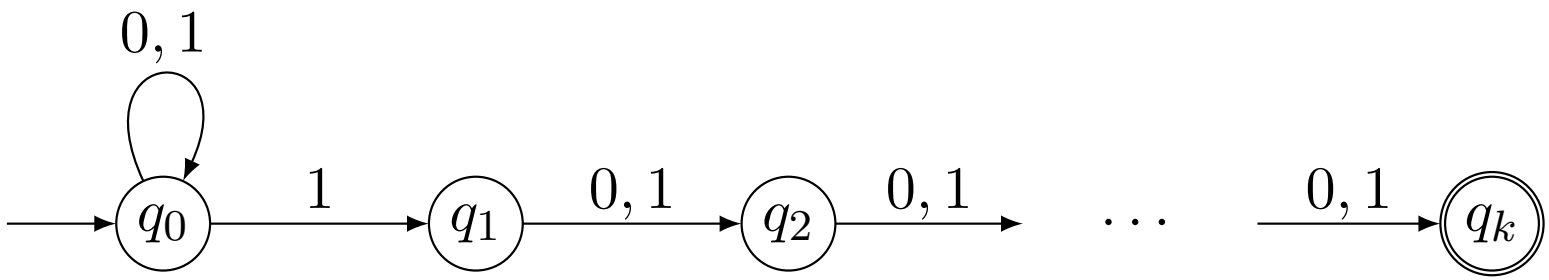
In der Praxis werden nur die aus q_0 erreichbaren Zustände konstruiert.

Trotzdem: Für einen NFA mit n Zuständen kann der entsprechende DFA bis zu 2^n Zustände haben.

Beispiel 3.9

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

Ein NFA für diese Sprache:



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

Lemma 3.10

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

Im *schlimmsten* Fall ist ein exponentieller Sprung unvemeidlich.

Beweis:

Sei M ein DFA mit $L(M) = L_k$.

- Zuerst zeigen wir für alle $w_1, w_2 \in \{0,1\}^k$:
  wenn $w_1 \neq w_2$ dann $\hat{\delta}(q_0, w_1) \neq \hat{\delta}(q_0, w_2)$.
  Annahme: Es gibt $w_1, w_2 \in \{0,1\}^k$ mit $w_1 \neq w_2$ aber $\hat{\delta}(q_0, w_1) = \hat{\delta}(q_0, w_2)$. Wir leiten einen Widerspruch ab:
- Sei $w_1 = wa_i \dots a_k$ und $w_2 = wb_i \dots b_k$ mit $a_i \neq b_i$

  OE sei $a_i = 1$, $b_i = 0$.

  Es gilt einerseits: $w_1 0^{i-1} = wa_i \dots a_k 0^{i-1} \in L_k$
  $w_2 0^{i-1} = wb_i \dots b_k 0^{i-1} \notin L_k$

  Aber es gilt auch: $\hat{\delta}(q_0, w_1 0^{i-1}) = \hat{\delta}(\hat{\delta}(q_0, w_1), 0^{i-1}) =$
  $\hat{\delta}(\hat{\delta}(q_0, w_2), 0^{i-1}) = \hat{\delta}(q_0, w_2 0^{i-1})$ ↯
- Es folgt: M hat mindestens 2^k Zustände.

□

3.4 NFAs mit ϵ-Übergängen

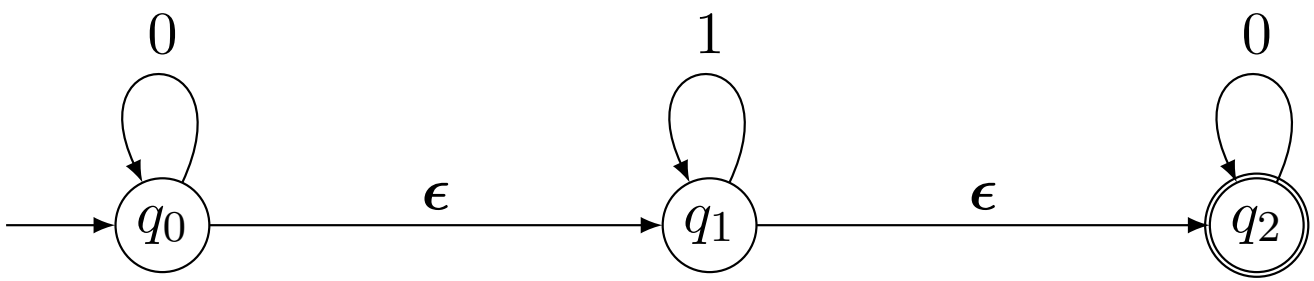
- Diese Erweiterung von NFAs macht sie nicht mächtiger (man bekommt wieder genau die regulären Sprachen)
- aber man kann manche Sprachen einfacher beschreiben als nur mit NFAs.

Definition 3.11

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

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

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



Akzeptiert: ϵ, 00, 11, . . . Nicht akzeptiert: , . . .

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

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

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

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

- Falls N das leere Wort ϵ akzeptiert, also falls

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

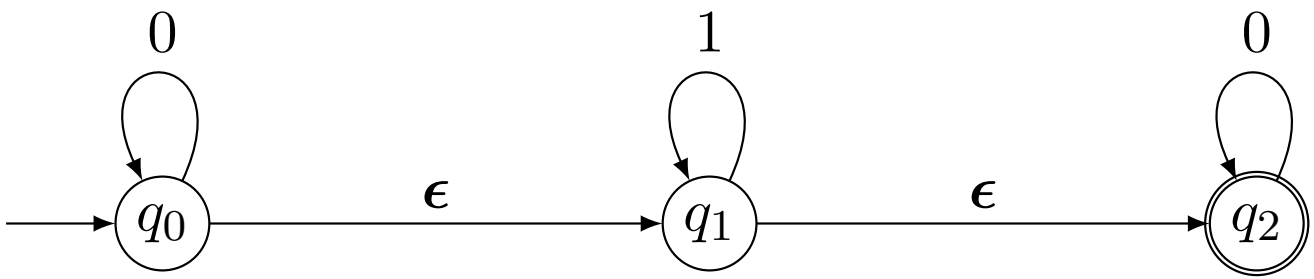
  dann setze $F' := F \cup \{q_0\}$, sonst setze $F' := F$.

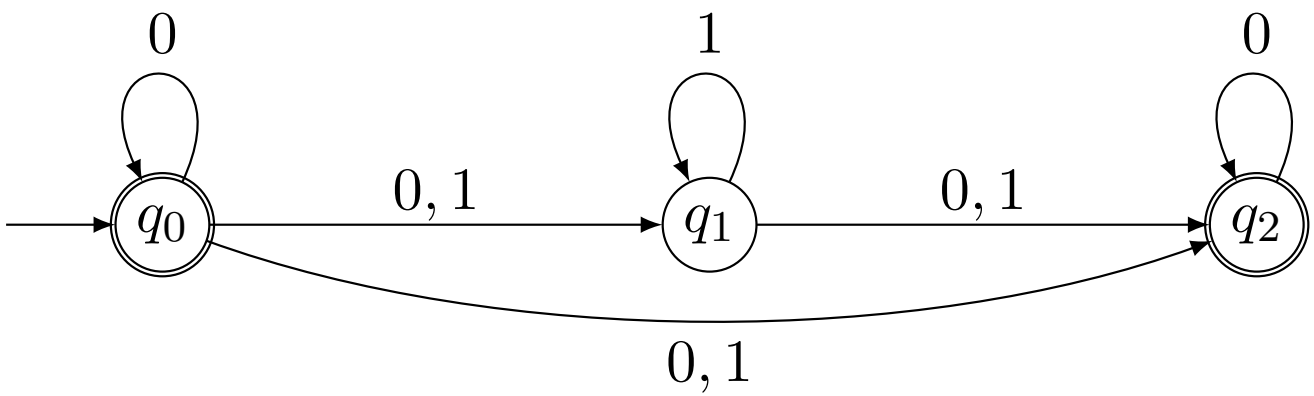
Damit gilt per definitionem:

Jeder $\boldsymbol{\epsilon}$-NFA ist äquivalent zu einem NFA.

Ab jetzt: "ϵ-Übergang" und "ϵ-NFA".

Beispiel 3.12





Fazit: Die Automatentypen

- DFA
- NFA
- ϵ-NFA

sind gleich mächtig.

3.5 Rechtslinearen Grammatiken

Erinnerung: rechtslineare Grammatiken haben nur Produktionen der Form $A \to a$, $A \to aB$ und $S \to \epsilon$.

Satz 3.13

Für jeden DFA M gibt es eine rechtslineare Grammatik G mit $L(M) = L(G)$.

Beweis:

Sei $M = (Q, \Sigma, \delta, q_0, F)$. Die Grammatik $G = (V, T, P, S)$ mit

- $V = Q$, $T = \Sigma$, $S = q_0$,
- $(q_1 \to aq_2) \in P$ gdw $\delta(q_1, a) = q_2$,
- $(q_1 \to a) \in P$ gdw $\delta(q_1, a) \in F$, und
- $(q_0 \to \epsilon) \in P$ gdw $q_0 \in F$,

ist vom Typ 3 und erfüllt $L(G) = L(M)$:
Es gilt (Induktion über n):

$$q_0 \to_G^* a_1 \cdots a_n \qquad \text{gdw} \qquad \hat{\delta}(q_0, a_1 \ldots a_n) \in F$$

□

Satz 3.14

Für jede rechtslineare Grammatik G gibt es einen NFA M mit $L(G) = L(M)$.

Beweis:

Sei $G = (V, \Sigma, P, S)$ eine rechtslineare Grammatik.
Der ϵ-NFA $N = (Q, \Sigma, \delta, q_0, F)$ mit

- $Q = V \cup \{q_f\}$ (wobei q_f neu: $q_f \notin V$)
- $Y \in \delta(X, a)$ gdw $(X \to aY) \in P$
- $q_f \in \delta(X, u)$ gdw $(X \to u) \in P$ wobei $u \in \Sigma \cup \{\boldsymbol{\epsilon}\}$
- $q_0 = S$
- $F = \{q_f\}$

erfüllt $L(N) = L(G)$:
Es gilt (Induktion über n):

$$q_f \in \hat{\delta}(S, a_1 \dots a_n) \qquad \text{gdw} \qquad S \to_G^* a_1 \dots a_n$$

3.6 Reguläre Ausdrücke

Reguläre Ausdrücke sind eine weitere Notation für die Definition von formalen Sprachen.

Definition 3.15

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
  - $\alpha\beta$
  - $\alpha \mid \beta$ (oft $\alpha + \beta$ geschrieben)
  - α^*

Nichts sonst ist ein regulärer Ausdruck.

Notation:

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

Definition 3.16

Die Sprache $L(\gamma)$ eines regulären Ausdrucks γ ist rekursiv definiert:

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

Beispiel 3.17

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

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

  Hausaufgabe!

Beispiel 3.18

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

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

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

Erweiterte reguläre Ausdrücke in UNIX:

$$
\begin{aligned}
. &= a_1|\ldots|a_n \text{ wobei } \Sigma = \{a_1, \ldots a_n\} \\
[a_1 \ldots a_n] &= a_1|\ldots|a_n \\
[\hat{}\, a_1 \ldots a_n] &= b_1|\ldots|b_m \text{ wobei } \{b_1, \ldots, b_m\} = \Sigma \setminus \{a_1, \ldots a_n\} \\
\alpha? &= \epsilon|\alpha \\
\alpha+ &= \alpha\alpha^* \\
\alpha\{n\} &= \alpha \ldots \alpha \ (n \text{ copies})
\end{aligned}
$$

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

- $P(\emptyset)$
- $P(\boldsymbol{\epsilon})$
- $P(a)$ für alle $a \in \Sigma$
- $P(\alpha) \wedge P(\beta) \Rightarrow P(\alpha\beta)$
- $P(\alpha) \wedge P(\beta) \Rightarrow P(\alpha \mid \beta)$
- $P(\alpha) \Rightarrow P(\alpha^*)$

Komfortabler Spezialfall der Induktion über die Länge von γ.

Satz 3.19 (Kleene 1956)

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

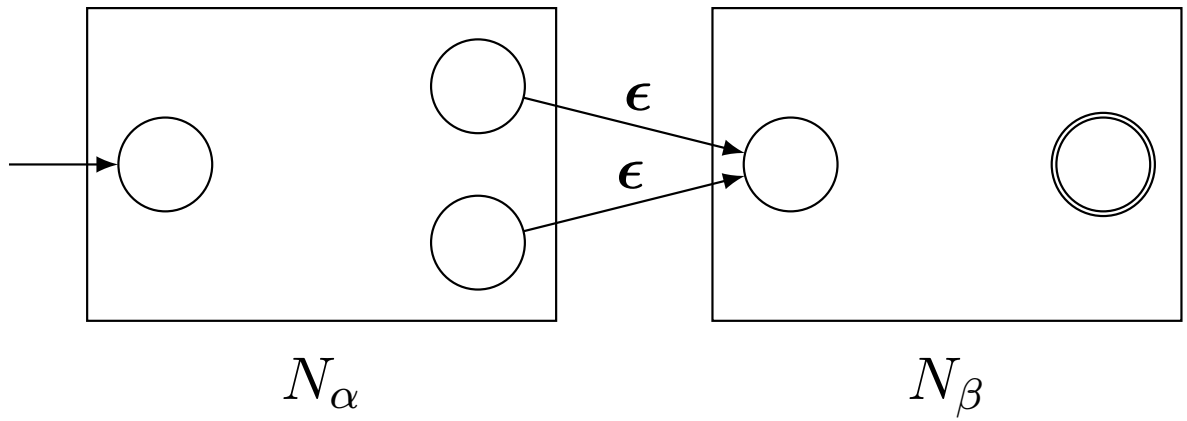
Beweis:

"$\Longrightarrow$":
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.

Fall $\gamma = \alpha\beta$:

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

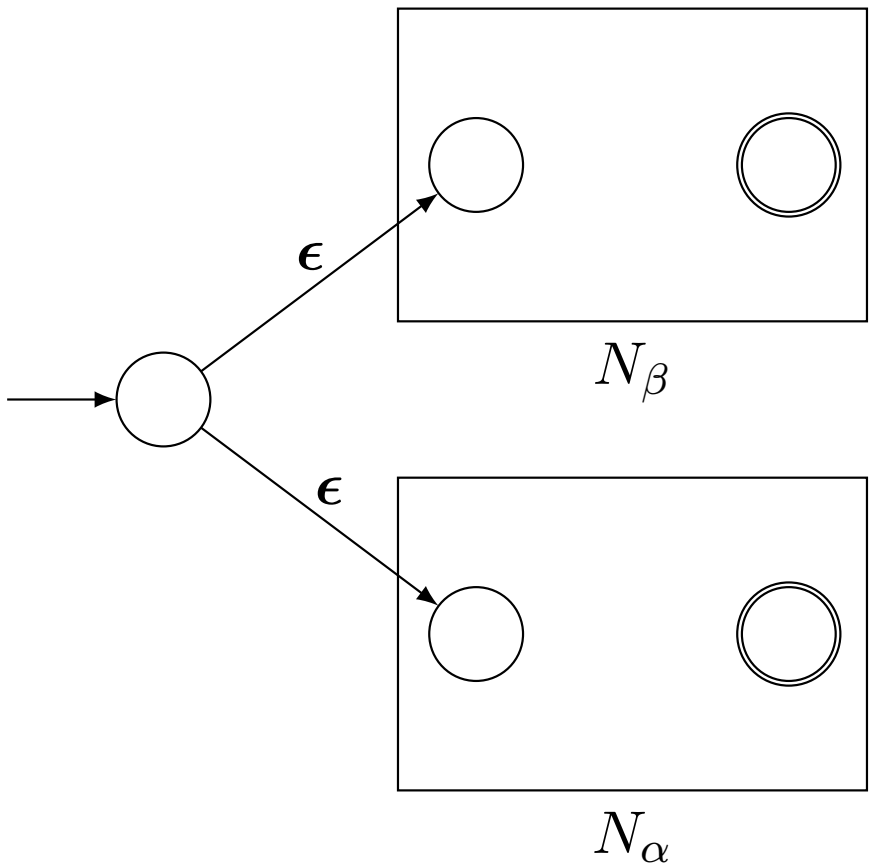


Formal:

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

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

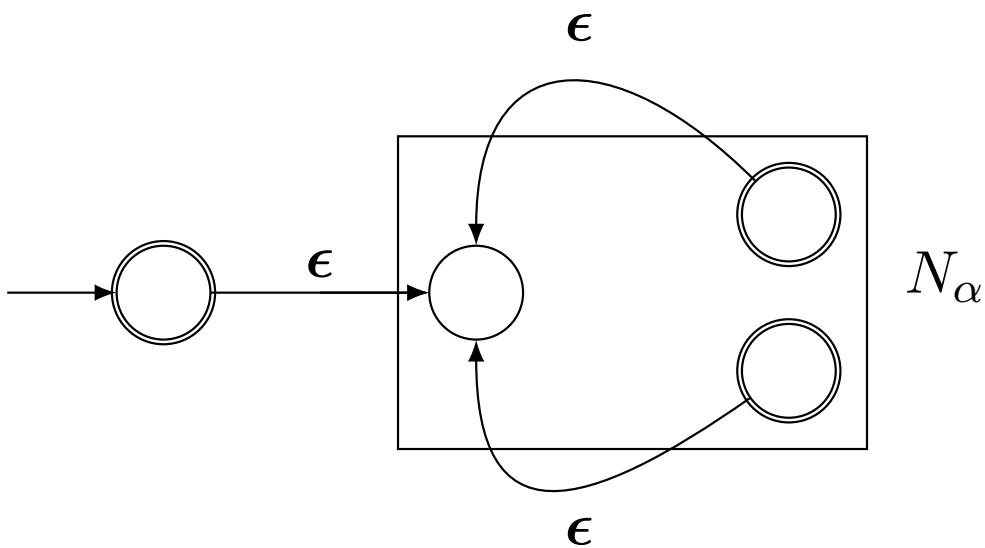
Nach Induktionsannahme können wir ϵ-NFAs N_α und N_β konstruieren mit $L(N_\alpha) = L(\alpha)$ und $L(N_\beta) = L(\beta)$.



Fall $\gamma = \alpha^*$:

Nach Induktionsannahme können wir einen ϵ-NFA N_α konstruieren mit

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



"$\Longleftarrow$":
Sei $M = (Q, \Sigma, \delta, q_1, F)$ ein DFA.
(Funktioniert analog auch für NFA)
Wir konstruieren einen RE γ mit $L(M) = L(\gamma)$.

Sei $Q = \{q_1, \ldots, q_n\}$. 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 $i, j \in \{1, \ldots, n\}$ und $k \in \{0, \ldots, n\}$ können wir einen RE α_{ij}^k konstruieren mit $L(\alpha_{ij}^k) = R_{ij}^k$.

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

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

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

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

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

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

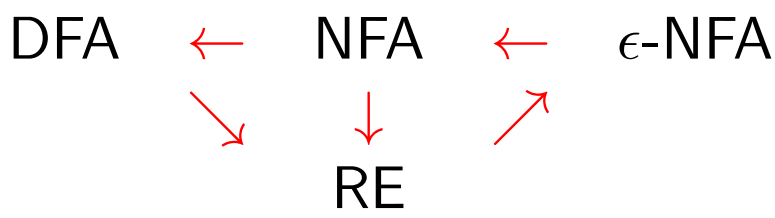
$$\alpha_{ij}^{k+1} = \alpha_{ij}^{k} \mid \alpha_{i(k+1)}^{k} (\alpha_{(k+1)(k+1)}^{k})^* \alpha_{(k+1)j}^{k}$$

Somit gilt

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

wobei $F = \{i_1, \ldots, i_r\}$. □

Unsere Konversionen auf einen Blick:



RE $\to$ ϵ-NFA:	RE der Länge $n \rightsquigarrow O(n)$ Zustände
ϵ-NFA $\to$ NFA:	$Q \rightsquigarrow Q$
NFA $\to$ DFA:	n Zustände $\rightsquigarrow O(2^n)$ Zustände
FA $\to$ RE:	n Zustände $\rightsquigarrow$ RE der Länge $O(n4^n)$

Beweis FA$\to$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: $O(n4^n)$

3.7 Abschlusseigenschaften regulärer Sprachen

Satz 3.20

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 $R = L(A)$ für einen $A = (Q, \Sigma, \delta, q_0, F)$ DFA.
Betrachte $A' = (Q, \Sigma, \delta, q_0, Q \setminus F)$.
Dann ist $L(A') = \overline{L(A)} = \overline{R}$

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

$R_1 \setminus R_2 =$ □

Bemerkung

Komplementierung ($\overline{R}$) 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.

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 3.21
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: $|Q_1 \times Q_2| = |Q_1||Q_2|$.

Beweis:

Durch Induktion über w läßt sich zeigen:

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

Damit gilt:

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

□

Funktioniert Durchschnitt durch Produkt auch für NFAs?

Definition 3.22

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 3.23

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 $q_0' \xrightarrow{\boldsymbol{\epsilon}} f$ für alle $f \in F$.
- Mache q_0 zum (einzigen) Endzustand.

Dann gilt $L(M') = A^R$. □

3.8 Rechnen mit regulären Ausdrücken

Definition 3.24
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):
$(\alpha \mid \beta) \equiv (\beta \mid \alpha)$ aber $(\alpha \mid \beta) \neq (\beta \mid \alpha)$.

Null und Eins:

Lemma 3.25

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

Lemma 3.26

Assoziativität:

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

Kommutativität:

- $\alpha \mid \beta \equiv \beta \mid \alpha$

Distributivität:

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

Idempotenz:

- $\alpha \mid \alpha \equiv \alpha$

Stern:

Lemma 3.27

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

Beispiel 3.28

Herleitung einer Äquivalenz aus obigen Lemmas:

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

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

Satz 3.29 (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.

3.9 Pumping Lemma

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

Satz 3.30 (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*
- $\forall i \geq 0.\ uv^i w \in R$.

- Die logische Struktur des Satzes:

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

- Das Pumping Lemma als Spiel:
  - Spieler I wählt eine reguläre Sprache R
  - Spieler II wählt ein $n > 0$
  - Spieler I wählt ein $z \in R$ mit $|z| \geq n$
  - Spieler II wählt eine Zerlegung uvw von z

  Spieler II gewinnt, wenn die Zerlegung die gewünschten Eigenschaften erfüllt, sonst gewinnt Spieler I.

  Das Pumping Lemma sagt, dass Spieler II eine *Gewinnstrategie* hat: Wenn er richtig spielt, dann gewinnt er jede Partie.

- 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.

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

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

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

Wir teilen z wie folg auf: $\underbrace{a_1 \dots a_i}_{u} \underbrace{a_{i+1} \dots a_j}_{v} \underbrace{a_{j+1} \dots a_m}_{w}$

Damit gilt:

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

□

[Darf A NFA sein?]

Fazit:

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

Pumping-Lemma-Zahl für $L(ab^*c)$?

Pumping-Lemma-Zahl für $\{aaaa\}$?

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

Beispiel für die Anwendung des Pumping Lemmas:

Satz 3.31

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 $a^{n-|v|}b^n = uw \in L$. ↯ □

"Endliche Automaten können nicht unbegrenzt zählen"

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

Beispiel für die Anwendung des Pumping Lemmas:

Satz 3.32

$L = \{0^{m^2} \mid m \geq 0\}$ *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:

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

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

Beispiel für die Anwendung des Pumping Lemmas:

Satz 3.33

Die Sprache der balancierten Klammer-Ausdrücke über $\{(,)\}$
ist nicht regulär.

Beweis:

Aufgabe. □

Satz 3.34

Die Sprache $Arith$ *der arithmetischen Ausdrücken ist nicht regulär.*

Beweis:

Angenommen, $Arith$ sei doch regulär.
Dann gibt es einen DFA A mit $L(A) = Arith$.
Ersetze alle Transitionen von A, die nicht mit $($ oder $)$ beschriftet sind durch ϵ-Transitionen.
Der resultierende ϵ-NFA erkennt die Sprache der balancierten Klammer-Ausdrücke. Widerspruch zu Satz 3.33. □

Bemerkung

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

$\Rightarrow$ Pumping-Lemma hinreichend aber nicht notwendig um Nicht-Regularität zu zeigen.

$$\text{regulär} \subset \text{Pumping-Lemma gilt} \subset \text{alle Sprachen}$$

3.10 Entscheidungsverfahren

Wir untersuchen Entscheidungsprobleme für reguläre Sprachen, d.h. Probleme der Gestalt

Eingabe: Ein oder mehrere Objekte, die reguläre Sprachen beschreiben (DFA, NFA, RE, Typ 3 Gram., . . .)
Frage: Haben diese Objekte eine Eigenschaft X?

Ein (Entscheidungs)Problem ist entscheidbar wenn es einen Algorithmus gibt, der bei jeder Eingabe in endlicher Zeit die richtige Antwort gibt.

In der zweiten Hälfte der Vorlesung wird gezeigt, dass nicht alle Entscheidungsprobleme für Grammatiken enstcheidbar sind!

Welche Entscheidungsprobleme sind für rechtslineare Grammatiken (oder DFA; NFA; RE . . .) entscheidbar?

Wie hängt die Laufzeit des Algorithmus mit der Beschreibung zusammen?

Definition 3.35

Sei D ein DFA, NFA, RE, rechtslineare Grammatik ....

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

Leerheitsproblem: Gegeben D, gilt $L(D) = \emptyset$?

Endlichkeitsproblem: Gegeben D, ist $L(D)$ endlich?

Äquivalenzproblem: Gegeben D_1, D_2, gilt $L(D_1) = L(D_2)$?

Lemma 3.36

Das Wortproblem ist für ein Wort w und DFA M in Zeit $O(|w| + |M|)$ entscheidbar.

Lemma 3.37

Das Wortproblem ist für ein Wort w und NFA N in Zeit $O(|Q|^2|w| + |N|)$ entscheidbar.

Beweis:

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

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

Lemma 3.38

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

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 $O(|Q|)$.

Lemma 3.39

Das Endlichkeitsproblem ist für DFAs oder NFAs entscheidbar.

Beweis:

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

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

$Reach(K) =$ $R := \emptyset$; $W := K$
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

□

Lemma 3.40

Das Äquivalenzproblem ist für DFAs entscheidbar.

Beweis:

Folgt direkt aus

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

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

Satz 3.41

Das Äquivalenzproblem für DFAs ist in Zeit $O(|Q_1||Q_2||\Sigma|)$ entscheidbar.

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

Korollar 3.42

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 $\quad\rightsquigarrow$
2 DFAs mit 2^m und 2^n Zuständen $\quad\rightsquigarrow$
Äquivalenztest in Zeit $O(2^m 2^n)$ □

Korollar 3.43

Das Äquivalenzproblem für reguläre Ausdrücke ist entscheidbar.

Fazit:

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

3.11 Automaten und Gleichungssysteme

Nicht mehr *Beweisen* von Äquivalenzen

$$\text{Gilt } XX^* \equiv X^*X \text{ für alle } X?$$

sondern *Lösen*:

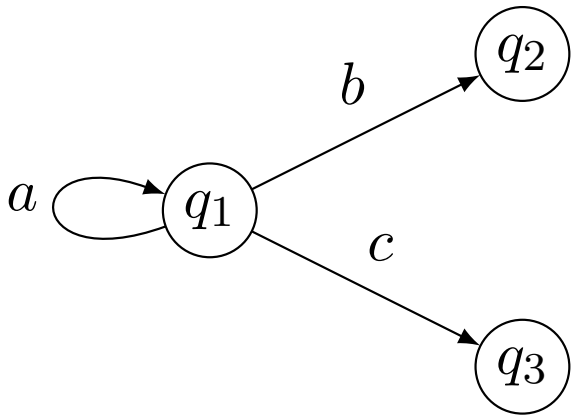
$$\text{Für welches } X \text{ gilt } X \equiv aX \mid b \text{ ?}$$

Anwendung:

$$\text{Automat} \rightsquigarrow \text{Gleichungssystem} \rightsquigarrow \text{RE}$$

Beispiel 3.44

Ein Automatenfragment:



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

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.

Satz 3.45 (Ardens Lemma)

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

$$X = AX \cup B \quad \Longrightarrow \quad X = A^*B$$

Korollar 3.46

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

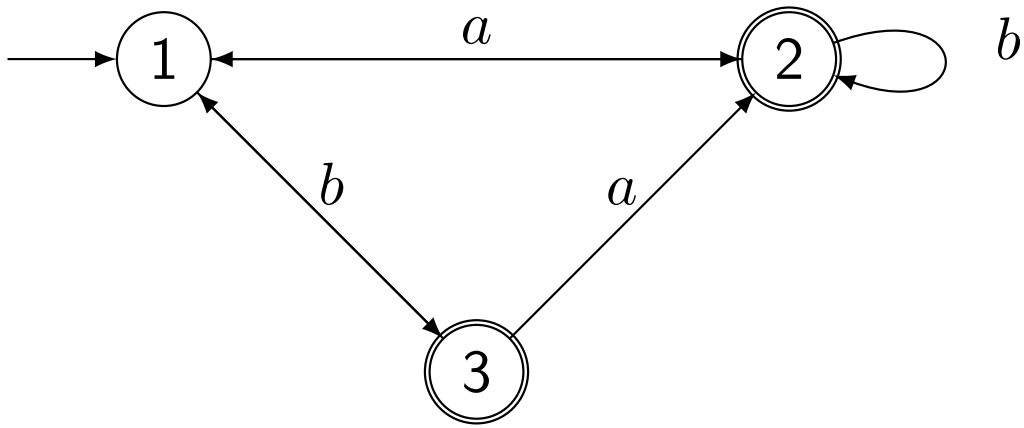
$$X \equiv \alpha X \mid \beta \quad \Longrightarrow \quad X \equiv \alpha^*\beta$$

Bemerkungen

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

Umwandlung eines DFAs in einen äquivalenten regulären Ausdruck:

Beispiel 3.47



Äquivalentes Gleichungssystem:

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

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

Lösen des Gleichungssystems:

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

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

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

Zurück einsetzen:

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

Ausmultiplizieren und X_i ausklammern:

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

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

X_3 ist gelöst, in 1. Gleichung einsetzen:

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

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:

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

Umwandlung eines FAs in einen äquivalenten regulären Ausdruck:

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

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

$$a_{ij} := c_1 \mid \cdots \mid c_k \quad \text{falls } \{c_1, \ldots, c_k\} = \{c \in \Sigma \mid q_i \xrightarrow{c} q_j\}$$
$$\text{wobei } a_{ij} := \emptyset \text{ falls } q_i \xrightarrow{c} q_j \text{für kein } c \in \Sigma$$

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

- Löse das System durch schrittweise Elimination von Variablen mit Hilfe von Ardens Lemma für REs (Korollar 3.46).
- Ist k der Startzustand, so beschreibt X_k die vom Automaten akzeptierte Sprache.

Beweis von Ardens Lemma:

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

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

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.

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

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

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

$A^*B \subseteq X$: $\quad w \in A^*B = \bigcup_{i \in \mathbb{N}} A^i B$

$$\implies \exists n.\ w \in A^n B$$

$$\implies w \in X$$

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

$$\epsilon \notin A$$

$$\implies \forall u \in A^{n+1}.\ |u| \geq n+1$$

$$\implies w \notin A^{n+1}X$$

$$\implies w \in \bigcup_{i \leq n} A^i B \qquad \text{(wegen } (*)\text{)}$$

$$\implies w \in \bigcup_{i \in \mathbb{N}} A^i B = A^*B$$

□

3.12 Minimierung endlicher Automaten

Wir zeigen, dass jede reguläre Sprache einen einzigen minimalen DFA hat und geben Algorithmen an, die diesen DFA konstruieren.

1. Beispiele
2. Algorithmen
3. Minimalitätsbeweis

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 unterscheidbar 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 unterscheidbar sind, d.h. wenn für alle $w \in \Sigma^*$ gilt:

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

- Gilt $p \in F$ und $q \notin F$, dann sind p und q unterscheidbar.
- Sind $\delta(p, a)$ und $\delta(q, a)$ unterscheidbar, dann auch p und q.

  $\Rightarrow$ Unterscheidbarkeit pflanzt sich rückwärts fort.

Berechnung äquivalenter Zustände eines DFA

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 unterscheidbar

Lemma 3.48

Am Ende gilt: U ist Menge aller unterscheidbaren Zustandspaare.

Beweis:

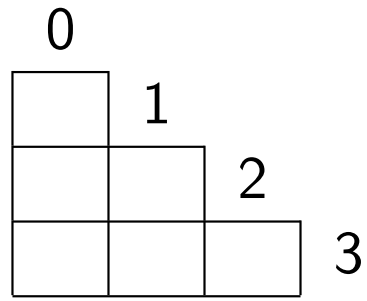
$\{p, q\} \in U \implies p$ und q unterscheidbar: Invariante

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

Induktion über die Länge eines unterscheidenden Worts. □

Implementierung von U:

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



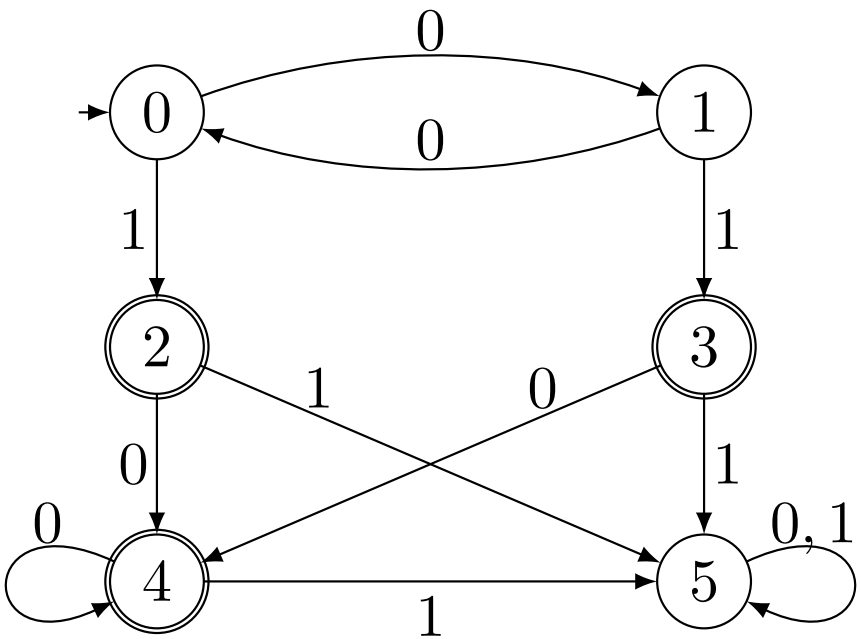
for all $p \in F$, $q \in Q \setminus F$ **do** markiere $\{p, q\}$
while $\exists$ unmarkiertes $\{p, q\}$ $\exists a \in \Sigma$. $\{\delta(p, a), \delta(q, a)\}$ ist markiert
do markiere $\{p, q\}$

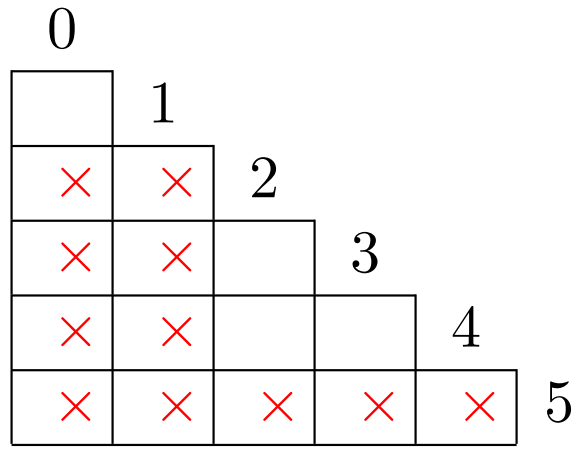
Komplexität:

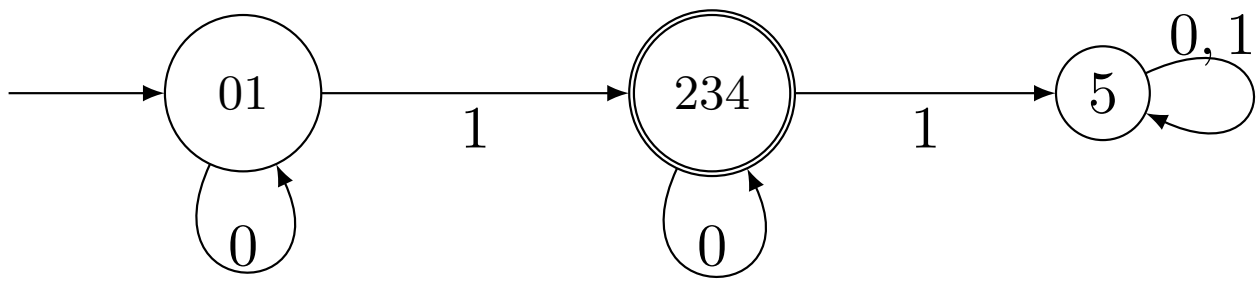
$$O\left(\binom{n}{2} + \binom{n}{2}\binom{n}{2}|\Sigma|\right) = O\left(\frac{n(n-1)}{2} + \frac{n^2(n-1)^2}{4}|\Sigma|\right) = O(n^4)$$

bei fixem Σ.

Beispiel 3.49







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

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

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

for all $\{p, q\} \subseteq Q$ mit $p \neq q$, $a \in \Sigma$ **do**
  $p' := \delta(p, a)$; $q' := \delta(q, a)$
  if $p' \neq q'$ **then** $D[\{p', q'\}] := D[\{p', q'\}] \cup \{\{p, q\}\}$
for all $p' \in F$, $q' \in Q \setminus F$ **do** $mark(\{p', q'\})$

$mark(\{p', q'\}) =$
  if $\{p', q'\}$ unmarkiert **then**
    markiere $\{p', q'\}$
    for all $\{p, q\} \in D[\{p', q'\}]$ **do** $mark(\{p, q\})$

Komplexität: $O(n^2 + n^2) = O(n^2)$.
Korrektheit?

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

Eine weitere Anwendung: Äquivalenztest von DFAs.

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

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.

Formale Definition des “kollabierten Automaten”

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:

$$[a]_\approx = [b]_\approx \quad \Leftrightarrow \quad a \approx b$$

Quotientenmenge:

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

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

Definition 3.50 (Äquivalenz von Zuständen)

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

Intuition: Zwei Zustände sind äquivalent gwd sie die selbe Sprache erkennen.

Wir schreiben $\equiv$ statt $\equiv_M$ wenn M klar ist.

Einfache Fakten:

- $\equiv_M$ ist eine Äquivalenzrelation.
- $p \equiv_M q \implies \delta(p, a) \equiv_M \delta(q, a)$
- Algorithmus U liefert die unterscheidbaren Zustände, also $\not\equiv$.

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

Die „Kollabierung“ von M bzgl. $\equiv$ ist der *Quotientenautomat*:

Definition 3.51 (Quotientenautomat)

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

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

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

Lemma 3.52

$$L(M/{\equiv}) = L(M)$$

Beweis: Übung.

Minimalität des Quotientenautomaten

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

$$\begin{aligned} p \equiv_M q \quad &\Leftrightarrow \quad \forall w \in \Sigma^*.\ \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F \\ &\Leftrightarrow \quad \forall w \in \Sigma^*.\ \hat{\delta}(q_0, uw) \in F \Leftrightarrow \hat{\delta}(q_0, vw) \in F \\ &\Leftrightarrow \quad \forall w \in \Sigma^*.\ uw \in L(M) \Leftrightarrow vw \in L(M) \end{aligned}$$

Definition 3.53

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

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

Obige Beobachtung lässt sich nun schreiben als

$$\hat{\delta}(q_0, u) \equiv_M \hat{\delta}(q_0, v) \quad \Leftrightarrow \quad u \equiv_{L(M)} v$$

Achtung

$p \equiv_M q$ ist eine Relation auf Zuständen von M

$u \equiv_L v$ ist eine Relation auf Wörtern

$$u \equiv_{L(M)} v \quad \Leftrightarrow \quad \hat{\delta}(q_0, u) \equiv_M \hat{\delta}(q_0, v)$$

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

$$[u]_{\equiv_{L(M)}} \quad \mapsto \quad [\hat{\delta}(q_0, u)]_{\equiv_M}$$

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

Satz 3.54

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

Beweis:

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

$$|Q'| \geq |Q'/\equiv_{M'}| = |\Sigma^*/\equiv_L| = |Q/\equiv_M|$$

□

Es gilt sogar:

Fakt 3.55

Alle Quotientenautomaten $M/\equiv_M$ für die gleiche Sprache $L(M)$ 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 3.56

Sei $L := \{w \in \{0,1\}^* \mid w \text{ endet mit } 00\}$.
Die einzigen drei $\equiv_L$ Äquivalenzklassen sind:

$$
\begin{aligned}
[\epsilon]_{\equiv_L} &= \{w \mid w \text{ endet nicht mit } 0\} \\
[0]_{\equiv_L} &= \{w \mid w \text{ endet mit 0, aber nicht mit } 00\} \\
[00]_{\equiv_L} &= \{w \mid w \text{ endet mit } 00\}
\end{aligned}
$$

Definition 3.57 (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 3.58 (Myhill-Nerode)

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

Beweis:

„$\Longrightarrow$“: Ist L regulär, so wird L von einem DFA M akzeptiert.
Daher hat $\equiv_L$ so viele Äquivalenzklassen wie $M/\equiv_M$ Zustände.
„$\Longleftarrow$“: Hat $\equiv_L$ endlich viele Äquivalenzklasse,
so ist M_L ein DFA mit $L(M_L) = L$. □

Wie sieht M_L aus wenn L *nicht* regulär ist?

Beispiel 3.59

$L = \{w \mid w \ \#_0(w) = \#_1(w)\}$ $\quad$ ($\#_a(w) =$ Anzahl der a in w)

$$\cdots \underset{0}{\overset{1}{\rightleftarrows}} [0^2] \underset{0}{\overset{1}{\rightleftarrows}} [0] \underset{0}{\overset{1}{\rightleftarrows}} [\epsilon] \underset{0}{\overset{1}{\rightleftarrows}} [1] \underset{0}{\overset{1}{\rightleftarrows}} [1^2] \underset{0}{\overset{1}{\rightleftarrows}} \cdots$$

Endzustände?

$$[1^i]_{\equiv_L} = \{w \mid \#_1(w) = \#_0(w) + i\}$$

Warum $1^i \not\equiv_L 1^j$ falls $i \neq j$?

Vollständige Methode um Nichtregularität von L zu zeigen:

Gib unendliche Menge $w_1, w_2, \ldots$ 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 $p \not\equiv_M q$ gdw $\exists w.\ \hat{\delta}(p,w) \in F \not\Leftrightarrow \hat{\delta}(q,w) \in F$

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

Knobelaufgabe:
Sei $L = \{ww \mid w \in \Sigma^*\}$.

Man zeige: $[w]_L = \{w\}$

4. Kontextfreie Sprachen

4.1 Kontextfreie Grammatiken

Beispiel 4.1 (Arithmetische Ausdrücke)

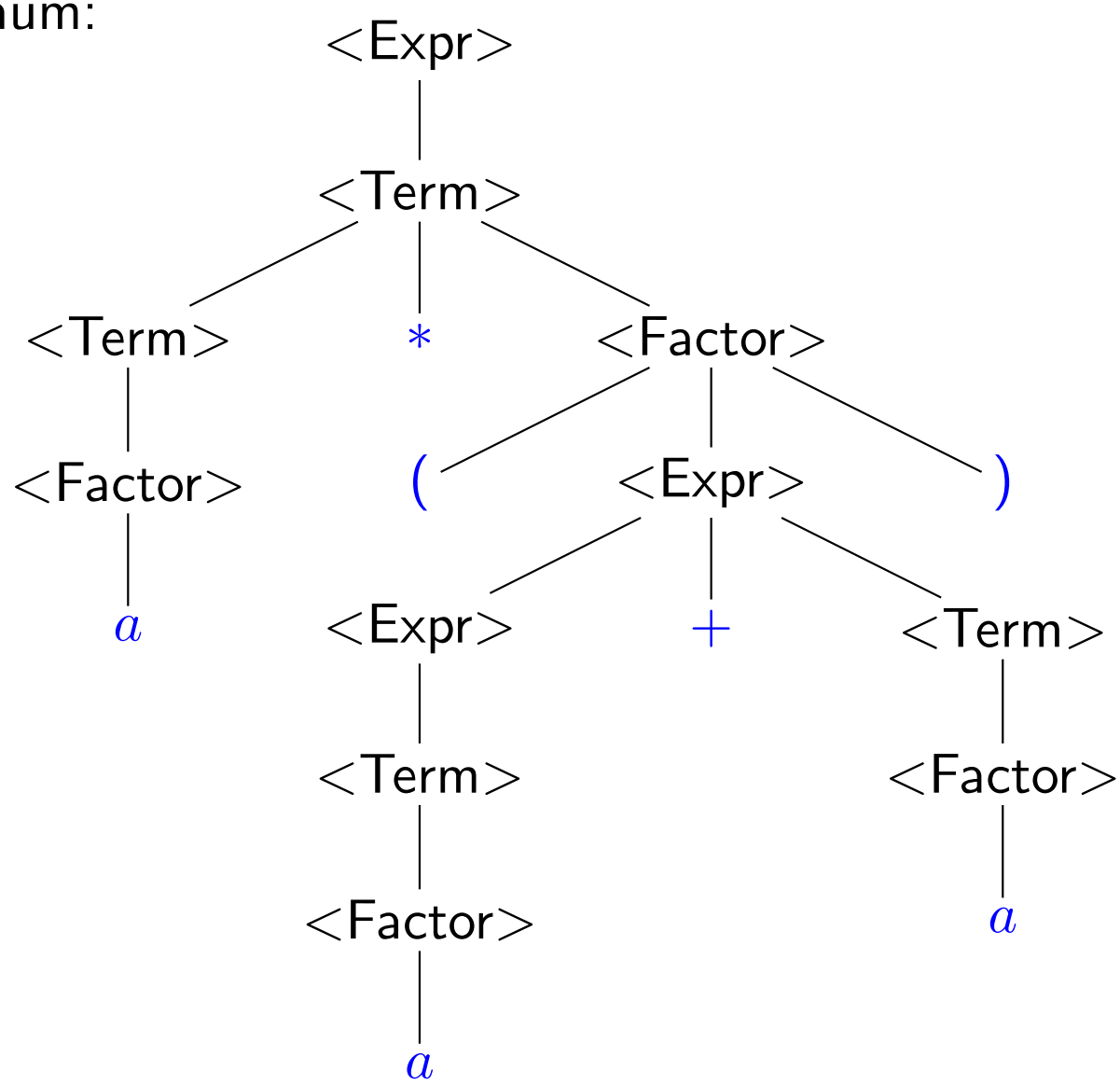
$$
\begin{array}{rcl}
\text{<Expr>} & \to & \text{<Term>} \\
\text{<Expr>} & \to & \text{<Expr>} + \text{<Term>} \\
\text{<Term>} & \to & \text{<Factor>} \\
\text{<Term>} & \to & \text{<Term>} * \text{<Factor>} \\
\text{<Factor>} & \to & a \\
\text{<Factor>} & \to & (\text{<Expr>})
\end{array}
$$

Eine (Links)Ableitung:

<Expr> $\to$

$$\to a * (a + a)$$

Der Syntaxbaum:



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

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. Das Parsen ist:
  - 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.

Definition 4.2

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:

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

$$A \to \alpha_1 \mid \alpha_2 \mid \alpha_3$$

Beispiel 4.3 (Arithmetische Ausdrücke)

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

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

$$P = \left\{ \begin{array}{lcl} E & \rightarrow & T \mid E + T \\ T & \rightarrow & F \mid T * F \\ F & \rightarrow & a \mid (E) \end{array} \right\}$$

$$S = E$$

Definition 4.4

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:

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

Definition 4.5 (Iteration von $\to_G$)

$$\begin{aligned}
\alpha &\to_G^0 \alpha \\
\alpha &\to_G^{n+1} \gamma \quad :\Leftrightarrow \quad \exists \beta.\ \alpha \to_G^n \beta \to_G \gamma \\
\alpha &\to_G^* \beta \quad :\Leftrightarrow \quad \exists n.\ \alpha \to_G^n \beta \\
\alpha &\to_G^+ \beta \quad :\Leftrightarrow \quad \exists n > 0.\ \alpha \to_G^n \beta
\end{aligned}$$

Wir nennen

$$\alpha_1 \to_G \alpha_2 \to_G \cdots \to_G \alpha_n$$

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

Definition 4.6 (Kontextfreie Sprache)

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

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

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 $\alpha \to \beta$ statt $\alpha \to_G \beta$.

Beispiel 4.7

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

$$S \to aSb \mid \epsilon$$

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

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

Der unendliche Baum aller möglichen (Links-)Ableitungen:

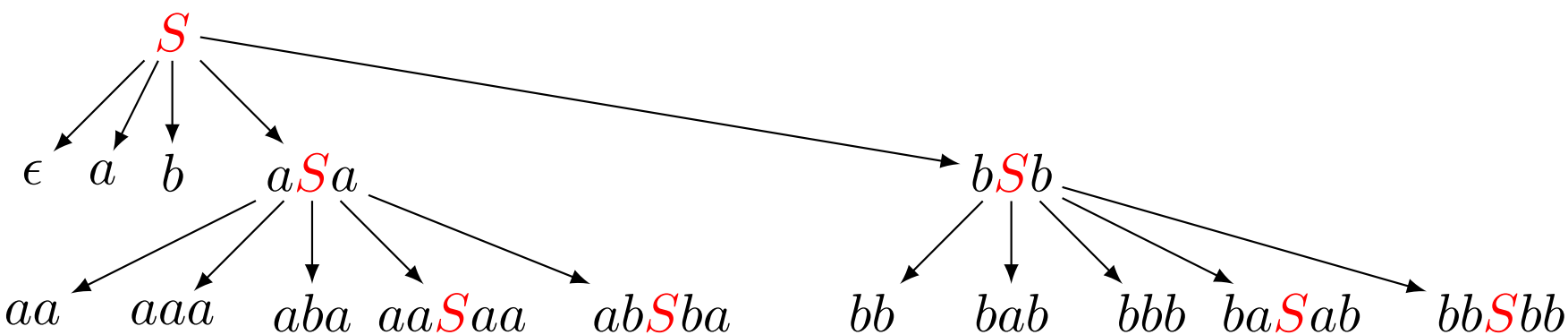
$$\begin{array}{ccccccccc} S & \to & aSb & \to & a^2Sb^2 & \to & a^3Sb^3 & \to & \dots \\ & \searrow & & \searrow & & \searrow & & \searrow & \\ & & \epsilon & & ab & & a^2b^2 & & a^3b^3 \end{array}$$

Beispiel 4.8

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

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

Der Anfang des unendlichen Baums aller Ableitungen (Achtung, dies ist *kein* Syntaxbaum!):



Lemma 4.9 (Dekompositionslemma)

$$\alpha_1\alpha_2 \to_G^n \beta$$
$$\Leftrightarrow$$
$$\exists \beta_1, \beta_2, n_1, n_2.\ \ \beta = \beta_1\beta_2 \ \wedge\ n = n_1 + n_2 \ \wedge\ \alpha_i \to_G^{n_i} \beta_i \ (i = 1, 2)$$

Beweis:

Übung! □

Definition 4.10

Eine CFG heißt rechtslinear gdw jede Produkion von der Form

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

Eine CFG heißt linkslinear gdw jede Produkion von der Form

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

Lemma 4.11

Die rechtslinearen und linkslinearen Grammatiken erzeugen jeweils genau die regulären Sprachen.

Beweis: Übung!

Korollar 4.12

Die regulären Sprachen sind eine echte Teilklasse der kontextfreien Sprachen.

4.2 Induktive Definitionen, Syntaxbäume und Ableitungen

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

Beispiel 4.13

$$S \to \epsilon \mid \texttt{[}S\texttt{]} \mid SS$$

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

$$\begin{array}{rcl} & & \epsilon \in L_G(S) \\ u \in L_G(S) & \Longrightarrow & \texttt{[}u\texttt{]} \in L_G(S) \\ u \in L_G(S) \wedge v \in L_G(S) & \Longrightarrow & uv \in L_G(S) \end{array}$$

Damit gilt zB: $\epsilon \in L(S) \Longrightarrow \texttt{[]} \in L(S) \Longrightarrow \texttt{[[]]} \in L(S)$
$\Longrightarrow \texttt{[[]][]} \in L(S)$

Bemerkungen

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

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:

$$\begin{array}{rcl} & & P(\epsilon) \\ P(u) & \Longrightarrow & P(\texttt{[}u\texttt{]}) \\ P(u) \wedge P(v) & \Longrightarrow & P(uv) \end{array}$$

„Induktion über die Erzeugung von u“

Lemma 4.14

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.

□

Hinweis

Die Aussage

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

ist eine Abkürzung für

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

Der Allquantor wird dann oft weggelassen:

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

Definition 4.15

Präfix:

$$u \preceq w \quad :\Leftrightarrow \quad \exists v.\ uv = w$$

Anzahl der Vorkommen:

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

Für unser Beispiel führen wir zwei Abk. ein:

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

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: `][`, `[]][[]`

Satz 4.16

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

Beweis:

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

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

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

$[u]$: 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 $q \preceq v$:

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

Beweis (Forts.)

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

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 $h(w) := A(w) - B(w)$:

$h(\epsilon)$, $h(a_1)$, $h(a_1a_2)$,. . . , $h(a_1 \dots a_n)$ (alle ≥ 0!).

Insb. gilt $a_1 =$ [und $a_n =$]. Wir betrachten zwei Fälle:

- 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) $A(v) = A(\,[v]\,) - 1 = B(\,[v]\,) - 1 = B(v)$
  (2) $p \preceq v$: $h(\,[p) = A(\,[p) - B(\,[p) > 0 \implies$
  $A(p) = A(\,[p) - 1 > B(\,[p) - 1 = B(p) - 1 \implies$
  $A(p) \geq B(p)$
  $\implies$ $v \in L_G(S)$ (nach IA) $\implies$ $u = [v] \in L_G(S)$

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) $A(u_1) = B(u_1)$ da $h(u_1) = 0$;
  $A(u_2) = A(u) - A(u_1) = B(u) - B(u_1) = B(u_2)$
  (2) $p \preceq u_1 \implies p \preceq u \implies A(p) \geq B(p)$
  $p \preceq u_2$: $A(p) = A(u_1 p) - A(u_1) \geq B(u_1 p) - B(u_1) = B(p)$
  $\implies u_1, u_2 \in L_G(S)$ (nach IA, da $|u_i| < n$)
  $\implies u = u_1 u_2 \in L_G(S)$

□

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.

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

$$A_i \to w_0 A_{i_1} w_1 \ldots w_{n-1} A_{i_n} w_n$$

Man kann G als eine (simultane) induktive Definition von Sprachen $L_G(A_i)$ für all $A_i \in V$ sehen. Jede Produktion korrespondiert zu einer Erzeugungsregel

$$u_1 \in L_G(A_{i_1}) \wedge \cdots \wedge u_n \in L_G(A_{i_n}) \implies w_0 u_1 w_1 \ldots u_n w_n \in L_G(A_i)$$

Aussagen der Form

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

werden durch simultane Induktion über die Erzeugung von u bewiesen, indem man für jede Produktion zeigt:

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

Beispiel 4.17

Grammatik:

$$A \to \epsilon \mid aB \qquad B \to Aa$$

Induktive Definition:

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

Beim induktiven Beweis von

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

muss man zeigen

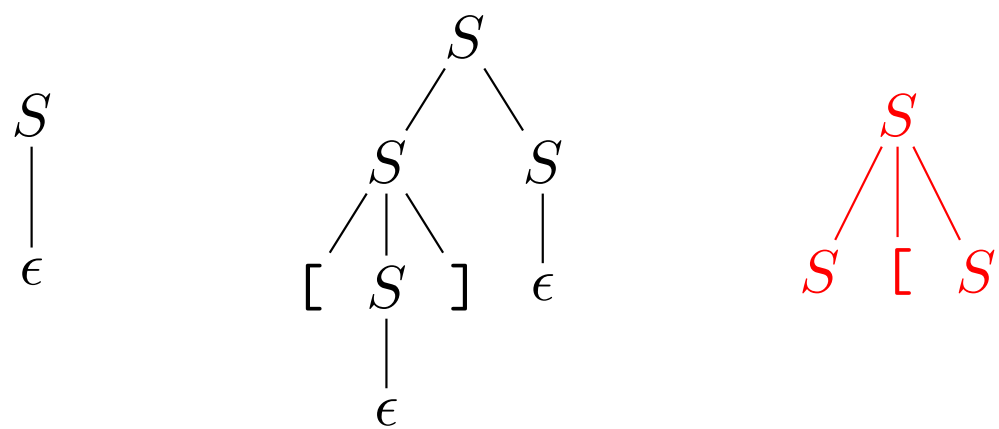
- $\#_a(\epsilon)$ ist gerade
- $\#_a(w)$ ist ungerade $\implies$ $\#_a(aw)$ ist gerade
- $\#_a(w)$ ist gerade $\implies$ $\#_a(wa)$ ist ungerade

Definition 4.18 (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 $X_1, \ldots, X_n \in V \cup \Sigma \cup \{\epsilon\}$ beschriftet sind, dann ist $A \to X_1 \ldots X_n$ eine Produktion in P,
- ein Blatt ϵ der einzige Nachfolger seines Vorgängers ist.

Beispiel 4.19 (Syntaxbäume für $S \to \epsilon \mid [S] \mid SS$)



Satz 4.20

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

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

Beweis:

Skizze (Details: [HMU])

$1 \Rightarrow 2$: Sei $A \to_G^n w$. Wir beweisen $w \in L_G(A)$ mit Ind. über n.

Die Ableitung muss von folgender Form sein:

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

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

$2 \Rightarrow 3$: 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 (sogar Links)Ableitung transformieren. Induktion über die Höhe des Baums: Transformiere die Unterbäume t_i mit Beschriftung A_i in Ableitungen $A_i \to_G^* u_i$ und füge diese mit dem Dekompositionslemma zu einer grossen Ableitung $A \to_G w_0 A_1 w_1 \ldots A_m w_m \to_G^* w_0 u_1 w_1 \ldots u_m w_m$ zusammen.

Definition 4.21

- 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 4.22

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

Satz 4.23

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

4.3 Die Chomsky-Normalform

Definition 4.24

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

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

haben.

Wir konstruieren und beweisen nun schrittweise:

Satz 4.25

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 $S \to \epsilon$ hinzu.

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

Lemma 4.26

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 eine Obermenge $\hat{P}$:

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

Offensichtlich gilt $L(\hat{G}) = L(G)$: Jede neue Produktion kann von 2 alten simuliert werden.

Wir definieren G' als $\hat{G}$ ohne die ϵ-Produktionen.
Denn diese sind nun überflüssig:

Beweis (Forts.):

Sei $S \to^*_{\hat{G}} w$, $w \neq \epsilon$, eine Ableitung minimaler Länge.
Käme darin $B \to \epsilon$ vor, also

$$S \to^*_{\hat{G}} \gamma B \delta \to_{\hat{G}} \gamma\delta \to^*_{\hat{G}} w$$

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

$$S \to^k_{\hat{G}} \alpha' A \beta' \to_{\hat{G}} \alpha'\alpha B \beta\beta' \to^m_{\hat{G}} \gamma B \delta \to_{\hat{G}} \gamma\delta \to^n_{\hat{G}} w$$

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

$$S \to^k_{\hat{G}} \alpha' A \beta' \to_{\hat{G}} \alpha'\alpha\beta\beta' \to^m_{\hat{G}} \gamma\delta \to^n_{\hat{G}} w$$

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

Beispiel 4.27

$$S \to AB, \quad A \to aAA \mid B, \qquad B \to bBS \mid \epsilon$$

Wir nennen $A \to B$ eine Kettenproduktion.

Lemma 4.28

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 eine Obermenge $\hat{P}$:

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

Das Ergebnis G' ist $\hat{G}$ ohne die (nun überflüssigen) Kettenproduktionen.
Rest des Beweises analog zur Elimination von ϵ-Produktionen. □

Beispiel 4.29

$$A \to a \mid B, \qquad B \to b \mid C \mid aBB, \qquad C \to A \mid AB$$

Konstruktion einer Chomsky-Normalform

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

1. 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 $A_a \to a$ zu P hinzu.
2. Ersetze jede Produktion der Form

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

   durch

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

   wobei $C_2, \ldots, C_{k-1}$ neue Nichtterminale sind.
3. Eliminiere alle ϵ-Produktionen.
4. Eliminiere alle Kettenproduktionen.

Aufgabe:

- Zeige: Mit diesem Algorithmus ist das Ergebnis höchstens quadratisch größer als G.
- Zeige: die Reihenfolge der Schritte ist wichtig!
  - Finde eine andere Reihenfolge, die inkorrekt ist: Das resultierende Verfahren ergibt nicht immer eine Grammatik in CNF.
  - Finde eine andere Reihenfolge, die zwar zu einem korrekten Verfahren führt, aber eine Grammatik in CNF produzieren kann, die exponentiell größer als G ist.

Definition 4.30

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

$$A \to aA_1 \dots A_n$$

ist.

Satz 4.31

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

4.4 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 4.32 (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*
- $\forall i \in \mathbb{N}.\ uv^i wx^i y \in L$.

Beweis:

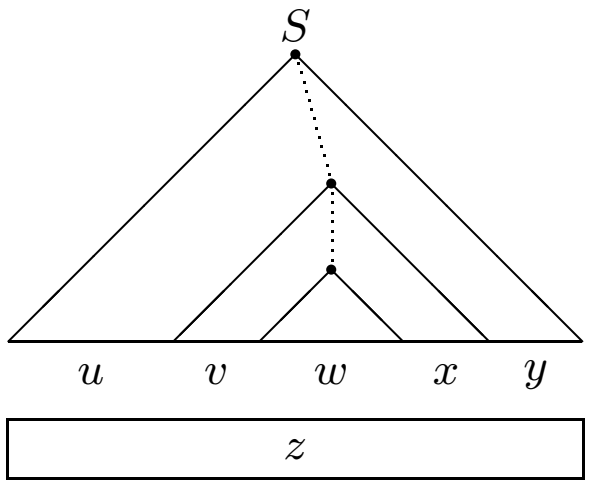
Sei $G = (V, \Sigma, P, S)$ eine CFG in Chomsky-Normalform mit $L(G) = L \setminus \{\epsilon\}$. Wähle $n = 2^{|V|}$. Sei $z \in L(G)$ mit $|z| \geq n$.

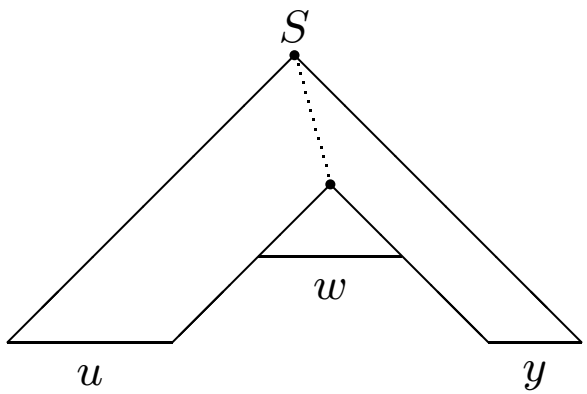
Jeder Binärbaum mit $\geq 2^k$ Blättern hat die Höhe $\geq k$

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.

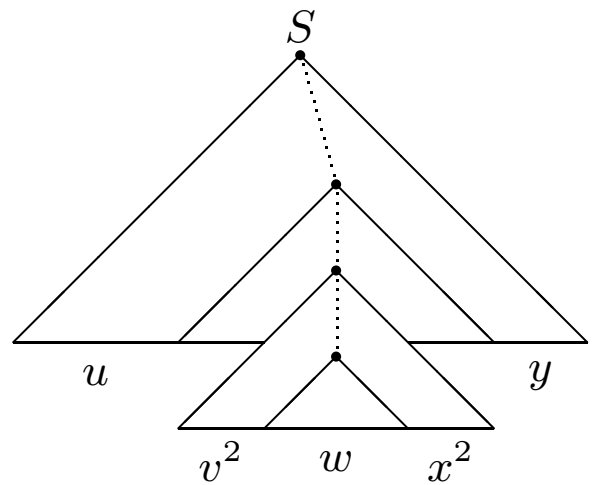
Auf diesem Pfad der Länge $\geq |V|$ kommen $\geq |V| + 1$ Nichtterminale vor, also mindestens eins doppelt. Nennen wir es A.

Die zwischen zwei Vorkommen von A liegende Teilableitung kann nun beliebig oft wiederholt werden.





Dieser Ableitungsbaum zeigt
$uwy \in L$



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

Beweis (Forts.):

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

Um $|vwx| \leq n$ zu erreichen, darf das obere A maximal $|V| + 1$ Schritte von jedem Blatt entfernt sein, weil $n = 2^{|V|}$. 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 4.33

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 enhält 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 $uv^2wx^2y \notin L$. ↯

Beispiel 4.34

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.

4.5 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 $S \to_G^* w \in \Sigma^*$ gibt, in der X vorkommt.
- **erzeugend** gdw es eine Ableitung $X \to_G^* w \in \Sigma^*$ gibt.
- **erreichbar** gdw es eine Ableitung $S \to_G^* \alpha X \beta$ gibt.

Fakt 4.35

Nützliche Symbole sind erzeugend und erreichbar.
Aber nicht notwendigerweise umgekehrt:

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

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

Beispiel 4.36

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

1. Elimination nicht erzeugender Symbole:

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

2. Elimination unerreichbarer Symbole:

$$S \to a$$

Umgekehrte Reihenfolge:

1. Elimination unerreichbarer Symbole:

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

2. Elimination nicht erzeugender Symbole:

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

Satz 4.37

Eliminiert man aus einer Grammatik G

1. *alle nicht erzeugenden Symbole, mit Resultat G_1, und*
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 $S \to_G^* w$.
Jedes Symbol in dieser Ableitung ist erreichbar und erzeugend.
Also gilt auch $S \to_{G_2}^* w$, dh $w \in L(G_2)$.

Beweis (Forts.): [G_1 : erzeugend in G, G_2 : erreichbar in G_1]

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

$$S \quad \to^*_{G_2} \quad \ldots X \ldots \quad \to^*_{G_2} \quad \ldots \in \Sigma_2^*$$

X muss in G_1 erreichbar sein, dh $S \to^*_{G_1} \alpha X \beta$.
Da alle Symbole in der Ableitung erreichbar sind: $S \to^*_{G_2} \alpha X \beta$.

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

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

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

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

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

Die Ableitung $\alpha X \beta \to^*_{G_1} w$ enthält nur Symbole, die in G_1 erreichbar sind. Daher auch $\alpha X \beta \to^*_{G_2} w$.

$$S \ \to^*_{G_2} \ \alpha X \beta \ \to^*_{G_2} \ w$$

□

Satz 4.38

Die Menge der erzeugenden Symbole einer CFG ist berechenbar.

Beweis:

Wir berechnen die erzeugenden Symbole induktiv:

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

□

Beispiel 4.39

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

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

Korollar 4.40

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

Beweis:

□

Satz 4.41

Die Menge der erreichbaren Symbole einer CFG ist berechenbar.

Beweis:

Wir berechnen die erreichbaren Symbole induktiv:

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

$\square$

Satz 4.42

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 4.26).
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 $(B \to \beta) \in P$ und $|\alpha\beta\gamma| \leq |w|$, dann auch $\alpha\beta\gamma \in R$.

Es gilt:

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

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

4.6 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 4.43

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

Damit gilt:

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

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

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

Korrektheitsbeweis: Induktion über $j - i$.

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

Beispiel 4.44

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

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

Satz 4.45

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.

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

Für jede dieser Mengen werden

- $j - i < n$ Werte für k betrachtet,
- für jedes k wird für alle Produktionen $A \to BC$ 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 $O(\ )$.] □

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} die Syntaxbäume, 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?

4.7 Abschlusseigenschaften

Satz 4.46

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.

Beweis:

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

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

□

$\Longrightarrow$ Verallgemeinerte kontextfreie Grammatiken mit Produktionen der Gestalt $X \to r$, wobei r ein regulärer Ausdruck über $(V \cup T)$ ist, erzeugen kontextfreie Sprachen.

Satz 4.47

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

Beweis:

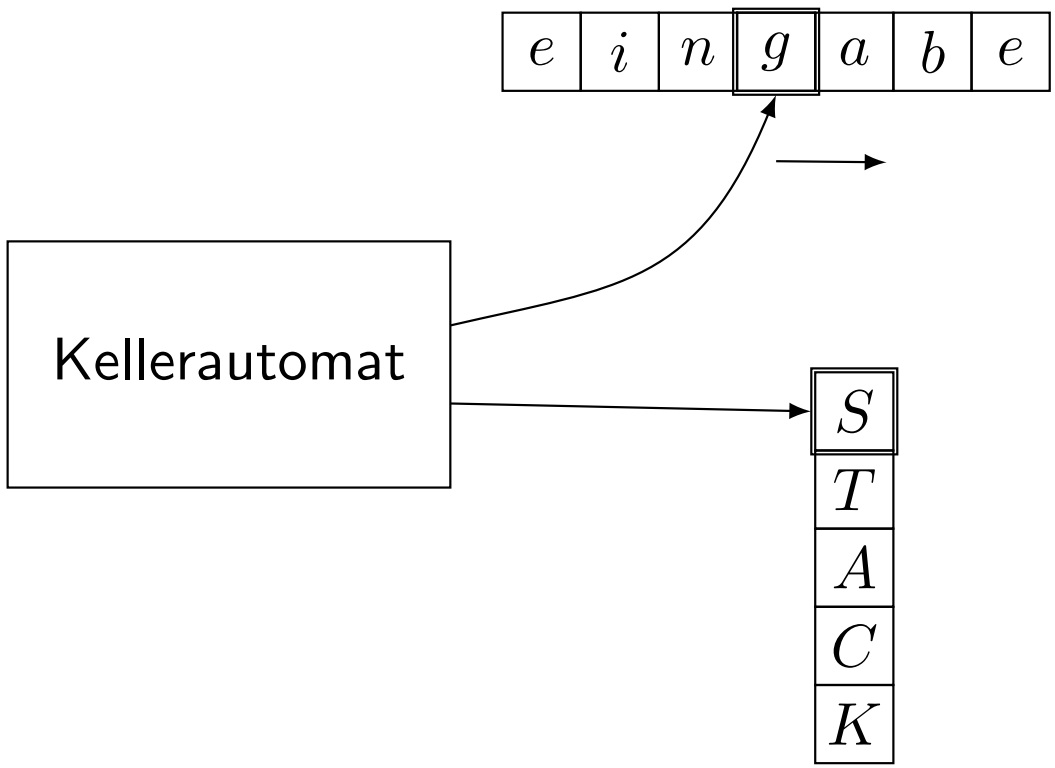
$$L_1 := \{a^i b^i c^j \mid i, j \in \mathbb{N}\} \text{ ist kontextfrei}$$

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

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

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

4.8 Kellerautomaten



Anwendungsgebiete von Kellerautomaten:

- Syntaxanalyse von Programmiersprachen
- Analyse von Programmen mit Rekursion

Definition 4.48

Ein (nichtdeterministischer) Kellerautomat (PDA = Pushdown Automaton) $M = (Q, \Sigma, \Gamma, q_0, Z_0, \delta, F)$ besteht aus

- Q endliche Menge von Zuständen
- Σ endliches Eingabealphabet
- Γ endliches Kelleralphabet

$q_0 \in Q$ Anfangszustand

$Z_0 \in \Gamma$ initialer Kellerinhalt

- δ Übergangsfunktion $\delta\colon Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to \mathcal{P}_e(Q \times \Gamma^*)$
  ($\mathcal{P}_e$ = Menge aller endlichen Teilmengen)

$F \subseteq Q$ Endzustände

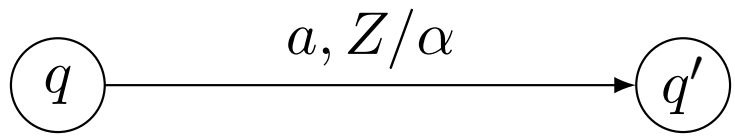
Intuitive Bedeutung von $(q', \alpha) \in \delta(q, a, Z)$:

> *Wenn sich M im Zustand q befindet, das Eingabezeichen a liest, und Z das oberste Kellerzeichen ist,*
> *so kann M im nächsten Schritt in q' übergehen und Z durch α ersetzen.*

Achtung: $\delta(q, a, Z) \ni (q', \alpha)$

- α kann die Länge, 0, 1, und mehr haben
- Falls $a = \epsilon$: kein Eingabezeichen wird gelesen

Eine graphische Notation für $\delta(q, a, Z) \ni (q', \alpha)$:



Kein endlicher Automat!

Definition 4.49

Eine Konfiguration eines Kellerautomaten M ist ein Tripel (q, w, α) mit $q \in Q$, $w \in \Sigma^*$ und $\alpha \in \Gamma^*$.
Die Anfangskonfiguration von M für die Eingabe $w \in \Sigma^*$ ist (q_0, w, Z_0).

Intuitiv stellt eine Konfiguration (q, w, α) eine "Momentaufnahme" des Kellerautomaten dar:

- Der momentane Zustand ist q.
- Der noch zu lesende Teil der Eingabe ist w.
- Der aktuelle Kellerinhalt ist α
  (das oberste Kellerzeichen ganz links stehend).

Definition 4.50

Die Transitionsrelation $\rightarrow_M$ zwischen Konfigurationen:

$$(q, aw, Z\alpha) \rightarrow_M (q', w, \beta\alpha) \quad \text{falls } (q', \beta) \in \delta(q, a, Z)$$

$$(q, w, Z\alpha) \rightarrow_M (q', w, \beta\alpha) \quad \text{falls } (q', \beta) \in \delta(q, \epsilon, Z)$$

Intuitive Bedeutung von $(q, w, \alpha) \rightarrow_M (q', w', \alpha')$:

> *Wenn M sich in der Konfiguration (q, w, α) befindet, dann kann er in einen Schritt in die Nachfolgerkonfiguration (q', w', α') übergehen.*

Achtung: eine Konfiguration kann mehrere Nachfolger haben (Nichtdeterminismus!)

Notation $\rightarrow_M^*$, $\rightarrow_M^n$, etc wie üblich

Definition 4.51

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

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

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

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

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

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

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

Beispiel 4.52

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

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

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

sowohl mit Endzustand als auch mit leerem Keller akzeptiert.

Bsp: $(p,110011,Z_0) \to_M^* (r,\epsilon,\epsilon)$.

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

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

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 PDA kann es wegen ϵ-Übergängen auch unendliche Berechnungen $\to_M$ geben, zB $\delta(q, \epsilon, Z) = (q, ZZ)$:

$$(q, w, Z) \to_M (q, w, ZZ) \to_M (q, w, ZZZ) \to_M \ldots$$

  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.

Ziel:

Akzeptanz durch Endzustände und leeren Keller gleich mächtig.

Satz 4.53 (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:

$$(q_0, w, Z_0) \rightarrow_M^* (f, \epsilon, \gamma) \quad \Leftrightarrow \quad (q_0', w, Z_0') \rightarrow_{M'}^* (q, \epsilon, \epsilon)$$

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':

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

□

Satz 4.54 (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.

$$
\begin{array}{lcl}
Q' & := & Q \uplus \{q_0', f\} \\
\Gamma' & := & \Gamma \uplus \{Z_0'\} \\
F & := & \{f\}
\end{array}
$$

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

□

Die folgenden Sätze erleichtern Beweise über Berechnungen von PDAs.

Lemma 4.55 (Erweiterungslemma)

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

Beweis:

Nach Definition von $\to_M$ gilt

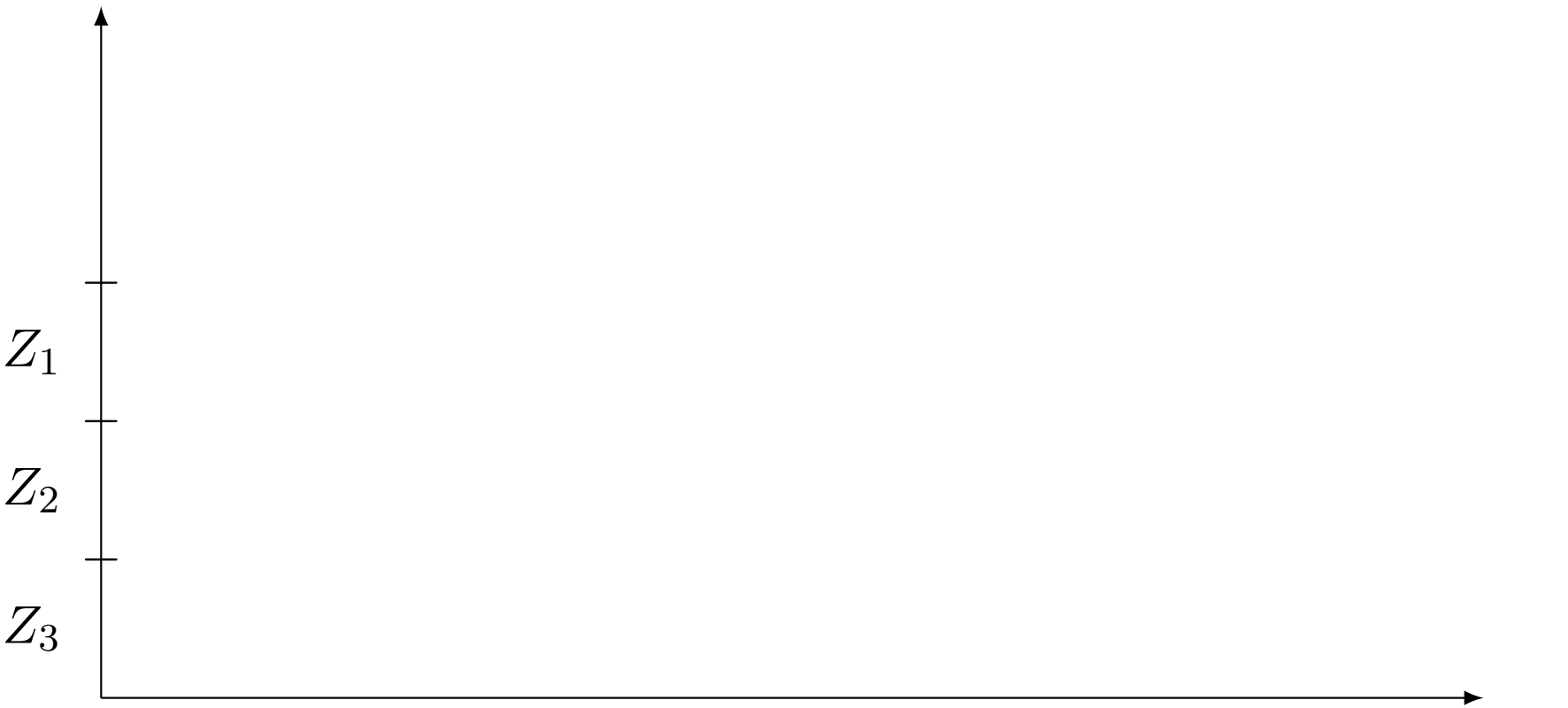
$$(q_i, u_i, \alpha_i) \to_M (q_{i+1}, u_{i+1}, \alpha_{i+1}) \implies$$
$$(q_i, u_i v, \alpha_i \beta) \to_M (q_{i+1}, u_{i+1} v, \alpha_{i+1} \beta)$$

Mit Induktion über n folgt die Behauptung. □

Gilt die Umkehrung, zB

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

Keller



Schritte

Satz 4.56 (Zerlegungssatz)

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

$$(p_{i-1}, u_i, Z_i) \to_M^{n_i} (p_i, \epsilon, \epsilon) \qquad (i = 1, \ldots, k)$$

und $w = u_1 \ldots u_k$, $p_0 = q$, $p_k = q'$, $\sum n_i = n$.

Beweis: Mit Induktion über n. Basis trivial.

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

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

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

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

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

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

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

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

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

4.9 Äquivalenz von PDAs und CFGs

Satz 4.57 (CFG $\rightarrow$ PDA)

Zu jeder CFG G kann man einen PDA M konstruieren, der mit leerem Keller 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 Produkion $A_a \rightarrow a$ hinzu.

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

$$A \to bB_1 \dots B_k$$

Der PDA wird wie folgt definiert:

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

wobei

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

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

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

Lemma 4.58

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

$$A \to_G^n u\gamma \text{ mit Linksableitung gdw } (q, uv, A) \to_M^n (q, v, \gamma)$$

Beweis:

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

$$\begin{aligned}
& A \to_G^{n+1} u\gamma \\
\Leftrightarrow\ & \\
& \exists (B \to b\beta) \in P, w, \alpha.\ A \to_G^n wB\alpha \to_G wb\beta\alpha = u\gamma \\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{dh } wb = u, \beta\alpha = \gamma) \\
\Leftrightarrow\ & \\
& (q, wbv, A) \to_M^n (q, bv, B\alpha) \wedge (q, bv, B\alpha) \to_M (q, v, \beta\alpha) \\
\Leftrightarrow\ & \\
& (q, uv, A) \to_M^{n+1} (q, v, \gamma)
\end{aligned}$$

□

$A \to_G^n u\gamma$ mit Linksableitung gdw $(q, uv, A) \to_M^n (q, v, \gamma)$

Satz 4.59
$L(G) = L_\epsilon(M)$

Beweis:

$u \in L(G)$

$\Leftrightarrow S \to_G^* u$ mit Linksableitung

$\Leftrightarrow (q, u, S) \to_M^* (q, \epsilon, \epsilon)$

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

Satz 4.60 (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: $[q, Z, r_k] \rightarrow_G^* w$ gdw $(q, w, Z) \rightarrow_M^* (r_k, \epsilon, \epsilon)$

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!)

Lemma 4.61

$[q, Z, p] \to_G^n w$ *gdw* $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$

Beweis:

($\Rightarrow$): Wenn $[q, Z, p] \to_G^n w$ dann $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$.
Mit Induktion über n.
IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n.
Die Ableitung $[q, Z, p] \to_G^n w$ muss von folgender Form sein:

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

mit $r_k = p$, $(r_0, Z_1 \dots Z_k) \in \delta(q, b, Z)$,
$[r_{i-1}, Z_i, r_i] \to_G^{n_i} u_i$ $(i = 1, \dots, k)$ und $\sum n_i = n - 1$.
IA $\Rightarrow (r_{i-1}, u_i, Z_i) \to_M^{n_i} (r_i, \epsilon, \epsilon)$
Erweiterungslemma
$\Rightarrow (r_{i-1}, u_i \dots u_k, Z_i \dots Z_k) \to_M^{n_i} (r_i, u_{i+1} \dots u_k, Z_{i+1} \dots Z_k)$
$\Rightarrow (q, w, Z) \to_M (r_0, u_1 \dots u_k, Z_1 \dots Z_k) \to_M^{n-1} (r_k, \epsilon, \epsilon)$

Beweis (Forts.):

($\Leftarrow$): Wenn $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$ dann $[q, Z, p] \to_G^n w$.
Mit Induktion über n.
IA: Beh. gilt für alle Werte $< n$. Zz: Beh. gilt für n.
Die Berechnung $(q, w, Z) \to_M^n (p, \epsilon, \epsilon)$ muss von dieser Form sein:

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

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

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

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

$$\begin{array}{lcll} \text{IA} & \Rightarrow & [r_{i-1}, Z_i, r_i] \to_G^{n_i} u_i & \\ & \Rightarrow & [q, Z, p] & \to_G \quad b[r_0, Z_1, r_1] \dots [r_{k-1}, Z_k, r_k] \\ & & & \to_G^{n-1} \quad bu_1 \dots u_k = w \end{array}$$

□

Damit haben wir bewiesen:

Satz 4.62

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

4.10 Deterministische Kellerautomaten

Definition 4.63

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 4.64

Der folgende DPDA mit $\Sigma = \{0, 1, \$\}$, $\Gamma = \{Z_0, 0, 1\}$ akzeptiert die Sprache $\{w\$w^R \mid w \in \{0, 1\}^*\}$.

Definition 4.65

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 4.66

Jede reguläre Sprache ist eine DCFL.

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

Beispiel 4.67

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

Lemma 4.68

$\exists$*DPDA* M. $L = L_\epsilon(M) \quad \Leftrightarrow$
$\exists$*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 $L_\epsilon(M) = L(a^*)$.
Im Folgenden: Akzeptanz durch Endzustand.

Satz 4.69

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 4.70

Die DCFLs sind eine echte Teilklasse der CFLs.

Hier ist eine konkrete Sprache in CFL\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):

$$S \to AB \mid BA \mid A \mid B$$

$$A \to CAC \mid a \qquad B \to CBC \mid b \qquad C \to a \mid b$$

Wäre L eine DCFL, dann auch $\overline{L} = L_{ww}$ (wegen Satz 4.69).
Aber mit dem Pumping Lemma kann man zeigen, dass L_{ww} keine CFL ist, und daher erst recht keine DCFL.

Lemma 4.71

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. □

Lemma 4.72

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

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

Satz 4.73

Das Wortproblem für DCFLs ist in linearer Zeit lösbar.

Mehr Information: Vorlesungen zum Übersetzerbau

4.11 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

Teil 2
Berechenbarkeit, Entscheidbarkeit, Komplexität

5. Berechenbarkeit, Entscheidbarkeit

Überblick:

- Was kann man berechnen?
  - Welche Funktionen kann man in endlicher Zeit berechnen?
  - Welche Eigenschaften von Objekten können in endlicher Zeit entschieden werden?
- Mit welchen Sprachen/Maschinen?
- Was kann man in polynomieller Zeit berechnen?

5.1 Der Begriff der Berechenbarkeit

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

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

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

Terminologie: Eine Funktion $f : A \to B$ 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 5.1

Jeder Algorithmus berechnet eine partielle Funktion.

Der Algorithmus

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

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

Beispiel 5.2

Ist die Funktion

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

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

Ja, denn es Algorithmen gibt, die π iterativ auf beliebig viele Dezimalstellen genau berechnet werden.

Beispiel 5.3

Ist die Funktion

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

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.

Beispiel 5.4

Ist die Funktion

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

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.

Dieser Beweis ist nicht-konstruktiv.

Satz 5.5

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

Beweis:

Es gibt nur abzählbar viele Algorithmen, aber überabzählbar viele Funktionen in $\mathbb{N} \to \{0, 1\}$. □

Erinnerung: Eine Menge M ist abzählbar falls es eine injektive Funktion $M \to \mathbb{N}$ gibt.

Äquivalente Definitionen:

- Entweder gibt es eine Bijektion $M \to \{0, \ldots, n\}$ für ein $n \in \mathbb{N}$, oder eine Bijektion $M \to \mathbb{N}$.
- Es gibt eine Nummerierung der Elemente von M.

Eine Menge ist überabzählbar wenn sie nicht abzählbar ist.

Lemma 5.6

Σ^* ist abzählbar, falls Σ endlich ist.

Beweis:

Durch Beispiel:

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

□

Annahme:

Ein Algorithmus ist ein Wort über einem endlichen Alphabet.

Gilt für alle bekannten Programmiersprachen.

$\Longrightarrow$ Die Menge der Algorithmen ist abzählbar.

Satz 5.7

Die Menge aller Funktionen $\mathbb{N} \to \{0, 1\}$ *ist überabzählbar.*

Beweis:

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

	0	1	2	3	$\ldots$
f_0	0	1	1	0	$\ldots$
f_1	1	0	0	0	$\ldots$
f_2	0	0	1	0	$\ldots$
f_3	0	0	1	0	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Eine Funktion f, die nicht in der Tabelle ist: ¬Diagonale!

f	1	1	0	1	$\ldots$

Widerspruch!

Formal: Sei $f(i) := 1 - f_i(i)$. Dann $f \neq f_i$ für alle i, da $f(i) \neq f_i(i)$. □

Satz 5.8

Wenn Algorithmen als endliche Wörter kodiert werden können, dann gibt es nicht-berechenbare Funktionen $\mathbb{N} \to \{0,1\}$.

Beweis:

Sei $\mathcal{F}$ die Menge aller Funktionen $\mathbb{N} \to \{0,1\}$.
Sei $\mathcal{A}$ die Menge der Wörter, die Algorithmen kodieren.
Beweis durch Widerspruch.
Annahme: Alle Funktionen von $\mathcal{F}$ sind berechenbar.
Da $\mathcal{A}$ abzählbar ist (Lemma 5.6), gibt es eine injektive Funktion $anum\colon \mathcal{A} \to \mathbb{N}$.
Definiere $fnum\colon \mathcal{F} \to \mathbb{N}$ durch

$$fnum(f) = \min\{anum(a) \mid a \text{ berechnet } f\ \}$$

$fnum$ is injektiv: Wenn $fnum(f_1) = fnum(f_2)$ dann gibt es einen Algorithmus der sowohl f_1 wie auch f_2 berechnet und so $f_1 = f_2$.
Aber dann ist $\mathcal{F}$ abzählbar. Widerspruch zu Satz 5.7 □

Verschiedene Formalisierungen des Begriffs der Berechenbarkeit:

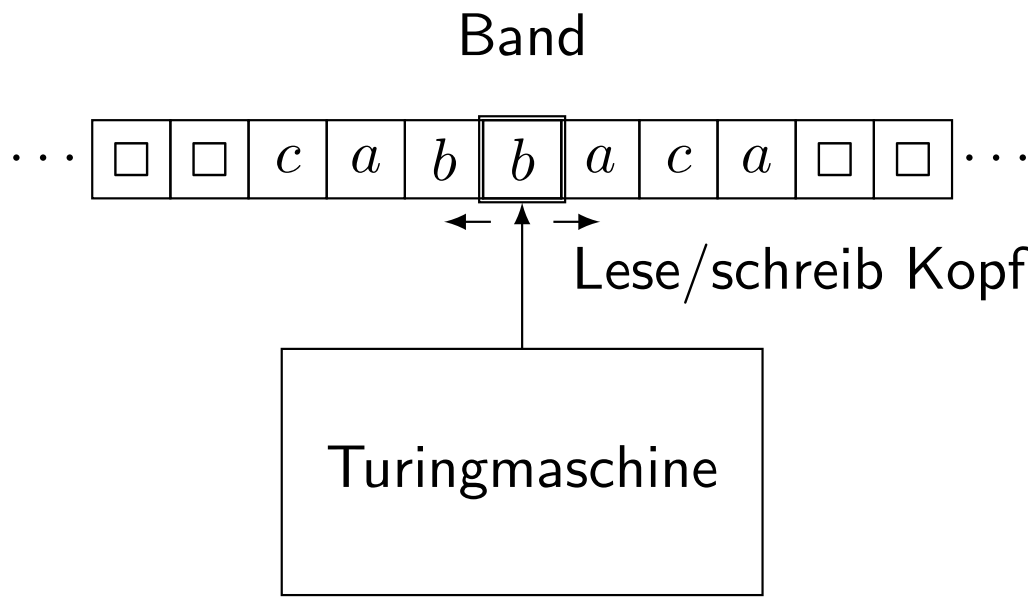
- Turing-Maschinen (Turing 1936)
- λ-Kalkül (Church 1936)
- μ-rekursive Funktionen
- Markov-Algorithmen
- Registermaschinen
- Awk, Basic, C, Dylan, Eiffel, Fortran, Java, Lisp, Modula, Oberon, Pascal, Python, Ruby, Simula, TEX, ...-Programme
- while-Programme
- goto-Programme
- DNA-Algorithmen
- Quantenalgorithmen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

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

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

Die Church-Turing-These ist keine formale Aussage und so nicht beweisbar. Sie wird jedoch allgemein akzeptiert.

5.2 Turingmaschinen



Definition 5.9

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 $\Sigma \subset \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ ist die Übergangsfunktion. δ darf partiell 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.

Annahme: $\delta(q, a)$ is nicht definiert für alle $q \in F$ und $a \in \Gamma$.

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

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 $D = N$).

Definition 5.10

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

Dies modelliert

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

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

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

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

wobei

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

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

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

Beispiel 5.11 (Unär +1)

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

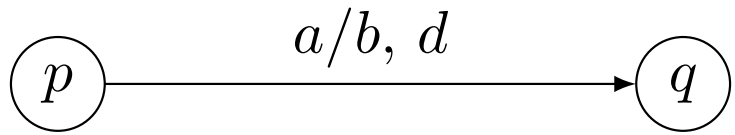
$$\begin{aligned} \delta(q, \Box) &= (f, 1, N) \\ \delta(q, 1) &= (q, 1, R) \end{aligned}$$

- Beispiellauf:

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

- Welche Eingaben führen von q nach f?

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



Beispiel 5.12 (Binär +1)

ZB $1011 \mapsto 1100$

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

wobei

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

Beispiellauf:

$(\epsilon, q_0, 101) \to_M (1, q_0, 01) \to_M (10, q_0, 1) \to_M (101, q_0, \epsilon) \to_M$
$(10, q_1, 1\Box) \to_M (1, q_1, 00\Box) \to_M$
$(\epsilon, q_2, 110\Box) \to_M (\epsilon, q_2, \Box 110\Box) \to_M$
$(\Box, q_f, 110\Box)$

Definition 5.13

Eine Turingmaschine M akzeptiert die Sprache

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

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

$$f(u) = v \quad \Leftrightarrow \quad \exists r \in F.\ (\epsilon, q_0, u) \rightarrow_M^* (\Box \ldots \Box, r, v\Box \ldots \Box)$$

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

$$f(n_1, \ldots, n_k) = m \quad \Leftrightarrow$$
$$\exists r \in F.\ (\epsilon, q_0, bin(n_1)\#bin(n_2)\# \ldots \#bin(n_k)) \rightarrow_M^* (\Box \ldots \Box, r, bin(m)\Box \ldots \Box)$$

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

Zum Halten/Terminieren von TM

Eine TM hält wenn sie eine Konfiguration $(\alpha, q, a\beta)$ erreicht und $\delta(q, a)$ nicht definiert oder (bei einer nichtdetermistischen TM) $\delta(q, a) = \emptyset$.

Nach Annahme hält eine TM immer, wenn sie einen Endzustand erreicht. Damit ist die von einer TM berechnete Funktion wohldefiniert.

Eine TM kann auch halten, bevor sie einen Endzustand erreicht.

Satz 5.14

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. □

Satz 5.15

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

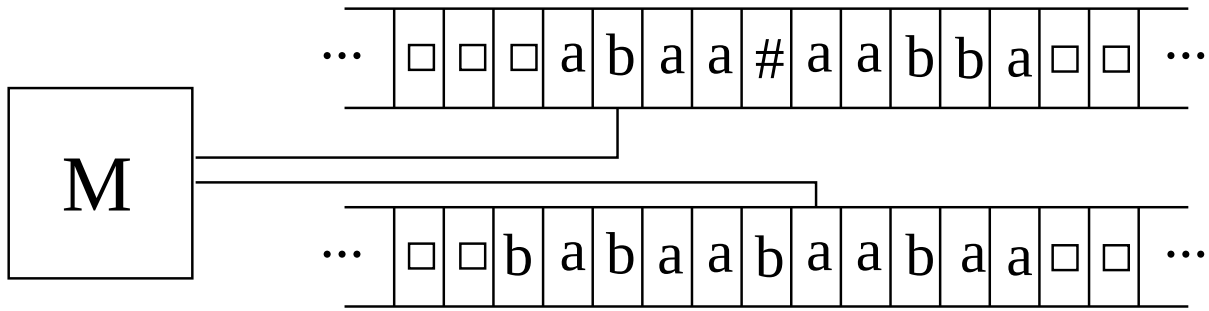
Beweis:

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

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

„$\Longleftarrow$“: 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. □

Eine beliebte Modellvariante ist die k-Band-Turingmaschine:



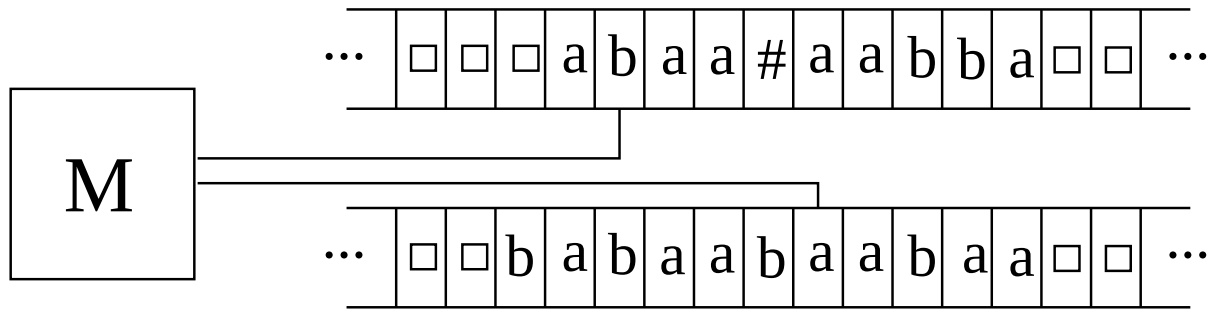
Die k Köpfe sind völlig unabhängig voneinander:

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, N\}^k$$

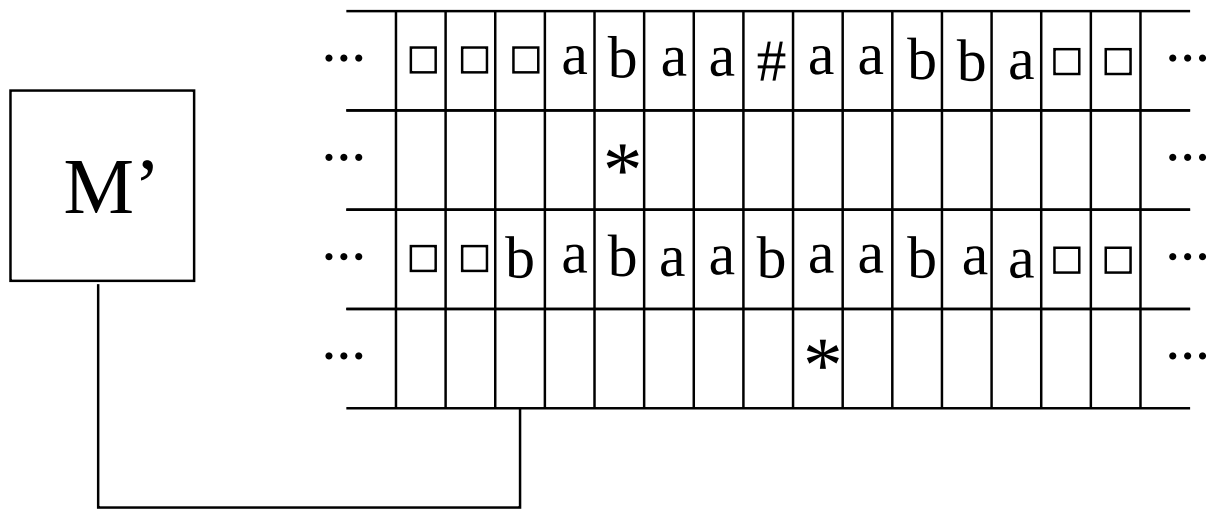
Satz 5.16

Jede k-Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Beweisidee: Aus



wird



Beweisskizze:

- $\Gamma' := (\Gamma \times \{\star, \square\})^k$
- M' simuliert einen M-Schritt durch mehrere Schritte: M'
  - 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.

5.3 Programmieren mit Turingmaschinen

Die folgenden Basismaschinen sind leicht programmierbar:

- Band i := Band $i + 1$
- Band i := Band $i - 1$
- Band i := 0
- Band i := Band j

Seien $M_i = (Q_i, \Sigma, \Gamma_i, \delta_i, q_i, \Box, F_i)$, $i = 1, 2$.
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:

$$M := (Q_1 \cup Q_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, q_1, \Box, F_2)$$

wobei (oE) $Q_1 \cap Q_2 = \emptyset$ und

$$\delta := \delta_1 \cup \delta_2 \cup \{(f_1, a) \mapsto (q_2, a, N) \mid f_1 \in F_1, a \in \Gamma_1\}$$

Sind f_1 und f_2 Endzustände von M so bezeichnet

$$\begin{array}{ccccc} \longrightarrow & M & \xrightarrow{f_1} & M_1 & \longrightarrow \\ & \downarrow f_2 & & & \\ & M_2 & & & \\ & \downarrow & & & \end{array}$$

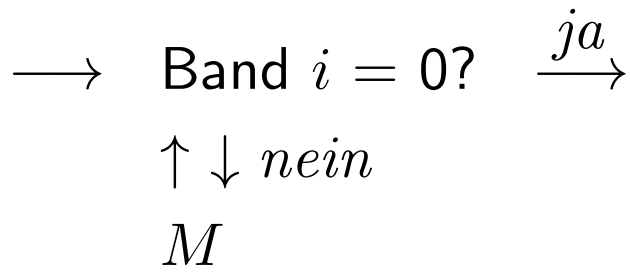
eine Fallunterscheidung, dh eine TM, die vom Endzustand f_1 von M nach M_1 übergeht, und von f_2 aus nach M_2.

Die folgende TM nennen wir „Band=0?“ (bzw „Band $i = 0$?“):

$$\begin{array}{rcll} \delta(q_0, 0) & = & (q_0, 0, R) & \\ \delta(q_0, \Box) & = & (ja, \Box, L) & \\ \delta(q_0, a) & = & (nein, a, N) & \text{für } a \neq 0, \Box \end{array}$$

wobei ja und $nein$ Endzustände sind.

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`

5.4 WHILE- und GOTO-Berechenbarkeit

WHILE	$\equiv$	strukturierte Programme mit `while`-Schleifen
GOTO	$\equiv$	Assembler

Wir definieren WHILE- und GOTO-Berechenbarkeit und zeigen ihre Äquivalenz mit Turing-Berechenbarkeit.

Syntax von WHILE-Programmen:

$$
\begin{aligned}
P \;\rightarrow\; & X \texttt{ := } X \texttt{ + } C \\
\mid\; & X \texttt{ := } X \texttt{ - } C \\
\mid\; & P \texttt{ ; } P \\
\mid\; & \texttt{IF } X = 0 \texttt{ DO } P \texttt{ ELSE } Q \texttt{ END} \\
\mid\; & \texttt{WHILE } X \neq 0 \texttt{ DO } P \texttt{ END}
\end{aligned}
$$

wobei X eine der Variablen $x_0, x_1, \ldots$
und C eine der Konstanten $0, 1, \ldots$ sein kann.

Beispiel 5.17

`WHILE` $x_2 \neq 0$ `DO` x_1 `:=` x_0`+1 END`

Die modifizierte Differenz ist $m \dot{-} n := \begin{cases} m - n & \text{falls } m \geq n \\ 0 & \text{sonst} \end{cases}$

Semantik von WHILE-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.
`WHILE` $x_i \neq 0$ `DO` P `END`	Führe P bis die Variable x_i (wenn je) den Wert 0 annimmt.

Beispiel 5.18

`WHILE` $x_1 \neq 0$ `DO` x_2 `:=` x_2 `+` 1`;` $x_1 := x_1$ `-` 1 `END`
simuliert

Zu Beginn der Ausführung stehen die Eingaben in $x_1, \ldots, x_k$.
Alle anderen Variablen sind 0. Die Ausgabe wird in x_0 berechnet.

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`;`

(falls $i \neq k$)

```
WHILE x_k ≠ 0 DO
  x_i := x_i + 1; x_k := x_k - 1
END
```

x_i `:=` x_j `*` x_k $\equiv$ x_i `:=` 0`;`

(falls $i \neq j, k$)

```
WHILE x_k ≠ 0 DO
  x_i := x_i + x_j; x_k := x_k - 1
END
```

`DIV`, `MOD`, geschachtelte Ausdrücke

Definition 5.19

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist WHILE-berechenbar gdw es ein WHILE-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.)

- terminiert mit $f(n_1, \ldots, n_k)$ in x_0, falls $f(n_1, \ldots, n_k)$ definiert ist,
- terminiert nicht, falls $f(n_1, \ldots, n_k)$ undefiniert ist.

Turingmaschinen können WHILE-Programme simulieren:

Satz 5.20 (WHILE $\to$ 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. □

Übersetzungen

Ein GOTO-Programm ist eine Sequenzen von markierten Anweisungen

$$M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$$

(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 5.21 (WHILE $\rightarrow$ GOTO)

Jedes WHILE-Programm kann durch ein GOTO-Programm simuliert werden.

Satz 5.22 (GOTO $\rightarrow$ WHILE)

Jedes GOTO-Programm kann durch ein WHILE-Programm simuliert werden.

Beweis: Simuliere $M_1 : A_1;\ M_2 : A_2;\ \ldots;\ M_k : A_k$ durch

pc := 1;
WHILE $pc \neq 0$ DO
  IF $pc - 1 = 0$ THEN P_1 ELSE
  $\vdots$
  IF $pc - k = 0$ THEN P_k ELSE $pc := 0$ END ... END
END

wobei P_i A_i simuliert und $A_i \mapsto P_i$ 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

Korollar 5.23

WHILE- und GOTO-Berechenbarkeit sind äquivalent.

Korollar 5.24 (Kleenesche Normalform)

Jedes WHILE-Programm ist zu einem WHILE-Programm mit genau einer WHILE-Schleife äquivalent.

Übersetzungen

Satz 5.25 (TM $\rightarrow$ GOTO)

Jede TM kann durch ein GOTO-Programm simuliert werden.

Übersetzung: TM $(Q, \Sigma, \Gamma, \delta, q_0, \square, F) \rightarrow$ GOTO-Programm.

$$Q = \{q_0, \ldots, q_k\} \qquad \Gamma = \{a_0 (= \square), \ldots, a_n\}$$

Eine Konfiguration

$$(a_{i_p} \ldots a_{i_1},\ q_l,\ a_{j_1} \ldots a_{j_q})$$

wird durch die Programmvariablen x, y, z wie folgt repräsentiert:

$$x = (i_p \ldots i_1)_b, \quad y = (j_q \ldots j_1)_b, \quad z = l$$

wobei $(i_p \ldots i_1)_b$ die Zahl $i_p \ldots i_1$ zur Basis $b := n + 1$ ist:

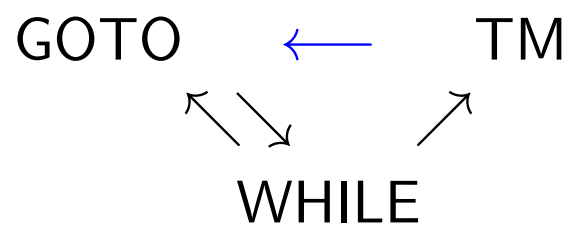
$$x = \sum_{r=1}^{p} i_r b^{r-1}$$

Der Kern des GOTO-Programms ist die iterierte Simulation von δ:

$$
\begin{array}{ll}
M\colon & \texttt{IF } z \in F \texttt{ GOTO } M_{end}\texttt{;} \\
& a \texttt{ := } y \texttt{ MOD } b\texttt{;} \\
& \texttt{IF } z = 0 \texttt{ AND } a = 0 \texttt{ GOTO } M_{00}\texttt{;} \\
& \texttt{IF } z = 0 \texttt{ AND } a = 1 \texttt{ GOTO } M_{01}\texttt{;} \\
& \ldots \\
& \texttt{IF } z = k \texttt{ AND } a = n \texttt{ GOTO } M_{kn}\texttt{;} \\
M_{00}\colon & P_{00}\texttt{; GOTO } M\texttt{;} \\
M_{01}\colon & P_{01}\texttt{; GOTO } M\texttt{;} \\
& \ldots \\
M_{kn}\colon & P_{kn}\texttt{; GOTO } M
\end{array}
$$

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 DIV b;	Löschen von a_j
y := b*y + s;	Schreiben von a_s
y := b*y + (x MOD b);	Bewegung L (I): Einfügen von a_{i_1} in y
x := x DIV b	Bewegung L (II): Löschen von a_{i_1} aus x



Übersetzungen

LOOP–Berechenbarkeit

Statt `while`-Schleifen betrachten wir nun `for`-Schleifen.

LOOP-Programme haben die gleiche Syntax WHILE-Programme aber statt der WHILE-Schleifen gibt es nur LOOP-Schleifen mit folgender Syntax:

$$\texttt{LOOP } X \texttt{ DO } P \texttt{ END}$$

Semantik:
Führe P genau n mal aus, wobei n der *Anfangswert* von X ist.
Zuweisungen an X in P ändern die Anzahl n der Schleifendurchläufe nicht.

Beispiel 5.26

`LOOP` x_0 `DO` x_1 `:=` x_1`+1` `END` simuliert

Definition 5.27

Eine Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist LOOP-berechenbar gdw es ein LOOP-Programm P gibt, so dass für alle $n_1, \ldots, n_k \in \mathbb{N}$:

P, gestartet mit $n_1, \ldots, n_k$ in $x_1, \ldots, x_k$ (0 in den anderen Var.) terminiert mit $f(n_1, \ldots, n_k)$ in x_0.

Lemma 5.28

Alle LOOP-berechenbaren Funktionen sind total.

Beweis mit Induktion über die Syntax der LOOP-Programme.

Sind alle totalen Funktionen LOOP-berechenbar?

Fakt 5.29

WHILE-Schleifen können LOOP-Schleifen simulieren.

Fakt 5.30

LOOP-Schleifen können WHILE-Schleifen nicht simulieren.

5.5 Primitiv rekursive Funktionen

Klassen von Programmiersprachen

Imperativ	Funktional
C, Java,. . . TM, WHILE, GOTO	OCaml, Lisp, . . . μ-rekursiv
LOOP	Primitiv rekursiv

Wir betrachten Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$.
Wir identifizieren $\mathbb{N}^0 \to \mathbb{N}$ 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

$$\begin{aligned} f(0) &= 1 \\ f(n+1) &= n * f(n) \end{aligned}$$

Definition 5.31 (Basisfunktionen)

- Die konstante Funktion 0
- Die Nachfolgerfunktion $s(n) = n + 1$
- Die Projektionsfunktionen $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$, $1 \leq i \leq k$:

$$\pi_i^k(x_1, \ldots, x_k) = x_i$$

Definition 5.32

Die Komposition von g und $h_1, \ldots, h_k$ erzeugt die Funktion f

$$f(\overline{x}) \;=\; g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

wobei $\overline{x} = (x_1, \ldots, x_n)$ und

$$\begin{aligned} f &: \mathbb{N}^n \to \mathbb{N} \\ g &: \mathbb{N}^k \to \mathbb{N} \\ h_i &: \mathbb{N}^n \to \mathbb{N} \qquad (i = 1, \ldots, k) \end{aligned}$$

Definition 5.33

Das Schema der primitiven Rekursion erzeugt aus g und h die Funktion f

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

wobei $\overline{x} = (x_1, \ldots, x_n)$ und

$$\begin{aligned} f &: \mathbb{N}^{n+1} \to \mathbb{N} \\ g &: \mathbb{N}^{n} \to \mathbb{N} \\ h &: \mathbb{N}^{n+2} \to \mathbb{N} \end{aligned}$$

Definition 5.34 (PR)

Die Menge PR der primitiv rekursiven Funktionen ist die folgende induktiv definierte Teilmenge aller Funktionen $\mathbb{N}^k \to \mathbb{N}$, $k \geq 0$:

- Die Basisfunktionen 0, s, und π_i^k sind primitiv rekursiv.
- Sind g und h_i primitiv rekursiv, dann auch ihre Komposition

$$f(\overline{x}) \;=\; g(h_1(\overline{x}), \ldots, h_k(\overline{x}))$$

- Sind g und h primitiv rekursiv, dann auch die mit primitiver Rekursion definierte Funktion

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(m+1, \overline{x}) &= h(f(m, \overline{x}), m, \overline{x}) \end{aligned}$$

Lemma 5.35

Jede primitiv-rekursive Funktion ist total.

Beweis:

Induktion über den Aufbau der primitiv-rekursiven Funktionen. □

Beispiel 5.36

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.

Beispiel 5.37

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 $mult$.

Definition 5.38

Wir bezeichen f als eine erweiterte Komposition der Funktionen $g_1, \ldots, g_k$ falls

$$f(x_1, \ldots, x_n) \;=\; t$$

so dass t ein Ausdruck ist, der nur aus den Funktionen $g_1, \ldots, g_k$ und den Variablen $x_1, \ldots, x_n$ besteht.

Beispiel: $f(x, y) = g_1(x, g_2(y, g_3(x)))$

Lemma 5.39

Eine erweiterte Komposition von PR Funktionen ist wieder PR.

Beweis:

Mit Induktion über Aufbau/Größe von t. □

Beispiel:

$$\begin{aligned} h_1(x, y) &= g_3(\pi^2_1(x, y)) \\ h_2(x, y) &= g_2(\pi^2_2(x, y), h_1(x, y)) \\ f(x, y) &= g_1(\pi^2_1(x, y), h_2(x, y)) \end{aligned}$$

Das erweiterte Schema der primitiven Rekursion erlaubt

$$\begin{aligned} f(0, \overline{x}) &= t_0 \\ f(m+1, \overline{x}) &= t \end{aligned}$$

wobei

- t_0 enthält nur PR Funktionen und die x_i,
- t enthält nur $f(m, \overline{x})$, PR Funktionen, m und die x_i.

Lemma 5.40

Das erweiterte Schema der primitiven Rekursion führt nicht aus PR hinaus.

Beweis:

Mit Hilfe der erweiterten Komposition. □

Moral:

Primitive Rekursion erlaubt $f(m+1, \overline{x}) = \ldots f(m, \overline{x}) \ldots$

Beispiel 5.41

Die Vorgängerfunktion ist PR:

$$\begin{aligned} pred(0) &= 0 \\ pred(x+1) &= x \end{aligned}$$

Die modifizierte Differenz ist PR:

$$\begin{aligned} x \dot{-} 0 &= x \\ x \dot{-} (y+1) &= pred(x \dot{-} y) \end{aligned}$$

Definition 5.42

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

$$\hat{P} : \mathbb{N} \to \{0, 1\}$$

zuordnen: $\hat{P}(x) = 1$ gdw $P(x) =$ **true**.

Wir nennen P primitiv rekursiv genau dann, wenn $\hat{P}$ primitiv rekursiv ist.

Ist P primitiv rekursiv, dann auch der beschränkte max-Operator

$$\max\{x \leq n \mid P(x)\} \;=:\; q(n)$$

wobei $\max \emptyset := 0$.

$$\begin{aligned} q(0) &= 0 \\ q(n+1) &= q(n) + \hat{P}(n+1) * ((n+1) \dot{-} q(n)) \\ &= \begin{cases} n+1 & \text{falls } P(n+1) \\ q(n) & \text{sonst} \end{cases} \end{aligned}$$

Ist P primitiv rekursiv, dann auch der beschränkte Existenzquantor

$$\exists x \leq n.\ P(x) \quad =:\ Q(x)$$

denn:

$$\begin{aligned}
\hat{Q}(0) &= \hat{P}(0) \\
\hat{Q}(n+1) &= \hat{Q}(n) + \hat{P}(n+1) * (1 \dot{-} \hat{Q}(n)) \\
&= \begin{cases} 1 & \text{falls } P(n+1) \\ \hat{Q}(n) & \text{sonst} \end{cases}
\end{aligned}$$

5.6 PR = LOOP

Hauptproblem bei LOOP $\rightarrow$ PR:
Kodierung *aller* Variablen eines LOOP Programms in *einer* Zahl.

Satz 5.43
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:

$$\begin{aligned} \binom{0}{2} &= 0 \\ \binom{n+1}{2} &= \binom{n}{2} + n \end{aligned}$$

Mit Komposition ist auch c PR:

$$c(x, y) = \binom{x + y + 1}{2} + x$$

Mit c kodiert man $k+1$ Tupel:

$$\langle n_0, n_1, \ldots, n_k \rangle \;:=\; c(n_0, c(n_1, \ldots, c(n_k, 0)\ldots))$$

Wir brauchen die Umkehrfunktionen p_1 und p_2 von c:

$$p_1(c(x,y)) = x \qquad p_2(c(x,y)) = y$$

Damit kann man Projektionsfunktionen auf Tupeln definieren:

$$\begin{array}{lcl} d_0(n) & := & p_1(n) \\ d_1(n) & := & p_1(p_2(n)) \\ & \vdots & \\ d_k(n) & := & p_1(\underbrace{p_2 \ldots p_2(}_{k} n)\ldots) \end{array}$$

Sind p_1, p_2 PR, so auch $d_0, \ldots, d_k$.

Satz 5.44

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.

□

Satz 5.45 (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 $x_0, \ldots, x_k$), das f berechnet.
Mit Induktion über die Struktur von P
konstruieren wir eine PR Funktion g_P mit

$$g_P(\underbrace{\langle a_0, \ldots, a_k \rangle}) = \underbrace{\langle b_0, \ldots, b_k \rangle}$$

Belegung der Variablen beim Start von P

Belegung der Variablen am Ende von P

Beweis (Forts.):

- P ist x_i `:=` $x_j \pm c$:

$$g_P(x) = \langle d_0(x), \ldots, d_{i-1}(x), d_j(x) \pm c, d_{i+1}(x), \ldots, d_k(x) \rangle$$

- 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 $f : \mathbb{N}^m \to \mathbb{N}$, dann ist f PR denn

$$f(x_1, \ldots, x_m) = d_0(g_P(\langle 0, x_1, \ldots x_m, \underbrace{0, \ldots, 0}_{k-m} \rangle)).$$

Beweis (Forts.): PR $\rightarrow$ LOOP

Sei $f : \mathbb{N}^k \rightarrow \mathbb{N}$ PR.

Mit Induktion über die Konstruktion 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 $\pi^3_2(x_1, x_2, x_3) = x_2$:

  $x_0 \ := \ x_2$

- Komposition, zB $f(x_1, x_2) = g(h_1(x_1, x_2), h_2(x_1, x_2))$:

  $x_{13} \ := \ x_1 ; \ x_{14} \ := \ x_2 ; \ P_{h_1} ; \ x_{15} \ := \ x_0 ;$
  $x_1 \ := \ x_{13} ; \ x_2 \ := \ x_{14} ; \ P_{h_2} ; \ x_{16} \ := \ x_0 ;$
  $x_1 \ := \ x_{15} ; \ x_2 \ := \ x_{16} ; \ P_g$

  Funktions-Komposition wird simuliert durch Hintereinanderausführung (;) wobei Zwischenergebnisse in separaten Variablen zwischengespeichert werden müssen.

Beweis (Forts.):

- Primitive Rekursion:

$$\begin{aligned} f(0, \overline{x}) &= g(\overline{x}) \\ f(y+1, \overline{x}) &= h(f(y, \overline{x}), y, \overline{x}) \end{aligned}$$

Folgendes LOOP-Programm berechnet $f(y, \overline{x})$:

r := $g(\overline{x})$;
k := 0;
LOOP y DO
  r := $h(r, k, \overline{x})$
  k := $k + 1$;
END
x_0 := r

wobei wir nach Induktionsannahme LOOP-Programme zur Berechnung von g und h konstruieren können.

Reversible Kodierung von Zahlenfolgen als Zahlen:

$\{0, \ldots, k\}^*$ Kodiere $(i_1, \ldots, i_n)$ als Zahl $i_1 \ldots i_n$ zur Basis $k+1$.

$\mathbb{N}^n$ Mit iterierter Paarfunktion c: $\langle i_1, \ldots, i_n \rangle$
NB n muss beim Dekodieren bekannt sein denn zB
$1 = c(0,1) = c(0, c(0,1))$.

$\mathbb{N}^*$ Auch mit $\langle \ldots \rangle$ reversibel kodierbar (Übung)

$\mathbb{N}^*$ Kodiere $(i_1, \ldots, i_n)$ als $2^{i_1} 3^{i_2} \cdots p_n^{i_n}$
wobei p_n die n-te Primzahl ist.
Dekodierung = Primzahlzerlegung

5.7 Die μ-rekursiven Funktionen

- Mit PR kann man nur beschränkt suchen: $n, \ldots, 0$.
- Die *unbschränkte* Suche $0, \ldots$ erfordert so etwas wie

$$f(n) \;=\; \ldots f(n+1) \ldots$$

- 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.“

Definition 5.46 (μ-Operator)

Sei $f : \mathbb{N}^{k+1} \to \mathbb{N}$ eine (nicht notwendigerweise totale) Funktion. Die durch Anwendung des μ-Operators entstehende Funktion $\mu f : \mathbb{N}^k \to \mathbb{N}$ ist definiert durch:

$$\overline{x} \mapsto \begin{cases} \min\{n \in \mathbb{N} \mid f(n, \overline{x}) = 0\} & \text{falls} \\ & \text{ein solches } n \text{ existiert und} \\ & f(m, \overline{x}) \neq \perp \text{ für alle } m \leq n \\ \perp & \text{sonst} \end{cases}$$

Intuitiv: $\mu f(\overline{x}) = \mathit{find}(0, \overline{x})$

$\mathit{find}(n, \overline{x}) = \textbf{if } f(n, \overline{x}) = 0 \textbf{ then } n \textbf{ else } \mathit{find}(n+1, \overline{x})$

Beispiel 5.47

Ist $f(n, x) = x \dot{-} (n + n)$

dann ist $\mu f(x) =$

Definition 5.48 (μR)

Die Menge der μ-rekursiven Funktionen ist induktiv wie folgt definiert:

- Die Basisfunktionen 0, $+1$, π_i^k sind μ-rekursiv.
- Wenn eine Funktion durch

  Komposition, primitive Rekursion, oder den μ-Operator

  aus μ-rekursive Funktionen definiert werden kann, ist sie μ-rekursiv.

Satz 5.49 (μR = WHILE)

Die μ-rekursiven sind genau die WHILE-berechenbaren Funktionen.

Beweis: als Ergänzung des Beweises von PR = LOOP.
μR $\rightarrow$ WHILE:
Fall μf: Nach IA gibt es ein WHILE-Programm für f.
Dann ist μf auch WHILE-berechenbar:

x_0 := 0; y := $f(0, \overline{x})$;
WHILE $y \neq 0$ DO x_0 := x_0 + 1; y := $f(x_0, \overline{x})$ END

Beweis (Forts.):

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.

$$g_P(x) = h(\mu h_i(x), x)$$

□

Durch die Transformation

$$\mu\text{R} \to \text{WHILE mit einer Schleife (Kleene)} \to \mu\text{R}$$

genügt also immer ein μ-Operator:

Korollar 5.50 (Kleene)

Für jede n-stellige μ-rekursive Funktionen f gibt es zwei $n+1$-stellige PR Funktionen h und h', so dass

$$f(\overline{x}) = h(\mu h'(\overline{x}), \overline{x})$$

5.8 Die Ackermann-Funktion

$$\begin{aligned} a(\quad 0, \quad n) &= n+1 \\ a(m+1, \quad 0) &= a(m,1) \\ a(m+1,n+1) &= a(m,a(m+1,n)) \end{aligned}$$

- Dies ist keine PR Definition.
- Aber: $\not\Rightarrow$ a ist nicht PR
- Ziel: a ist berechenbar, total, aber nicht PR

Fakt 5.51

Die Ackermann-Funktion ist (OCaml-)berechenbar.

Beispiel:

$$
\begin{aligned}
&a(2,2) = \\
&a(1,a(2,1)) = \\
&a(1,a(1,a(2,0))) = \\
&a(1,a(1,a(1,1))) = \\
&a(1,a(1,a(0,a(1,0)))) = \\
&a(1,a(1,a(0,a(0,1)))) = \\
&a(1,a(1,a(0,2))) = \\
&a(1,a(1,3)) = \\
&a(1,a(0,a(1,2))) = \\
&a(1,a(0,a(0,a(1,1)))) = \\
&a(1,a(0,a(0,a(0,a(1,0))))) = \\
&a(1,a(0,a(0,a(0,a(0,1))))) = \\
&a(1,a(0,a(0,a(0,2)))) = \\
&a(1,a(0,a(0,3))) = \\
&a(1,a(0,4)) = \\
&a(1,5) = \\
&a(0,a(1,4)) = \\
&a(0,a(0,a(1,3))) = \\
&a(0,a(0,a(0,a(1,2)))) = \\
&a(0,a(0,a(0,a(0,a(1,1))))) = \\
&a(0,a(0,a(0,a(0,a(0,a(1,0)))))) = \\
&a(0,a(0,a(0,a(0,a(0,a(0,1)))))) = \\
&a(0,a(0,a(0,a(0,a(0,2))))) = \\
&a(0,a(0,a(0,a(0,3)))) = \\
&a(0,a(0,a(0,4))) = \\
&a(0,a(0,5)) = \\
&a(0,6) = \\
&7
\end{aligned}
$$

Scherzfrage: $a(6,6) = ?$

Der Beginn:

$$\begin{aligned}
a(0,n) &= n+1 \\
a(1,n) &= 2+(n+3)-3 \\
a(2,n) &= 2(n+3)-3 \\
a(3,n) &= 2^{n+3}-3 \\
a(4,n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}}_{n+3}-3 \\
a(5,n) &= ???
\end{aligned}$$

Mit Induktion über n:

$$a(m+1,n) = \underbrace{a(m,a(m,\ldots a(m}_{n+1},1)\ldots)$$

Bsp: $a(1,n) = a(0,\ldots a(0,1)\ldots) = (+1)^{n+1}(1) = n+2$

Ackermann als Familie von Funktionen A_m: $A_m(n) := a(m,n)$.
Nun gilt:

$$\begin{aligned} A_0(n) &= s(n) \\ A_{m+1}(n) &= A_m^{n+1}(1) = \underbrace{A_m(\ldots A_m}_{n+1}(1)\ldots) \end{aligned}$$

Beobachtung: alle A_m sind total und PR (mit Ind. über m)

Mit Currying (z.B. in Haskell):

```
a 0       n  =  n+1
a (m+1)   n  =  iter (n+1) (a m) 1

iter 0       f x  =  x
iter (n+1)   f x  =  f (iter n f x)
```

Ackermann ist mit primitiver Rekursion höherer Stufe definierbar.

Man kann auch direkt zeigen:

Lemma 5.52

Die Ackermann-Funktion ist total.

Beweis:

Mit Induktion über m: $\forall n \in \mathbb{N}.\ a(m,n) \neq \bot \quad (1)$

- Basis: $a(0,n) = n+1 \neq \bot$
- Schritt: $a(m+1,n) \neq \bot \quad (2)$

  Beweis mit Induktion über n:

  - $a(m+1,0) = a(m,1) \neq \bot$ wg (1)
  - $a(m+1,n+1) = a(m,a(m+1,n)) = a(m,k) \neq \bot$ wg (2) und (1)

□

Kompakter: die lexikographische Ordnung auf den Argumenten von a nimmt bei jedem rekursiven Aufruf ab:

$$\begin{array}{rcl} (m+1,0) & > & (m,1) \\ (m+1,n+1) & > & (m,.) \\ (m+1,n+1) & > & (m+1,n) \end{array}$$

Definition 5.53 (lexikographische Ordnung)

$$(m,n) > (m',n') \quad :\Leftrightarrow \quad m > m' \vee (m = m' \wedge n > n')$$

Beispiel: $(2,3) > (2,2) > (1,42) > \cdots$

Satz 5.54

Die lexikographische Ordnung auf $\mathbb{N} \times \mathbb{N}$ *terminiert.*

Beweis:

mit geschachtelter Induktion, wie bei der Totalität von a. □

Warnung: Kein Beweis der Totalität einer rekursiven Funktion:

„denn in jedem rekursiven Aufruf wird eines der Argumente kleiner“

$$\begin{aligned} f(m+1,0) &= f(m,1) \\ f(0,n+1) &= f(1,n) \end{aligned}$$

Lemma 5.55

Für jede PR Funktion $f : \mathbb{N}^k \to \mathbb{N}$ gibt es ein $t \in \mathbb{N}$, so dass

$$\forall \overline{x} \in \mathbb{N}^k.\ f(\overline{x}) < a(t, \max \overline{x}).$$

Beweis:

Mit Induktion über den Aufbau der Definition von f. □

Prinzip: $t \approx$ Länge der Definition von f.

Details: [Felscher. *Berechenbarkeit*]

Satz 5.56

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.

$$f(t) < a(t, t) = f(t)$$

□

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:

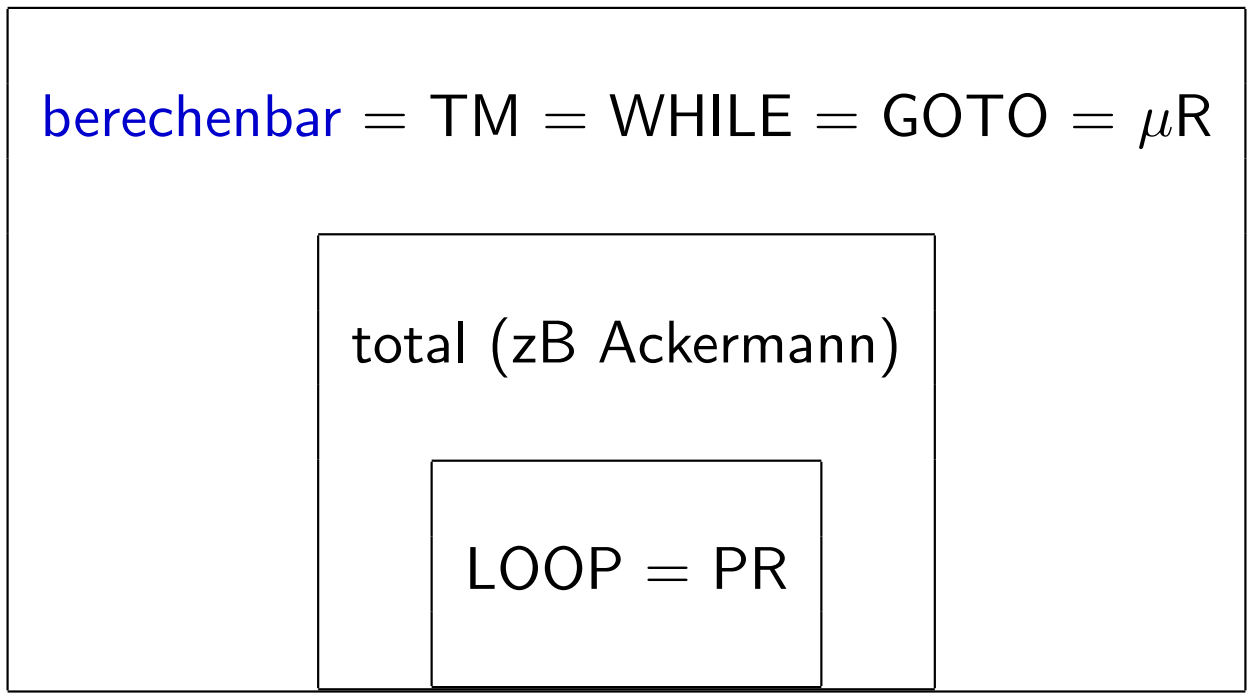
- Für fixes t ist $n \mapsto a(t, n)$ PR, denn dies ist A_t.
- A_t braucht aber eine PR Definition der Länge $O(t)$.
- Um $n \mapsto a(n, n)$ PR zu berechnen, müsste die Länge der PR Definition dynamisch mit der Eingabe wachsen.

Da die Ackermann-Funktion total, berechenbar und nicht PR ist:

Korollar 5.57

Die PR Funktionen sind eine echte Teilklasse der berechenbaren totalen Funktionen.

Die berechenbaren Funktionen



Knobelaufgabe: Sei $b : \mathbb{Z}^3 \to \mathbb{Z}$

$$b(i,j,k) \quad = \quad \textbf{if } j \leq i \textbf{ then } j \\ \textbf{else } b(b(i+1,j,k), b(j+1,k,i), b(k+1,i,j))$$

1. Finden Sie eine *nicht-rekursive* Definition $b(i,j,k) = t$ von b.
2. Sei R obige Rekursionsgleichung. Zeigen Sie

$$b(i,j,k) = t \implies R$$

3. Sonderpreis: Zeigen Sie, dass R immer terminiert.

Motto:

Alles ist beliebig kompliziert programmierbar.

5.9 Unentscheidbarkeit des Halteproblems

Ziel: Es ist unentscheidbar ob ein Programm terminiert.

Definition 5.58

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 5.59

Die entscheidbaren Mengen sind abgeschlossen unter Komplement: Ist A entscheidbar, dann auch $\overline{A}$.

Kodierung einer TM als Wort über $\Sigma = \{0, 1\}$, exemplarisch:

- Sei $\Gamma = \{a_0, \ldots, a_k\}$ und $Q = \{q_0, \ldots, q_n\}$.
- $\delta(q_i, a_j) = (q_{i'}, a_{j'}, d)$ wird kodiert als

$$\#bin(i)\#bin(j)\#bin(i')\#bin(j')\#bin(m)$$

  wobei $bin : \mathbb{N} \to \{0, 1\}^*$ 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 $\{0, 1\}^*$:

$$\begin{aligned} 0 &\mapsto 00 \\ 1 &\mapsto 01 \\ \# &\mapsto 11 \end{aligned}$$

Nicht jedes Wort über $\{0,1\}^*$ kodiert eine TM.

Sei $\hat{M}$ eine beliebige feste TM.

Definition 5.60
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.

Definition 5.61

$M[w]$ ist Abk. für „Maschine M mit Eingabe w“

$M[w]\downarrow$ bedeutet, dass $M[w]$ terminiert/hält.

Definition 5.62 (Spezielles Halteproblem)

Gegeben: Ein Wort $w \in \{0,1\}^*$.

Problem: Hält M_w bei Eingabe w ?

Als Menge:

$$K := \{w \in \{0,1\}^* \mid M_w[w]\downarrow\}$$

Satz 5.63

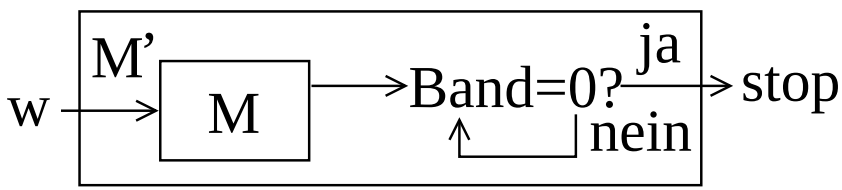
Das spezielle Halteproblem ist nicht entscheidbar.

Beweis:

Angenommen, K sei entscheidbar, dh χ_K ist berechenbar.
Dann ist auch folgende Funktion f berechenbar:

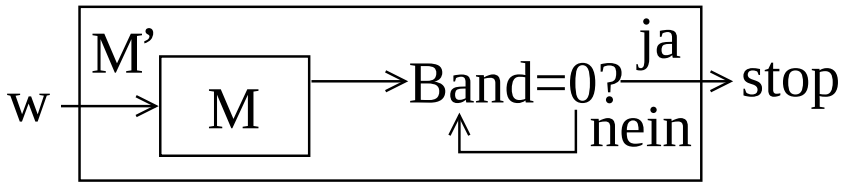
$$f(w) := \begin{cases} 0 & \text{falls } \chi_K(w) = 0 \\ \perp & \text{falls } \chi_K(w) = 1 \end{cases}$$

Denn berechnet eine TM M die Funktion χ_K,
so berechnet die folgende TM M' die Funktion f:



Dann gibt es ein w' mit $M_{w'} = M'$.

Beweis (Forts.):



$$
\begin{aligned}
f(w') = \perp &\Leftrightarrow \chi_K(w') = 1 && \text{(Def. von } f\text{)} \\
&\Leftrightarrow w' \in K && \text{(Def. von } \chi_K\text{)} \\
&\Leftrightarrow M_{w'}[w']\downarrow && \text{(Def. von } K\text{)} \\
&\Leftrightarrow M'[w']\downarrow && (M_{w'} = M') \\
&\Leftrightarrow f(w') \neq \perp && (M' \text{ berechnet } f)
\end{aligned}
$$

Wir erhalten einen Widerspruch: $f(w') = \perp \Leftrightarrow f(w') \neq \perp$.
Die Annahme, K sei entscheidbar, ist damit falsch. □

Definition 5.64 ((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 5.65

Das Halteproblem H ist nicht entscheidbar.

Beweis:

Wäre H entscheidbar, dann trivialerweise auch K:

$$\chi_K(w) = \chi_H(w, w)$$

□

Definition 5.66 (Reduktion)

Eine Menge $A \subseteq \Sigma^*$ ist reduzierbar auf eine Menge $B \subseteq \Gamma^*$ gdw es eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*$ 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.

Lemma 5.67

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 $\chi_A = \chi_B \circ f$:

$$\chi_A(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A \end{cases} = \begin{cases} 1, & f(x) \in B \\ 0, & f(x) \notin B \end{cases} = \chi_B(f(x)) \qquad \square$$

Korollar 5.68

Falls $A \leq B$ und A ist unentscheidbar, dann ist auch B unentscheidbar.

Beispiel 5.69

Da $K \leq H$ (mit Reduktion $f(w) := w\#w$) und K unentscheidbar ist, ist auch H unentscheidbar.

Satz 5.70

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 $f : \{0,1\}^* \to \{0,1\}^*$
$f(w)$ ist die Kodierung 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:

$$w \in K \;\Leftrightarrow\; M_w[w]\downarrow \;\Leftrightarrow\; M_{f(w)}[\epsilon]\downarrow \;\Leftrightarrow\; f(w) \in H_0$$

□

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

  $$P;\ x_7 := 2^{32}$$

  Variable x_7 den Wert 2^{32} erreicht.
  (OE: x_7 kommt in P nicht vor)

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?

Es müssen nicht immer TM sein:

Satz 5.71 (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:

$$H \leq H_{10}$$

□

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:

$$H \leq Eq \qquad \text{aber} \qquad Eq \not\leq H$$

5.10 Semi-Entscheidbarkeit

Definition 5.72
Eine Menge A ($\subseteq \mathbb{N}$ oder Σ^*) heißt semi-entscheidbar (s-e) gdw

$$\chi'_A(x) := \begin{cases} 1 & \text{falls } x \in A \\ \bot & \text{falls } x \notin A \end{cases}$$

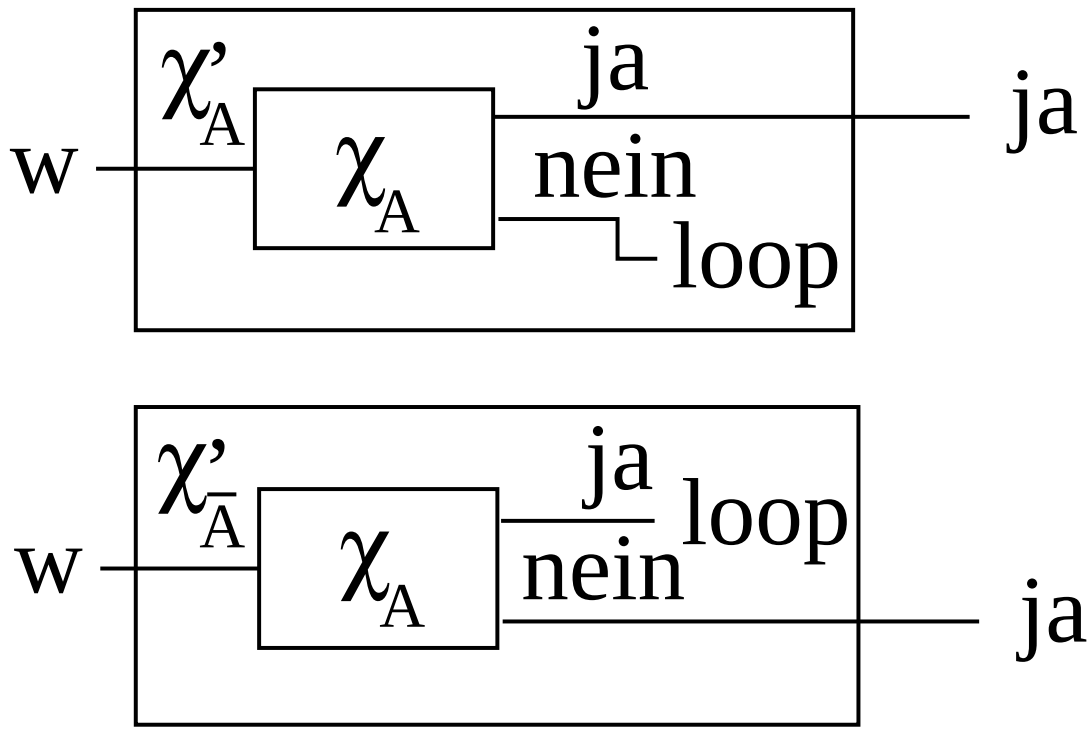
berechenbar ist.

Satz 5.73

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:



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 $s := 0, 1, 2, \ldots$ **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);
führe $M_1[x]$ und $M_2[x]$ parallel aus;
hält M_1, gib 1 aus, hält M_2, gib 0 aus. □

Lemma 5.74

Ist $A \leq B$ und ist B s-e, so ist auch A s-e.

Beweis: Übung

Definition 5.75

Eine Menge A heißt rekursiv aufzählbar (*recursively enumerable*) gdw $A = \emptyset$ oder es eine berechenbare totale Funktion $f : \mathbb{N} \to A$ gibt, so dass

$$A = \{f(0), f(1), f(2), \ldots\}$$

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:
  jede Sprache ist abzählbar aber nicht jede Sprache ist rekursiv aufzählbar (s.u.)

Lemma 5.76

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 rekursiv aufzählbar mit f. Dann ist A semi-entscheidbar:

input(x);
for $i := 0, 1, 2, \ldots$ **do**
  if $f(i) = x$ **then** output(1); **halt** **fi**

„$\Leftarrow$“: O.B.d.A. nehmen wir $A \subseteq \mathbb{N}$ an.
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.

Beweis (Forts.):

Idee: Wir benutzen eine geeignete Bijektion $c\colon \mathbb{N} \times \mathbb{N} \leftrightarrow \mathbb{N}$.
Seien $p_1 : \mathbb{N} \to \mathbb{N}$ und $p_2 : \mathbb{N} \to \mathbb{N}$ mit

$$p_1(c(n_1, n_2)) = n_1 \qquad \text{und} \qquad p_2(c(n_1, n_2)) = n_2$$

(Umkehrung von c).

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.

Vollständigktheit: Sei $a \in A \subseteq \mathbb{N}$.
Dann hält $P[a]$ nach einer endlichen Zahl k von Schritten. Dann liefert die Eingabe $n = c(a, k)$ 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 5.77

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 5.78

$\overline{K}$ ist nicht semi-entscheidbar.

Semi-Entscheidbarkeit ist nicht abgeschlossen unter Komplement.

5.11 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 5.79 (Rice)

Sei F eine Menge berechenbarer Funktionen.
Es gelte weder $F = \emptyset$ noch $F =$ alle ber. Funkt. („F nicht trivial“)
Dann ist unentscheidbar, ob die von einer gegebenen TM M_w berechnete Funktion Element F ist, dh ob $\varphi_w \in F$.

Alle nicht-triviale semantische Eigenschaften von Programmen sind unentscheidbar.

Beispiel 5.80

Es ist unentscheidbar, ob ein Programm

- für mindestens eine Eingabe hält.
  $(F = \{\varphi_w \mid \exists x.\ M_w[x]\downarrow\})$
- für alle Eingaben hält. $(F = \{\varphi_w \mid \forall x.\ M_w[x]\downarrow\})$
- bei Eingabe 42 Ausgabe 42 produziert.

Warnung

Es ist entscheidbar, ob ein Programm

- länger als 5 Zeilen ist.
- eine Zuweisung an die Variable x_{17} enhält.

Im Satz von Rice geht es um die von einem Programm berechnete Funktion (Semantik),
nicht um den Programmtext (Syntax).

Beweis:

Wir zeigen $C_F := \{w \in \{0,1\}^* \mid \varphi_w \in F\}$ ist unentscheidbar.

Fall 1: $\Omega := (x \mapsto \bot) \notin F$.

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 $f : \{0,1\}^* \to \{0,1\}^*$ und
$f(w)$ die Kodierung folgender TM:

Speichere die Eingabe x auf einem getrennten Band;
schreibe $w\#w$ auf die Eingabe und führe die universelle TM U aus;
führe M_u auf x aus.

Es gilt $\varphi_{f(w)} = \begin{cases} h & \text{falls } M_w[w]\downarrow \\ \Omega & \text{sonst} \end{cases}$ und damit

$$w \in K \;\Leftrightarrow\; M_w[w]\downarrow \overset{(*)}{\Leftrightarrow} \varphi_{f(w)} \in F \;\Leftrightarrow\; f(w) \in C_F$$

$$(*) : \begin{cases} M_w[w]\downarrow \Rightarrow \varphi_{f(w)} = h \in F \\ \varphi_{f(w)} \in F \Rightarrow \varphi_{f(w)} = h \Rightarrow M_w[w]\downarrow \end{cases}$$

Beweis (Forts.):

Fall 2: $\Omega \in F$.

Wähle berechenbares $h \notin F$.

Zeige analog, dass $\overline{K} \leq C_F$. □

Satz 5.81 (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 endliche 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.
Widerspruch

Beweis (Forts.):

Reduktion $\overline{K} \leq C_F$ mit $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ 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 eine Zahl t von Schritten.
  Damit gilt: $\varphi_{h(w)}$ ist f eingeschränkt auf $\{0, \ldots, t-1\}$.
  Nach Annahme folgt $\varphi_{h(w)} \notin F$, dh $h(w) \notin C_F$.

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 $h : \{0,1\}^* \to \{0,1\}^*$:
$h(w)$ 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

$$w \in \overline{K} \;\Leftrightarrow\; h(w) \in C_F$$

- $w \in \overline{K} \implies \neg M_w[w]\downarrow \implies \varphi_{h(w)} = g \in F \implies h(w) \in C_F$
- $w \notin \overline{K} \implies M_w[w]\downarrow \implies \varphi_{h(w)} = f \notin F \implies h(w) \notin C_F$

□

Rice-Shapiro (in Kurzform): $C_F := \{w \mid \varphi_w \in F\}$ s-e $\Longrightarrow$

$f \in F \Leftrightarrow$ es gibt endliche Funkt. $g \subseteq f$ mit $g \in F$.

Ein Programm heißt terminierend gdw es für alle Eingaben hält.

Korollar 5.82

- *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. $\square$

Grenzen automatischer Terminationsanalyse von Programmen

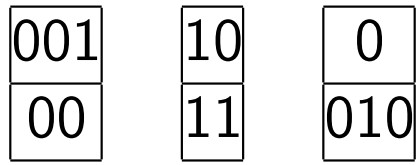
- Termination ist unentscheidbar (Rice):
  Klare Ja/Nein Antwort unmöglich.
- Termination ist nicht semi-entscheidbar (Rice-Shapiro):
  Es gibt kein Zertifizierungs-Programm,
  das alle terminierenden Programme erkennt.
- Nicht-Termination ist nicht semi-entscheidbar (Rice-Shapiro):
  Es gibt keinen perfekten *Bug Finder*,
  der alle nicht-terminierenden Programme erkennt.

Aber es gibt mächtige heuristische Verfahren, die für
relativ viele Programme aus der Praxis (Gerätetreiber)

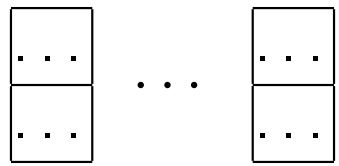
- Termination beweisen können, oder
- Gegenbeispiele finden können.

5.12 Das Postsche Korrespondenzproblem

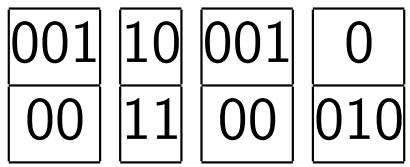
Gegeben beliebig viele Kopien der 3 „Spielkarten“



gibt es dann eine Folge dieser Karten



so dass oben und unten das gleiche Wort steht?



Kurz: 1,2,1,3.

Definition 5.83 (Postsche Korrespondenzproblem, *Post's Correspondence Problem*, PCP)

Gegeben: Eine endliche Folge $(x_1, y_1), \ldots, (x_k, y_k)$, wobei $x_i, y_i \in \Sigma^+$.

Problem: Gibt es eine Folge von Indizes $i_1, \ldots, i_n \in \{1, \ldots, k\}$, $n > 0$, mit $x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n}$?

Dann nennen wir $i_1, \ldots, i_n$ eine Lösung der Instanz $(x_1, y_1), \ldots, (x_k, y_k)$ des PCP Problems.

Beispiel 5.84

- 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?

Emil Post.
A Variant of a Recursively Unsolvable Problem.
Bulletin American Mathematical Society, 1946.

Emil Leon Post, 1897 (Polen) – 1954 (NY).

Lemma 5.85

Das PCP ist semi-entscheidbar.

Beweis:

Zähle die möglichen Lösungen der Länge nach auf, und probiere jeweils, ob es eine wirkliche Lösung ist. □

Wir zeigen nun:

$$H \leq MPCP \leq PCP$$

wobei

Definition 5.86 (Modifiziertes PCP, MPCP)

Gegeben: wie beim PCP

Problem: Gibt es eine Lösung $i_1, \ldots, i_n$ mit $i_1 = 1$?

Satz 5.87

$MPCP \leq PCP$

Beweis:

Für $w = a_1 \dots a_n$:

$$\begin{aligned} \overline{w} &:= \#a_1\#a_2\#\dots\#a_n\# \\ \overleftarrow{w} &:= a_1\#a_2\#\dots\#a_n\# \\ \overrightarrow{w} &:= \#a_1\#a_2\#\dots\#a_n \end{aligned}$$

$$f((x_1, y_1), \dots, (x_k, y_k)) := ((\overline{x_1}, \overrightarrow{y_1}), (\overleftarrow{x_1}, \overrightarrow{y_1}), \dots, (\overleftarrow{x_k}, \overrightarrow{y_k}), (\$, \#\$))$$

Satz 5.88

$$H \leq MPCP$$

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 $q \in F$

Aus $H \leq PCP$ folgt direkt

Korollar 5.89

Das PCP ist unentscheidbar.

Korollar 5.90

Das PCP ist auch für $\Sigma = \{0, 1\}$ unentscheidbar

Beweis:

Wir nennen dies das 01-PCP und zeigen PCP $\leq$ 01-PCP.

Sei $\Sigma = \{a_1, \ldots, a_m\}$ das Alphabet des gegebenen PCPs.

Abbildung auf ein 01-PCP:

$$\begin{array}{rcl} \widehat{a_j} & := & 01^j \\ \widehat{a_{j_1} \ldots a_{j_n}} & := & \widehat{a_{j_1}} \ldots \widehat{a_{j_n}} \end{array}$$

Dann hat $(x_1, y_1), \ldots, (x_k, y_k)$ eine Lösung gdw $(\widehat{x_1}, \widehat{y_1}), \ldots, (\widehat{x_k}, \widehat{y_k})$ eine Lösung hat.

„$\Rightarrow$“ klar, „$\Leftarrow$“ folgt da $\hat{\cdot} : \Sigma^* \to \{0, 1\}^*$ injektive ist:

$$\begin{array}{lcl} \widehat{x_{i_1}} \ldots \widehat{x_{i_n}} = \widehat{y_{i_1}} \ldots \widehat{y_{i_n}} & \Longrightarrow & \\ \widehat{x_{i_1} \ldots x_{i_n}} = \widehat{y_{i_1} \ldots y_{i_n}} & \Longrightarrow & x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n} \end{array}$$

□

Bemerkungen

- Das PCP ist entscheidbar falls $|\Sigma| = 1$
- Das PCP ist entscheidbar falls $k \leq 2$.
- Das PCP ist unentscheidbar falls $k \geq 5$.
- Für $k = 3, 4$ ist noch offen, ob das PCP unentscheidbar ist.

5.13 Unentscheidbare CFG-Probleme

- Für DFAs ist fast alles entscheidbar:
  $L(A) = \emptyset$, $L(A) = L(B)$, ...
- Für TMs ist fast nichts entscheidbar:
  $L(M) = \emptyset$, $L(M_1) = L(M_2)$, ...
- Für CFGs ist manches entscheidbar ($L(G) = \emptyset$, $w \in L(G)$),
  und manches unentscheidbar.

Satz 5.91

Für CFGs G_1, G_2 sind folgende Probleme unentscheidbar:

P_1: *Ist* $L(G_1) \cap L(G_2) = \emptyset$*?*

P_2: *Ist* $|L(G_1) \cap L(G_2)| = \infty$ *?*

P_3: *Ist* $L(G_1) \cap L(G_2)$ *kontextfrei?*

P_4: *Ist* $L(G_1) \subseteq L(G_2)$*?*

P_5: *Ist* $L(G_1) = L(G_2)$*?*

Beweis:

P_1: Ist $L(G_1) \cap L(G_2) = \emptyset$?

Reduktion von PCP auf
$P_1 := \{(G_1 \in \text{CFG}, G_2 \in \text{CFG}) \mid L(G_1) \cap L(G_2) \neq \emptyset\}$.

Beweisidee: Instanz von PCP wird abgebildet auf (G_1, G_2) die jeweils die Wörter folgender Gestalt erzeugen:

$$\text{Indexseq}_1^R \quad \text{Oben}_1 \quad \text{Unten}_2^R \quad \text{Indexseq}_2$$

$$\text{Indexseq}^R \quad \text{Wort} \quad \text{Wort}^R \quad \text{Indexseq}$$

Der Schnitt $L(G_1) \cap (G_2)$ enthält somit alle Wörter der Gestalt

$$\text{Indexseq}_1^R \quad \text{Oben}_1 \quad \text{Unten}_2^R \quad \text{Indexseq}_2$$

mit $\text{Indexseq}_1 = \text{Indexseq}_2$ und $\text{Oben}_1 = \text{Unten}_2$.

Diese Wörter sind die Lösungen der Instanz des PCPs.

Beweis (Forts.):

PCP Instanz $K = (x_1, y_1), \ldots, (x_k, y_k)$ (über $\{0, 1\}$)
wird abgebildet auf (G_1, G_2) mit:

$L(G_1) =$

$$\{\ a_{i_n} \cdots a_{i_1} x_{i_1} \cdots x_{i_n}\ \$\ y^R_{j_m} \cdots y^R_{j_1} a_{j_1} \cdots a_{j_m} \mid i_1, \ldots, j_m \in [1..k]\ \}$$

$L(G_2) =$

$$\{\ a_{i_n} \cdots a_{i_1} b_1 \ \cdots\ b_m\ \$\ b_m \ \cdots\ b_1\ a_{i_1} \cdots a_{i_n}\ \mid i_1, \ldots, i_n\ \in [1..k]\ \}$$

$$\begin{array}{lrcl} G_2: & S & \to & a_1 S a_1 \mid \cdots \mid a_k S a_k \mid T \\ & T & \to & 0T0 \mid 1T1 \mid \$ \end{array}$$

$$\begin{array}{lrcl} G_1: & S & \to & A\,\$\,B \\ & A & \to & a_1 A x_1 \mid \cdots \mid a_k A x_k \\ & A & \to & a_1 x_1 \mid \cdots \mid a_k x_k \\ & B & \to & y^R_1 B a_1 \mid \cdots \mid y^R_k B a_k \\ & B & \to & y^R_1 a_1 \mid \cdots \mid y^R_k a_k \end{array}$$

Beweis (Forts.):

$$L(G_1) = \{\ a_{i_n} \cdots a_{i_1} x_{i_1} \cdots x_{i_n}\ \$\ y^R_{j_m} \cdots y^R_{j_1} a_{j_1} \cdots a_{j_m} \mid \ldots\}$$

$$L(G_2) = \{\ a_{i_n} \cdots a_{i_1} b_1 \ \cdots\ b_m\ \$\ b_m \ \cdots\ b_1\ a_{i_1} \cdots a_{i_n} \mid \ldots\}$$

$\Longrightarrow$

$$\begin{aligned} L(G_1) \cap L(G_2) \neq \emptyset \quad &\Leftrightarrow \quad \exists i_1, \ldots, i_n.\ x_{i_1} \ldots x_{i_n} = (y^R_{i_n} \ldots y^R_{i_1})^R \\ &\Leftrightarrow \quad \exists i_1, \ldots, i_n.\ x_{i_1} \ldots x_{i_n} = y_{i_1} \ldots y_{i_n} \\ &\Leftrightarrow \quad \text{PCP Instanz } K \text{ hat eine Lösung} \end{aligned}$$

$\Longrightarrow PCP \leq P_1$ $\square$

Beweis (Forts.):

P_2: Ist $|L(G_1) \cap L(G_2)| = \infty$?

Eine PCP Instanz hat (mind.) eine Lösung gdw es ∞ viele hat.
$\Longrightarrow$ Reduktion $PCP \leq P_1$ zeigt auch $PCP \leq P_2$

P_3: Ist $L(G_1) \cap L(G_2)$ kontextfrei?

Für G_1 und G_2 aus P_1: Mit Pumping Lemma beweisbar:
$L(G_1) \cap L(G_2) \neq \emptyset \Longrightarrow L(G_1) \cap L(G_2)$ ist nicht CFL

Da $\emptyset$ eine CFL ist, gilt sogar:
$L(G_1) \cap L(G_2) \neq \emptyset \Leftrightarrow L(G_1) \cap L(G_2)$ ist nicht CFL

Damit sind beide Probleme gleich unentscheidbar.

Beweis (Forts.):

P_4/P_5: Ist $L(G_1) \subseteq / = L(G_2)$?

$L(G_1)$ und $L(G_2)$ aus P_1 sind DCFL

$\Longrightarrow$ man kann CFG G_1', G_2' berechnen mit $L(G_i') = \overline{L(G_i)}$

$G_3 :=$ „$G_1 \cup G_2'$“

Reduktionen: $K \mapsto (G_1, G_2')/(G_3, G_2')$ zeigen $PCP \leq P_4/P_5$:

K hat keine Lösung

$\Leftrightarrow L(G_1) \cap L(G_2) = \emptyset$

$\Leftrightarrow L(G_1) \subseteq L(G_2')$

$\Leftrightarrow L(G_1) \cup L(G_2') = L(G_2')$

$\Leftrightarrow L(G_3) = L(G_2')$ □

Da $L(G_1)$ und $L(G_2)$ aus dem Beweis zu P_1 DCFL sind, gilt sogar:

Korollar 5.92

Die Probleme P_1 bis P_4 in Satz 5.91 sind bereits für DCFL unentscheidbar.

Theorem 5.93 (Sénizergues 1997)

Für zwei DPDA M_1 und M_2 ist $L(M_1) = L(M_2)$ entscheidbar.

Satz 5.94

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. Reduktion von PCP Instanz auf $G_3 := „G_1 \cup G_2“$:

Instanz 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
„$\neg\Rightarrow\neg$“: $L(G_1), L(G_2)$ sind DCFL und damit nicht mehrdeutig

2/3. Reduktion von PCP Instanz auf $G_4 := „\overline{G_1} \cup \overline{G_2}“$.

Instanz unlösbar $\Rightarrow L(G_1) \cap L(G_2) = \emptyset$
$\Rightarrow L(G_4) = \Sigma^* \Rightarrow L(G_4)$ reg/det.

Instanz 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. □

Satz 5.95

Für eine CFG G und einen RE α ist $L(G) = L(\alpha)$ unentscheidbar.

Beweis:

Im Beweis des vorherigen Satzes hatten wir eine Reduktion $PCP \mapsto G_4$ mit

$$PCP \text{ unlösbar } \Leftrightarrow \; L(G_4) = \Sigma^*.$$

Sei $\Sigma = \{a_1, \ldots, a_n\}$.
Dann reduziert $PCP \mapsto (G_4, (a_1| \ldots |a_n)^*)$
das PCP auf das Problem $L(G) = L(\alpha)$, □

6. 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!

Polynomielle und exponentielle Komplexität

Taktfrequenz: 1GHz

Zeit	Problemgröße n					
	10	20	30	40	50	60
n	.01ms	.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	.1 ms	0.003s	0.02s	0.1 s	0.3 s	0.7 s
2^n	1 μs	0.001s	1 s	18 m	13 T	36 J
3^n	59 μs	3 s	2 T	385J	$2 \cdot 10^7$ J	10^{12} J

μs=Mikrosek., ms=Millisek., 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	$N_4 + 10$	$N_5 + 6$

Zentrale Frage: Für welche Probleme gibt es/gibt es keine polynomielle Algorithmen?

Komplexitätsklasse P:

P = die von DTM in polynomieller Zeit lösbaren Probleme
= die „leichten“ Probleme (*feasible* problems)

- $A \in P$ wird durch Angabe eines Algorithmus bewiesen
  Bsp: CYK beweist

$$\{(G, w) \mid G \in \mathsf{CFG}, w \in L(G)\} \in \mathsf{P}$$

- $A \notin P$ zu zeigen ist viel schwieriger!
  Für sehr viele Probleme ist es nicht bekannt,
  ob sie zu P gehören oder nicht.

Viele der wichtigsten Problem der Informatik sind der Gestalt:

$$\text{Gegeben } X\text{, gibt es } Y \text{ mit } R(X,Y) \text{ ?}$$

- SAT: $X =$ Boole'sche Formel, $Y =$ Belegung der Variablen von X, $R(X,Y) :=$ „Y erfüllt X“.
- HAMILTONKREIS: $X =$ Graph, $Y =$ Kreis von X, $R(X,Y) :=$ „Y besucht alle Knoten von X genau einmal“.
- EULERKREIS: $X =$ Graph, $Y =$ Kreis von X, $R(X,Y) :=$ „Y besucht alle Kanten von X genau einmal“.

Die Y sind „Lösungskandidaten“. In allen diesen Problemen:

- Prüfen, ob ein Kandidat tatsächlich eine Lösung ist, ist in P.
- Es gibt jedoch exponentiell viele Kandidaten.
- Deshalb hat ein naïver Suchalgorithmus, der alle Kandidaten aufzählt und prüft, exponentielle Laufzeit.

Für kein solches Problem ist bewiesen worden, dass es *nicht* in P liegt.

Die Frage, ob *alle* solche Probleme in P liegen, ist die wichtigste offene Frage der Informatik.

Äquivalente Formulierung

Diese Probleme können nichtdeterministisch in polynomieller Zeit gelöst werden: Wähle einen Kandidaten und prüfe, ob er eine Lösung ist.

Komplexitätsklasse NP:

NP = die von NTM in polynomieller Zeit lösbaren Probleme

Zentrale Frage:

$$P = NP\ ?$$

Überblick:

1. Die Komplexitätsklassen P und NP
2. NP-Vollständigkeit
3. SAT: Ein NP-vollständiges Problem
4. Weitere NP-vollständige Probleme

6.1 Die Komplexitätsklasse P

Berechnungsmodell:

DTM $=$ deterministische Mehrband-TM

Definition 6.1

$\mathit{time}_M(w)$ $:=$ Anzahl der Schritte bis die DTM $M[w]$ hält
$\in \mathbb{N} \cup \{\infty\}$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.
Klasse der in Zeit $f(n)$ entscheidbaren Sprachen:

$$\mathit{TIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{DTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{time}_M(w) \leq f(|w|)\}$$

Merke:

- n ist implizit die Länge der Eingabe
- Die DTM entscheidet die Sprache A in $\leq f(n)$ Schritten

Wir betrachten nur Polynome mit Koeffizienten $a_k, \ldots a_0 \in \mathbb{N}$:

$$p(n) = a_k n^k + \cdots + a_1 n + a_0$$

Definition 6.2

$$\mathsf{P} := \bigcup_{p \ Polynom} \mathit{TIME}(p(n))$$

Damit gilt auch

$$\mathsf{P} = \bigcup_{k \geq 0} \mathit{TIME}(O(n^k))$$

wobei

$$\mathit{TIME}(O(f)) := \bigcup_{g \in O(f)} \mathit{TIME}(g)$$

Beispiel 6.3

- $\{ww^R \mid w \in \Sigma^*\} \in \mathit{TIME}(O(n^2)) \subseteq \mathsf{P}$
- $\{(G, w) \mid G$ ist CFG $\wedge\, w \in L(G)\} \in \mathsf{P}$
- $\{(G, w) \mid G$ ist Graph $\wedge\, w$ ist Pfad in $G\} \in \mathsf{P}$
- $\{bin(p) \mid p$ Primzahl$\} \in \mathsf{P}$
- $\{(G, w) \mid G$ ist Graph $\wedge\, w$ ist Eulerkreis in $G\} \in \mathsf{P}$

Beweis durch Angabe eines Algorithmus mit Komplexität $O(n^k)$ für ein $k \geq 1$.

Bemerkungen

- $O(n \log n) \subset O(n^2)$
- $n^{\log n}, 2^n \notin O(n^k)$ für alle k

Analog zu Sprachen nennen wir eine Funktion $f : \mathbb{N} \to \mathbb{N}$ in polynomieller Zeit berechenbar, gdw es eine DTM M gibt, die f berechnet und $\mathit{time}_M(w) \leq p(|w|)$ für ein Polynom p.

- 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.

6.2 Die Komplexitätsklasse NP

Berechnungsmodell:

NTM $=$ nichtdeterministische Mehrband-TM

- NP ist 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.

Definition 6.4

$$\mathit{ntime}_M(w) := \begin{cases} \text{minimale Anzahl der Schritte} \\ \text{bis NTM } M[w] \text{ akzeptiert} & \text{falls } w \in L(M) \\ 0 & \text{falls } w \notin L(M) \end{cases}$$

Sei $f : \mathbb{N} \to \mathbb{N}$ eine totale Funktion.

$$\mathit{NTIME}(f(n)) := \{A \subseteq \Sigma^* \mid \exists \text{NTM } M.\ A = L(M) \wedge \forall w \in \Sigma^*.\ \mathit{ntime}_M(w) \leq f(|w|)\}$$

$$\text{NP} := \bigcup_{p\ \mathit{Polynom}} \mathit{NTIME}(p(n))$$

Bemerkungen:

- P $\subseteq$ NP
- Seit etwa 1970 ist offen ob P = NP.

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 Berechnungsfolgen 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“)

Beispiel 6.5

- Ein Euler-Kreis ist ein geschlossener Pfad in einem (ungerichteten) Graphen, der jede Kante genau einmal enthält.
- Satz: 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.
- $\Longrightarrow \{G \mid G \text{ hat Euler-Kreis}\} \in \mathsf{P}$

Beispiel 6.6

- 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 substanziell besseren Algorithmus kennt, als alle Permutationen auszuprobieren.

Erinnerung: Viele Probleme sind von der Art dass

- schwer ist, zu entscheiden, ob sie lösbar sind,
- leicht ist, zu entscheiden, ob ein Lösungsvorschlag eine Lösung ist.

Formalisierung:

Definition 6.7

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 $\mathit{time}_M(w\#c) \leq p(|w|)$.

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 6.8 (HAMILTON)

Zertifikat: Knotenpermutation. In polynomieller Zeit verifizierbar.

Satz 6.9

$A \in$ NP gdw es einen polynomiell beschränkten Verifikator für A gibt.

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 Berechnunsgfolge von $N[w]$ mit $\leq p(n)$ Schritten.
Ein polynomiell beschränkter Verfikator 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.

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 $i := 1, \ldots, p(|w|)$ **do**
   schreibe ein Zeichen aus Δ und gehe nach rechts
3. Führe M aus (mit Eingabe $w\#c$).

Nach Annahme gilt $\mathit{time}_M(w\#c) \leq p(|w|)$.
Damit braucht $N[w]$ $O(p(|w|))$ Schritte. □

Fazit:

P sind die Sprachen, bei denen $w \in L$ schnell entschieden werden kann.

NP sind die Sprachen, bei denen ein Zertifkat 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 NP\P liegt.

6.3 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

Definition 6.10

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 $f : \Sigma^* \to \Gamma^*$ gibt, so dass für alle $w \in \Sigma^*$

$$w \in A \iff f(w) \in B$$

Lemma 6.11

Die Relation $\leq_p$ ist transitiv.

Da $p_2(p_1(n))$ ein Polynom ist falls p_1 und p_2 Polynome sind.

Beispiel 6.12

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

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 $\leq k$?

HAMILTON $\leq_p$ TSP

$$(\{1, \ldots, n\}, E) \;\mapsto\; (M, n)$$

wobei

$$M_{ij} \;:=\; \begin{cases} 1 & \text{falls } \{i,j\} \in E \\ 2 & \text{sonst} \end{cases}$$

Lemma 6.13

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 beschränkt Rechenzeit von M_f.
- Sei $B \in$ P mittels DTM M.
  Polynom q beschränkt 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 □

Ein Problem ist NP-schwer,
wenn es mindestens so schwer wie alles in NP ist:

Definition 6.14

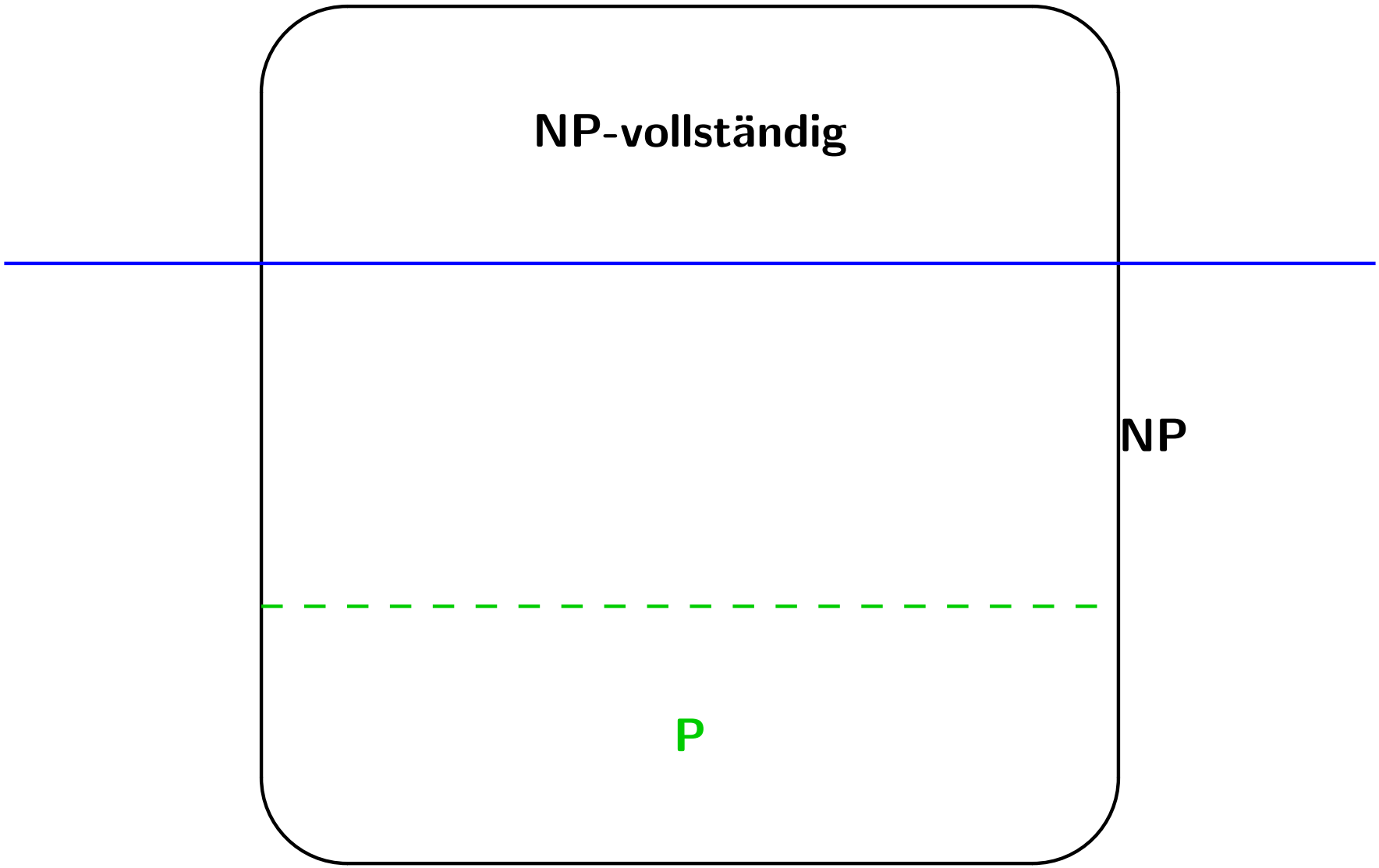
Eine Sprache L heißt NP-schwer gdw $A \leq_p L$ für alle $A \in$ NP.

Definition 6.15

Eine Sprache L heißt NP-vollständig gdw
L NP-schwer ist und $L \in$ NP.

NP-vollständige Probleme sind die schwierigsten Probleme in NP:
alle Probleme in NP sind polynomiell auf sie reduzierbar.

NP-schwer



Wie man P$\stackrel{?}{=}$NP lösen kann:

Lemma 6.16

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-schwer) und nach Lemma 6.13 $A \in$ 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?

Aussagenlogik

Syntax der Aussagenlogik:

$$\begin{array}{rrcl} \text{Variablen:} & X & \to & x \mid y \mid z \mid \ldots \\ \text{Formeln:} & F & \to & X \mid \neg F \mid (F \wedge F) \mid (F \vee F) \mid X \end{array}$$

Bsp: $((\neg x \wedge y) \vee (x \wedge \neg z))$

Man darf einige Klammern weglassen:

- Äußerste Klammern: $(x \vee y) \wedge z$ statt $((x \vee y) \wedge z)$
- Assoziativität: $(x \vee y \vee z) \wedge \neg x$ statt $((x \vee y) \vee z) \wedge \neg x$.

Abkürzungen:

$$F_1 \to F_2 \equiv \neg F_1 \vee F_2$$

$$\bigwedge_{i=1}^{n} F_i \equiv F_1 \wedge \cdots \wedge F_n$$

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 $\sigma(F) = 1$.

SAT

Gegeben: Eine aussagenlogische Formel F.

Problem: Ist F erfüllbar?

Lemma 6.17

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. □

Satz 6.18 (Cook 1971)

SAT ist NP-vollständig.

Beweis:

Da SAT $\in$ NP, bleibt noch zu zeigen, SAT ist NP-schwer.
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 $\mathit{ntime}_M(w) \leq p(|w|)$.

Reduktion bildet $w = x_1 \ldots x_n \in \Sigma^*$ auf eine Formel F ab.

- In polynomieller Zeit.
- So dass $w \in L(M) \Leftrightarrow F \in SAT$.

Beweis (Forts.):

Die Variablen von F beschreiben das *mögliche* Verhalten von $M[w]$:

		Intendierte Bedeutung
$zust_{t,q}$	$t = 0, \ldots, p(n)$ $q \in Q$	$zust_{t,q} = 1 \Leftrightarrow$ Zustand nach t Schritten ist q
$pos_{t,i}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots p(n)$	$pos_{t,i} = 1 \Leftrightarrow$ Kopfposition nach t Schritten ist i
$band_{t,i,a}$	$t = 0, \ldots, p(n)$ $i = -p(n), \ldots, p(n)$ $a \in \Gamma$	$band_{t,i,a} = 1 \Leftrightarrow$ Bandinhalt nach t Schritten auf Bandposition i ist Zeichen a

Jede Berechnunge von $M[w]$ ist als Variablenbelegung kodierbar, aber nicht umgekehrt.

Wir konstruieren F mit

Akzeptierende Berechnung von $M[w]$ $\Leftrightarrow$ Erfüllende Belegung von F

Beweis (Forts.):

Sei $Q = \{q_1, \ldots, q_k\}$ und $\Gamma = \{a_1, \ldots, a_l\}$.

$$F \;:=\; R \wedge A \wedge T_1 \wedge T_2 \wedge E$$

Hilfsformel „Genau ein Argument ist 1“:

$$G(v_1, \ldots, v_r) := (\bigvee_{i=1}^{r} v_i) \wedge \left(\bigwedge_{i=1}^{r-1} \bigwedge_{j=i+1}^{r} \neg(v_i \wedge v_j) \right)$$

$R \;\Leftrightarrow$ Zu jeder Zeit: Ein Zustand, eine Position, ein Zeichen pro Feld

$$R \;:=\; \bigwedge_{t=0}^{p(n)} [G(\mathit{zust}_{t,q_1}, \ldots, \mathit{zust}_{t,q_k}) \wedge G(\mathit{pos}_{t,-p(n)}, \ldots, \mathit{pos}_{t,p(n)}) \wedge \bigwedge_{i=-p(n)}^{p(n)} G(\mathit{band}_{t,i,a_1}, \ldots, \mathit{band}_{t,i,a_l})]$$

Beweis (Forts.):

$A \Leftrightarrow$ Berechnung startet aus der Anfangskonfiguration mit w

$$A \quad := \quad \mathit{zust}_{0,q_1} \wedge \mathit{pos}_{0,1} \wedge \bigwedge_{j=1}^{n} \mathit{band}_{0,j,x_j} \wedge \bigwedge_{j=-p(n)}^{0} \mathit{band}_{0,j,\Box} \wedge \bigwedge_{j=n+1}^{p(n)} \mathit{band}_{0,j,\Box}$$

Beweis (Forts.):

$T_1 \wedge T_2 \Leftrightarrow$ Berechnung respektiert δ von M

$$
\begin{aligned}
T_1 \quad := \quad & \bigwedge_{t,q,i,a} [(\mathit{zust}_{t,q} \wedge \mathit{pos}_{t,i} \wedge \mathit{band}_{t,i,a}) \\
& \rightarrow \bigvee_{(q',a',y)\in\delta(q,a)} (\mathit{zust}_{t+1,q'} \wedge \mathit{pos}_{t+1,i+y} \wedge \mathit{band}_{t+1,i,a'})] \\
T_2 \quad := \quad & \bigwedge_{t,i,a} [(\neg \mathit{pos}_{t,i} \wedge \mathit{band}_{t,i,a}) \rightarrow \mathit{band}_{t+1,i,a}]
\end{aligned}
$$

$E \approx$ Berechnung erreicht einen Endzustand

$$
E \quad := \quad \bigvee_{t} \bigvee_{q\in F} \mathit{zust}_{t,q}
$$

Beweis (Forts.):

Nun einfach zu sehen:

Akzeptierende Berechnung von $M[w] \Leftrightarrow$ Erfüllende Belegung von F

Größe von F polynomiell in n:

Mit $|Q| = k$, $|\Gamma| = l$, $|w| = n$:

$|G(v_1, \ldots, v_r)|$: $O(r^2)$

$|R|$: $O\big(p(n) \cdot \big(k^2 + (2 \cdot p(n))^2 + 2 \cdot p(n) \cdot l^2\big)\big)$

$|A|$: $O(p(n))$

$|T_1 \wedge T_2|$: $O(p(n)^2 \cdot k^2 \cdot l^2)$

$|E|$: $O(p(n) \cdot k)$

$\Longrightarrow |F| \in O(p(n)^3)$

□

Von der Lösbarkeit zur Lösung

Die Berechnung einer erfüllenden Belegung kann auf SAT "reduziert" werden

Sei F eine Formel mit den Variablen $x_1, \ldots, x_k$:

if $F \notin$ SAT **then** output("nicht lösbar")
else
  for $i := 1$ **to** k **do**
    if $F[x_i := 0] \in$ 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 $O(f(n))$
$\Longrightarrow$ Berechnung einer erf. Bel. in Zeit $O(n \cdot (f(n) + n))$
(falls es eine gibt.)

$f(n)$ polynomiell $\Longrightarrow$ $n \cdot (f(n) + n)$ polynomiell
$f(n)$ exponentiell $\Longrightarrow$ $n \cdot (f(n) + n)$ exponentiell

Bemerkungen:

- Die Reduktion 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.

Tutoren gesucht!

Diskrete Strukturen (Brandt)
Funktionale Programmierung/Info 2 (Nipkow)
Theo im SS20 (Nipkow)

- Sie hatten Spaß an DS/FP/Theo,
- hatten keine Probleme mit der Klausur,
- hätten Lust als Tutor zu arbeiten?

Begeben Sie Sich direkt zu

`http://www.in.tum.de/fuer-studierende/tutorbetrieb-der-fakultaet-fuer-informatik/`

Zur Funktionalen Programmierung:

- Haskell (aber OCaml Hacker willkommen!)
- Automatische Bepunktung der Hausaufgaben!

Von NP-schwer zu „NP-leicht“

- Bis vor ca. 20 Jahren:

  NP-vollständig = Todesurteil

- In den letzten 15 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.

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 (V, E) auf SAT Instanz:

- Variablen $= V \times \{1, 2, 3\}$. Notatation: x_{vi}
- Interpretation: $x_{vi} = 1$ gdw Knoten v hat Farbe i

Formel:

$$\bigwedge_{v \in V} G(x_{v1}, x_{v2}, x_{v3}) \wedge \bigwedge_{(u,v) \in E} \neg(x_{u1} \wedge x_{v1} \vee x_{u2} \wedge x_{v2} \vee x_{u3} \wedge x_{v3})$$

Bemerkungen

- Zeigt 3COL $\leq_p$ SAT und damit 3COL $\in$ NP.
- Zeigt nicht, dass 3COL NP-vollständig ist.

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*

Industrielle Anwendungen von SAT

1. Äquivalenztest von Schaltkreisen.

NICHTÄQUIVALENZ

Gegeben: Zwei aussagenlogische Formeln F_1, F_2.

Problem: Gibt es eine Variablenbelegung σ mit $\sigma(F_1) \neq \sigma(F_2)$?

Lemma 6.19
NICHTÄQUIVALENZ $\leq_p$ *SAT und SAT* $\leq_p$ *NICHTÄQUIVALENZ.*

Beweis:
NICHTÄQUIVALENZ $\leq_p$ SAT:

$$f(F_1, F_2) = (F_1 \wedge \neg F_2) \vee (\neg F_1 \wedge F_2)$$

SAT $\leq_p$ NICHTÄQUIVALENZ:

$$f(F) = (F, x \wedge \neg x)$$

□

2. 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!

6.4 Weitere NP-vollständige Probleme

Wie zeigt man, dass ein (weiteres) Problem B NP-vollständig ist?

- Zeige $B \in$ NP (meist trivial)
- Zeige $A \leq_p B$ für ein NP-vollständiges Problem A.

Lemma 6.20
Ist A NP-vollständig, so ist $B \in$ NP ebenfalls NP-vollständig, falls $A \leq_p B$.

Beweis:
Folgt direkt aus der Transitivität von $\leq_p$:
B ist NP-schwer, denn für $C \in$ NP gilt $C \leq_p A \leq_p B$. □

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.

Viele Varianten von SAT sind ebenfalls NP-vollständig:

Definition 6.21

- Eine Formel ist in Konjunktiver Normalform (KNF) gdw sie eine Konjunktion von Klauseln ist: $K_1 \wedge \cdots \wedge K_n$
- Eine Klausel ist eine Disjunktion von Literalen: $L_1 \vee \cdots \vee L_m$
- 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?

Satz 6.22

3KNF-SAT ist NP-vollständig.

Beweis:

Wir zeigen SAT $\leq_p$ 3KNF-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 fortgesetze 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.

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.

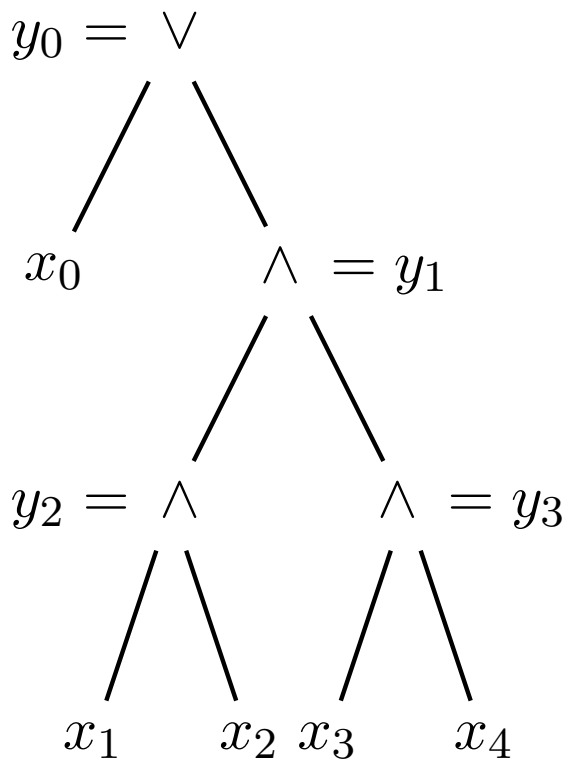
3. Betrachte die y_i als Abkürzung für die Teilbäume, an deren Wurzeln sie stehen

$$y_i = \begin{array}{ccc} & \circ_i & \\ / & & \backslash \\ l_i & & r_i \end{array}$$

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:

$$y_0 \wedge (y_0 \leftrightarrow (l_0 \circ_0 r_0)) \wedge (y_1 \leftrightarrow (l_1 \circ_1 r_1)) \cdots =: F_2$$

Beispiel: $F_1 = x_0 \vee ((x_1 \wedge x_2) \wedge (x_3 \wedge x_4))$



$F_2 =$

$$y_0 \wedge (y_0 \leftrightarrow (x_0 \vee y_1)) \wedge (y_1 \leftrightarrow (y_2 \wedge y_3)) \wedge (y_2 \leftrightarrow (x_1 \wedge x_2)) \wedge (y_3 \leftrightarrow (x_3 \wedge x_4))$$

Beweis (Forts.):

F_1 erf. $\Longrightarrow$ F_2 erf.: y_i bekommt Wert seines Teilbaums.
F_2 erf. $\Longrightarrow$ 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'.

Jede Transformation ist in polynomieller Zeit (in $|F|$) berechenbar.
Beispiel: Bei Transformation in NNF nimmt mit jedem Schritt

$$\text{Summe der } |G| \text{ für alle Teilformeln } \neg G \text{ von } F$$

ab. Daher erreicht man die NNF in $\leq |F|^2$ Schritten. Falls Formeln kopiert, nicht geteilt werden. □

Da jede Formel in 3KNF auch in KNF ist:

Korollar 6.23
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 6.24
2KNF-SAT $\in P$

Ohne Beweis

MENGENÜBERDECKUNG (MÜ)

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ einer endlichen Menge M und eine Zahl $k \leq n$.

Problem: Gibt es $i_1, \ldots, i_k \in \{1, \ldots, n\}$ mit $M = T_{i_1} \cup \cdots \cup T_{i_k}$?

Beispiel 6.25

$$\begin{array}{lllllll} T_1 & = & \{1,2\} & & T_2 & = & \{1,3\} \\ T_3 & = & \{3,4\} & & T_4 & = & \{3,5\} \\ M & = & \{1,2,3,4,5\} & & k & = & 3 \end{array}$$

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 6.26

MÜ $\in$ *NP.*

Satz 6.27

MÜ ist NP-vollständig.

Beweis: KNF-SAT $\leq_p$ MÜ.

Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

Wir konstruieren:

$$\begin{aligned} M &:= \{1, \ldots, m\} \cup \{m+1, \ldots, m+l\} \\ T_i &:= \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\ T_i' &:= \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \\ k &:= l \end{aligned}$$

F ist erfülbar

gdw

M wird durch l der Teilmengen $T_1, \ldots, T_l, T_1', \ldots, T_l'$ überdeckt

Beweis (Forts.):

Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

$$\begin{aligned} M &:= \{1, \ldots, m\} \cup \{m+1, \ldots, m+l\} \\ T_i &:= \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\ T_i' &:= \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \end{aligned}$$

Beispiel: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_4)$

$\Longrightarrow$

$$\begin{aligned} T_1 &= \{1, 2+1\} & T_1' &= \{2, 2+1\} \\ T_2 &= \{1, 2, 2+2\} & T_2' &= \{2+2\} \\ T_3 &= \{1, 2+3\} & T_3' &= \{2+3\} \\ T_4 &= \{2, 2+4\} & T_4' &= \{2+4\} \end{aligned}$$

Überdeckung: $T_1', T_2, T_3', T_4' \approx x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0$

Beweis (Forts.):

Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

$$\begin{aligned} M &:= \{1, \ldots, m\} \cup \{m+1, \ldots, m+l\} \\ T_i &:= \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\ T_i' &:= \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \end{aligned}$$

„$\Rightarrow$“ Gegeben σ mit $\sigma(F) = 1$.

$$U_i := \textbf{if } \sigma(x_i) = 1 \textbf{ then } T_i \textbf{ else } T_i'$$

Beh.: Die U_i überdecken M. Sei $j \in M$.

1. $1 \leq j \leq m$: Sei l ($= x_i$ oder $\neg x_i$) ein Literal in K_j mit $\sigma(l) = 1$.
$\Rightarrow j \in$ **if** x_i kommt positiv in K_j vor **then** T_i **else** T_i'
$\Rightarrow j \in$ **if** $\sigma(x_i) = 1$ **then** T_i **else** T_i'
$\Rightarrow j \in U_i$

2. $m+1 \leq j \leq m+k$
$\Rightarrow j \in U_{j-m}$ denn $m + (j-m) = j$

Beweis (Forts.):

Sei $F = K_1 \wedge \cdots \wedge K_m$ in KNF, mit Variablen $x_1, \ldots, x_l$.

$$\begin{aligned} M &:= \{1, \ldots, m\} \cup \{m+1, \ldots, m+l\} \\ T_i &:= \{j \mid x_i \text{ kommt in } K_j \text{ positiv vor}\} \cup \{m+i\} \\ T_i' &:= \{j \mid x_i \text{ kommt in } K_j \text{ negativ vor}\} \cup \{m+i\} \end{aligned}$$

„$\Leftarrow$“ Sei $U_1, \ldots, U_n \in \{T_1, \ldots, T_n'\}$ eine Überdeckung vom M.

$\Rightarrow$ Es gibt für alle $1 \leq i \leq n$ genau eine Menge $U_i \in \{T_i, T_i'\}$ mit $m+i \in U_i$.

Setze $\sigma(x_i) :=$ if $U_i = T_i$ then 1 else 0

$\Rightarrow \sigma(K_j) = 1$ für alle $1 \leq j \leq m$

$\Rightarrow \sigma(F) = 1$ $\square$

Das **Minimierungsproblem**

Gegeben: Teilmengen $T_1, \ldots, T_n \subseteq M$ 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.

$$f(n) \text{ polynomiell} \implies f(n) \cdot \log n \text{ polynomiell}$$
$$f(n) \text{ exponentiell} \implies f(n) \cdot \log n \text{ exponentiell}$$

Die Berechnung einer **Lösung** von

Gegeben: Teilmengen $\vec{T} := T_1, \ldots, T_n \subseteq M$ 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:

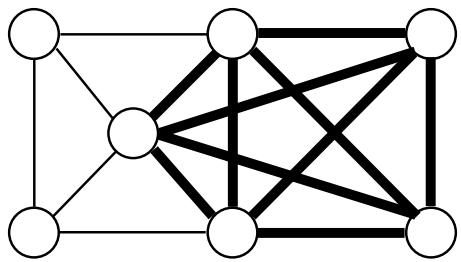
if $(\vec{T}, M, k) \notin$ MÜ **then** output("nicht lösbar")
else
  $\ddot{U} := \emptyset$
  for $i := 1$ **to** n **do**
    if $(\vec{T} - T_i, M, k) \in$ MÜ
    then $\vec{T} := \vec{T} - T_i$
    else $\ddot{U} := \ddot{U} \cup \{T_i\}$

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

$\Longrightarrow$ Clique ist Gruppe von parallelisierbaren Prozessen

Fakt 6.28

CLIQUE $\in$ *NP*

Satz 6.29

CLIQUE ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ CLIQUE:

Eine Formel F in 3KNF-SAT (oE: *genau* 3 Literale/Klausel)

$$F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \ldots \wedge (z_{k1} \vee z_{k2} \vee z_{k3})$$

wird auf einen Graphen abgebildet:

$$\begin{aligned} V &= \{(1,1),(1,2),(1,3),\ldots,(k,1),(k,2),(k,3)\} \\ E &= \{\{(i,j),(p,q)\} \mid i \neq p \text{ und } z_{ij} \neq \neg z_{pq}\} \end{aligned}$$

$$\begin{aligned} & F \text{ ist erfüllbar durch eine Belegung } \sigma \\ \iff & \text{Es gibt in jeder Klausel } i \text{ ein Literal } z_{ij_i} \text{ mit } \sigma(z_{ij_i}) = 1 \\ \iff & \text{Die Literale } z_{1j_1},\ldots,z_{kj_k} \text{ sind paarweise nicht komplementär} \\ \iff & \text{Die Knoten } (1,j_1),\ldots,(k,j_k) \text{ sind paarweise benachbart} \\ \iff & G \text{ hat eine Clique der Größe } k. \end{aligned}$$

□

RUCKSACK

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$ und $b \in \mathbb{N}$.

Problem: Gibt es $R \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in R} a_i = b$?

Satz 6.30

RUCKSACK ist NP-vollständig.

Beweis: 3KNF-SAT $\leq_p$ RUCKSACK:

Sei $F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \cdots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$, wobei

$$z_{ij} \in \{x_1, \ldots, x_n\} \cup \{\neg x_1, \neg x_2, \ldots, \neg x_n\}$$

D.h. $m =$ Anzahl der Klauseln und n Anzahl der Variablen.

Beweis (Forts.):

Wir geben Zahlen v_{i0}, v_{i1} für jede variable x_i, Zahlen k_{j1}, k_{j2} für jede Klausel K_j, und eine Zahl b an.

- Alle Zahlen haben genau $m + n$ Stellen im Dezimalsystem.
  Die j-te Stelle, $1 \leq j \leq m$, nennen wir
  "Stelle (der Klausel) K_j".
  Die $m + i$ Stelle, $1 \leq i \leq n$, nennen wir
  "Stelle (der Variable) x_i".
- $b = \underbrace{44\ldots444}_{m}\underbrace{11\ldots11}_{n}$
- An der Stelle x_i haben v_{i0}, v_{i1} eine 1, und die Zahlen und k_{j1}, k_{j2} eine 0.
  $\rightarrow$ genau eine von v_{i0}, v_{i1} muss in den Rucksack
  $\rightarrow$ modelliert die Wahl einer Belegung.

Beweis (Forts.):

Beschreibung der Zahlen $v_{i0}, v_{i1}, k_{j1}, k_{j2}$:

- v_{i0} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die $\neq x_i$ enthalten, sonst 0en.
- v_{i1} hat eine 1 an der Stelle x_i sowie an den Stellen der Klauseln, die x_i enthalten, sonst 0en.
- k_{j1} hat eine 1 an der Stell K_j, sonst 0en.
- k_{j2} hat eine 2 an der Stell K_j, sonst 0en.

Wenn die Summe einer Untermenge R die Zahl b ergibt, dann erfüllt die Belegung σ mit $\sigma(x_i) = 1$ gdw $v_{i1} \in R$ die Formel F.

Wenn F erfüllbar ist, dann wähle R so:

- Nehme eine erfüllende Belegung σ von F.
- Für jede Variable x_i: Füge v_{i1} zu R wenn $\sigma(x_i) = 1$, sonst füge v_{i0}.
- Für jede Klausel K_j: Füge k_{j1} oder k_{j2} hinzu, um eine 4 an der Stelle K_j zu gewinnen.

PARTITION

Gegeben: Zahlen $a_1, \ldots a_n \in \mathbb{N}$.

Problem: Gibt es $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = \sum_{i \notin I} a_i$?

Satz 6.31

PARTITION ist NP-vollständig.

Beweis: RUCKSACK $\leq_p$ PARTITION

Sei $a_1, a_2, \ldots, a_k, b$ Instanz von RUCKSACK und $M := \sum_{i=1}^{k} a_i$.

$$a_1, a_2, \ldots, a_k, b \;\mapsto\; a_1, a_2, \ldots, a_k, M - b + 1, b + 1$$

Beispiel: $a_1, \ldots, a_7 = 12, 7, 4, 9, 7, 3, 15$ und $b = 38$.

Lösung: Die Zahlen $7, 9, 7, 15$, dh $R = \{2, 4, 5, 7\}$

Es gilt: $M = 57, \quad M - b + 1 = 20, \quad b + 1 = 39$.

Resultierendes PARTITIONS-Problem: $12, 7, 4, 9, 7, 3, 15, \; 20, 39$

Lösung: $7 + 9 + 7 + 15 + 20 = 12 + 4 + 3 + 39$. □

BIN PACKING

Gegeben: Eine „Behältergröße“ $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$ und „Objekte“ $a_1, a_2, \ldots a_n$.

Problem: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft?

Satz 6.32

BIN PACKING ist NP-vollständig.

Beweis: PARTITION $\leq_p$ BIN PACKING.

Reduktion:

$$a_1, \ldots, a_n \;\mapsto\; b, k, a_1, \ldots, a_n$$

wobei $b := \sum_{i=1}^{n} a_i/2$ und $k := 2$. □

HAMILTONKREIS

Gegeben: Ungerichteter Graph G

Problem: Enthält G einen Hamilton-Kreis, dh einen geschlossenen Pfad, der jeden Knoten genau einmal enthält?

Satz 6.33

HAMILTON ist NP-vollständig.

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 6.34

TSP ist NP-vollständig.

Beweis: HAMILTON $\leq_p$ TSP

Reduktion:

$$(\{1, \ldots, n\}, E) \;\mapsto\; (M, n)$$

wobei

$$M_{ij} \;:=\; \begin{cases} 1 & \text{falls } \{i,j\} \in E \\ 2 & \text{sonst} \end{cases}$$

□

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 6.35

FÄRBBARKEIT ist NP-vollständig für $k \geq 3$.

Beweis:

3KNF-SAT $\leq_p$ 3FÄRBBARKEIT □

Satz 6.36

2FÄRBBARKEIT $\in P$

Die NP-Bibel, der NP-Klassiker:

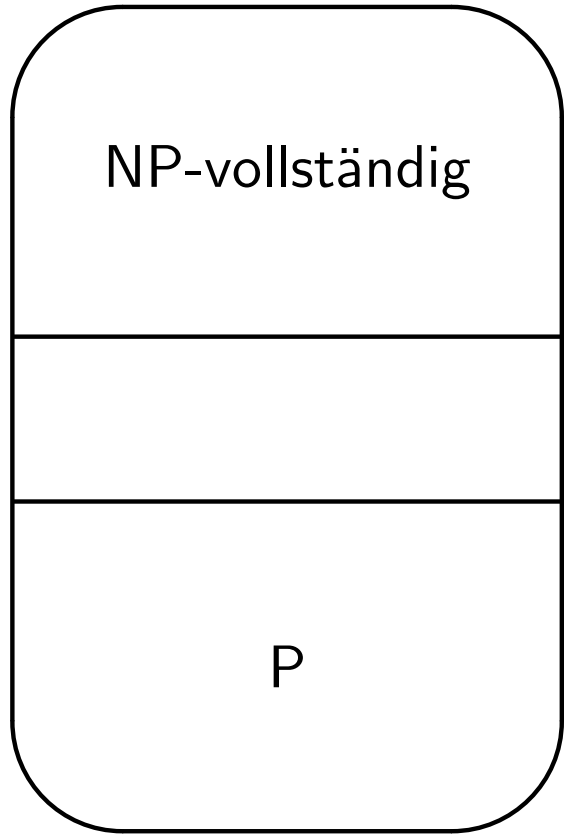
Michael Garey and David Johnson.
Computers and Intractability: A Guide to the Theory of NP-Completeness. 1979.

> *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.

Man weiß nicht ob $P = NP$. Aber man weiß (Ladner 1975)

$P \neq NP \implies NP =$



6.5 Approximation Algorithms

What to do with NP-complete problems?

- more computational power?
- encode into SAT
- approximation algorithms

Definition 6.37

A d-approximation algorithm ($d \in \mathbb{R}$) for an optimization problem is an algorithm that computes in polynomial time a solution to the problem that is at most d times worse than the optimal solution.

Travelling Salesman Problem

Definition 6.38 (TSP)

Given a *complete*, weighted, undirected graph $G = (V, E)$ with non-negative weights c: $E \to \mathbb{N}$, find a cycle that visits exactly all nodes and does so with *minimal length*.

Properties

- We can assume *triangle inequality*:

$$\forall u, v, w \in V. c(u, v) \leq c(u, w) + c(w, v)$$

- NP-complete
- We show a 2-approximation
- There is a 1.5-approximation
- There is no 123/122-approximation (since 2015)

2-Approximation Algorithm for TSP

Algorithm

1. T := a minimum spanning tree
2. cycle := traverse along depth-first search of T, jumping over visited nodes

Algorithm is

- *polynomial*
- *2-approximation*
  - $c(T) \leq$ minimal cycle
  - traversal costs $2 \cdot c(T)$ since jumping over costs at most the sum of traversed edges

Knapsack

Definition 6.39 (Knapsack)

Given weight W of knapsack and *weights* and *values* of n items: $w_1, \ldots, w_n, v_1, \ldots, v_n$, pick $I \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in I} w_i \leq W$ and $\sum_{i \in I} v_i$ is maximal.

Greedy Algorithm

- take items in the order $v_1/w_1 \geq v_2/w_2 \cdots \geq v_n/w_n$

Properties

- optimal for "fractional" knapsack problem
- for $v_1 = 1.001, w_1 = 1, v_2 = W, w_2 = W$ no better than a W-approximation.

2-Approximation of Knapsack

Modified Greedy Algorithm (ModGreedy):

- $S_1 :=$ solution by Greedy
- $S_2 :=$ item with the largest value
- Return whichever of S_1, S_2 that has more value

Lemma 6.40

ModGreedy is a 2-approximation.

Proof.

- If Greedy takes items $1, 2, \ldots, k-1$, then $\sum_{i=1}^{k} v_i \geq OPT_{frac} \geq OPT$: kth item might not be taken in full $+$ the optimal integral solution is not better than the optimal fractional solution
- $(v_1 + \cdots + v_{k-1}) + v_k \geq OPT$
- one of the two is $\geq OPT/2$
- $v(S_1) = \sum_{i=1}^{k-1} v_i$, and $v(S_2) = v_{\max} \geq v_k$

PTAS for Knapsack

- Polynomial-time approximation scheme (PTAS): any approximation ratio possible
- Idea: brute-force a part of the solution and then use Greedy Algorithm to finish up the rest

Algorithm, k fixed constant

- for all possible subsets of objects that have up to k objects: use the greedy algorithm to fill up the rest of the knapsack
- return the most profitable subset

Properties

- runtime $\mathcal{O}(kn^k)$ subsets, filling up in $\mathcal{O}(n)$
- thus total running time $\mathcal{O}(kn^{k+1})$
- $(1 + \frac{1}{k})-$approximation

7. Automaten, Berechenbarkeit, und Logik

7.1 Die Unvollständigkeit der Arithmetik

Kurt Gödel. *Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I.* 1931.

Kurt Gödel
1906 (Brünn) –
1978 (Princeton)

Syntax der Arithmetik:

$$\begin{array}{rrcl} \text{Variablen:} & V & \to & x \mid y \mid z \mid \ldots \\ \text{Zahlen:} & N & \to & 0 \mid 1 \mid 2 \mid \ldots \\ \text{Terme:} & T & \to & V \mid N \mid (T + T) \mid (T * T) \\ \text{Formeln:} & F & \to & (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F) \mid \exists V.\, F \end{array}$$

Wir betrachten $\forall x.\, F$ als Abk. für $\neg\exists x.\, \neg F$.

Definition 7.1

Ein Vorkommen einer Variablen x in einer Formel F ist gebunden gdw das Vorkommen in einer Teilformel der Form $\exists x.\, F'$ in der Teilformel F' liegt.
Ein Vorkommen ist frei gdw es nicht gebunden ist.

Notation: $F(x_1, \ldots, x_k)$ bezeichnet eine Formel F, in der höchstens $x_1, \ldots, x_k$ frei vorkommen.

Sind $n_1, \ldots, n_k \in \mathbb{N}$ so ist $F(n_1, \ldots, n_k)$ das Ergebnis der Substitution von $n_1, \ldots, n_k$ für die freien Vorkommen von $x_1, \ldots, x_k$.

Beispiel 7.2

$$\begin{aligned} F(x, y) &= (x = y \ \wedge \ \exists x.\, x = y) \\ F(5, 7) &= (5 = 7 \ \wedge \ \exists x.\, x = 7) \end{aligned}$$

Ein Satz ist eine Formel ohne freie Variablen.

Beispiel 7.3

$$\exists x.\ \exists y.\, x = y \qquad \exists y.\ \exists x.\, x = y$$

Mit S bezeichnen wir die Menge der arithmetischen Sätze.

Die Menge der wahren Sätze der Arithmetik nennen wir W:

Definition 7.4

$(t_1 = t_2) \in W$	gdw	t_1 und t_2 den selben Wert haben.
$\neg F \in W$	gdw	$F \notin W$
$(F \wedge G) \in W$	gdw	$F \in W$ und $G \in W$
$(F \vee G) \in W$	gdw	$F \in W$ oder $G \in W$
$\exists x.\, F(x) \in W$	gdw	es $n \in \mathbb{N}$ gibt, so dass $F(n) \in W$

Fakt 7.5

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.

Definition 7.6

Eine partielle Funktion $f : \mathbb{N}^k \to \mathbb{N}$ ist arithmetisch repräsentierbar gdw es eine Formel $F(x_1, \ldots, x_k, y)$ gibt, so dass für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \ldots, n_k) = m \quad \text{gdw} \quad F(n_1, \ldots, n_k, m) \in W$$

Satz 7.7

Jede WHILE-berechenbare Funktion ist arithmetisch repräsentierbar.

Satz 7.8

W ist nicht entscheidbar.

Korollar 7.9

W ist nicht semi-entscheidbar.

Anforderung an ein Beweissystem:
Es muss entscheidbar sein, ob ein gegebener Text ein Beweis einer gegebenen Formel ist.

Wir kodieren Beweise als Zahlen.

Definition 7.10

Ein Beweissystem für die Arithmetik ist ein entscheidbares Prädikat

$$Bew : \mathbb{N} \times \text{Satz} \ \rightarrow \ \{0, 1\}$$

wobei wir $Bew(b, F)$ lesen als „b ist Beweis für Formel F“.
Ein Beweissystem Bew ist korrekt gdw

$$Bew(b, F) \implies F \in W.$$

Ein Beweissystem Bew ist vollständig gdw

$$F \in W \implies \text{ es gibt } b \text{ mit } Bew(b, F).$$

Satz 7.11 (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)

□

7.2 Die Entscheidbarkeit der Presburger Arithmetik

Syntax der Presburger Arithmetik:

$$
\begin{array}{rrcl}
\text{Variablen:} & V & \to & x \mid y \mid z \mid \ldots \\
\text{Zahlen:} & N & \to & 0 \mid 1 \mid 2 \mid \ldots \\
\text{Terme:} & T & \to & V \mid N \mid T + T \\
\text{Formeln:} & F & \to & (T = T) \mid \neg F \mid (F \wedge F) \mid (F \vee F) \\
 & & \mid & \exists V.\, F
\end{array}
$$

Wir betrachten $\forall x.\, F$ als Abk. für $\neg\exists x.\, \neg F$.

Satz 7.12

Für Sätze der Presburger Arithmetik ist entscheidbar, ob sie wahr sind.

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

YAD VASHEM
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
היכל השמות - ת.ד. 3477, ירושלים 91034

Page of Testimony דף עֵד

דף עד לרישום והנצחה של הנספים בשואה; נא למלא דף עבור כל נספה בנפרד, 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

חוק זיכרון השואה והגבורה - תשי״ג 1953 קובע בסעיף מס׳ 2 כי תפקידו של יד ושם הוא לאסוף אל המולדת את זכרם של כל אלה מבני העם היהודי שנפלו ומסרו את נפשם, נלחמו ומרדו באויב הנאצי ובעוזריו ולהציב **שם** וזכר להם, לקהילות, לארגונים ולמוסדות שנחרבו בגלל השתייכותם לעם היהודי.
The Martyrs' and Heroes' Remembrance Law 5713-1953 determins 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".

Maiden name: שם משפחה לפני הנישואין:
Victim's family name: שם משפחה של הנספה: PRESBURGER
Previous/other family name: שם משפחה קודם/אחר:
First name (also nickname): שם פרטי (גם שם חיבה/כינוי): MOJZESZ 0 MIETEK ?
Approx. age at death: גיל משוער בעת המוות: 43
Date of birth: תאריך לידה: 1904
Gender: מין: M/F ז/נ
Title: תואר: philosopher
Nationality: נתינות: POLISH
Country: ארץ: POLAND
Region: מחוז: ?
Place of birth: מקום לידה: ?
Victim's father: אב הנספה: Family name: שם משפחה: First name: שם פרטי:
Victim's mother: אם הנספה: Maiden name: שם לפני הנישואין: First name: שם פרטי:

Presburger Arithmetik $=$ normale Arithmetik
ohne Multiplikation

Arithmetik : hochgradig unentscheidbar :-(
sogar sogar unvollständig :-((

$\Longrightarrow$ Hilberts 10tes Problem
$\Longrightarrow$ Gödels Theorem

Vereinfachte Presburger Formeln:

$$\begin{array}{lll} \phi & ::= & x + y = z \quad | \quad x = n \quad | \\ & & \phi_1 \wedge \phi_2 \quad | \quad \neg\phi \quad | \\ & & \exists\, x.\ \ \phi \end{array}$$

Bemerkungen

- Durch sukzessive Addition kann man Multiplikation mit Konstanten simulieren.
  (Wie?)

- Das Rucksackproblem lässt sich in PA ausdrücken.
  (Wie?)

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

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

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

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

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

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

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

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

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

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

Beobachtung:

Die Menge der erfüllenden Variablenbelegungen ist regulär !!

$$\begin{array}{lcll} \phi_1 \wedge \phi_2 & \Longrightarrow & \mathcal{L}(\phi_1) \cap \mathcal{L}(\phi_2) & \text{(Durchschnitt)} \\ \neg\phi & \Longrightarrow & \overline{\mathcal{L}(\phi)} & \text{(Komplement)} \\ \exists\, x: \quad \phi & \Longrightarrow & \pi_x(\mathcal{L}(\phi)) & \text{(Projektion)} \end{array}$$

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 ?

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

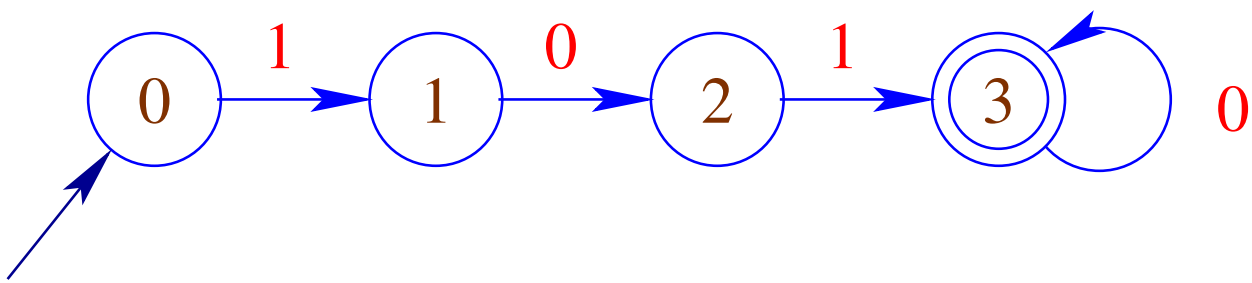
Weg-Projizierung der x-Komponente:

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.

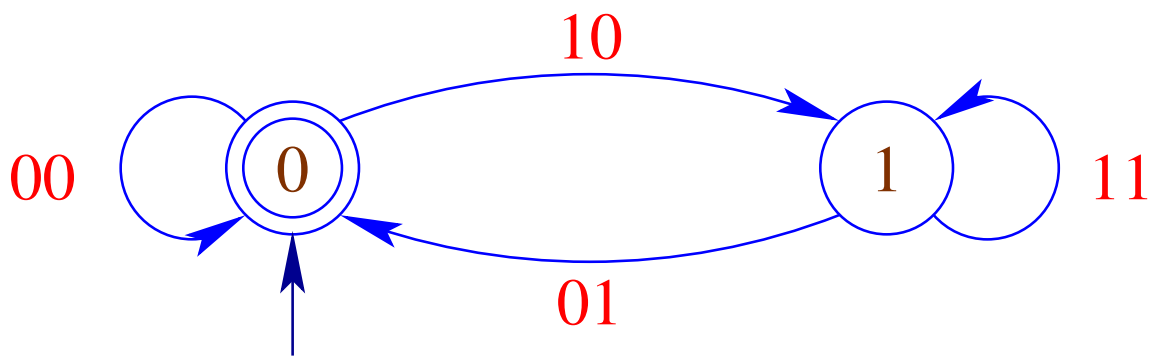
Automaten für Basis-Prädikate:

$$x = 5$$



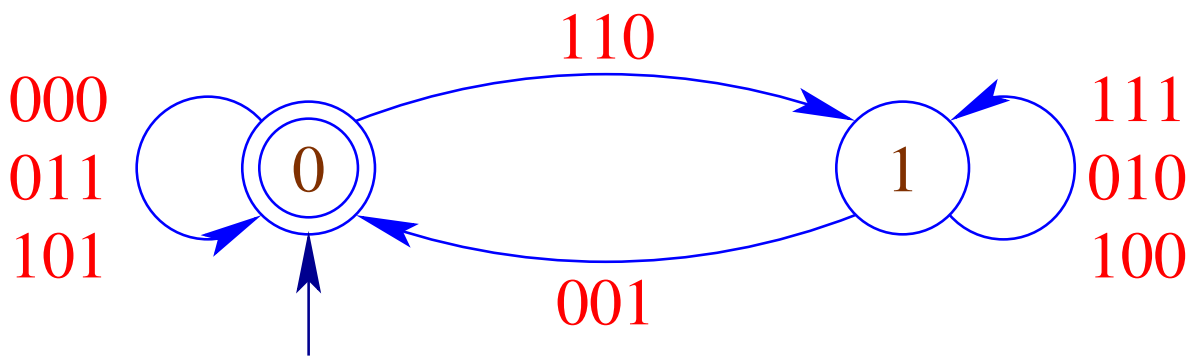
Automaten für Basis-Prädikate:

$x+x = y$



Automaten für Basis-Prädikate:

$$x+y = z$$



Ergebnisse:

Ferrante, Rackoff,1973 : $\mathrm{PSAT} \leq \mathrm{DSPACE}(2^{2^{c \cdot n}})$

Fischer, Rabin,1974 : $\mathrm{PSAT} \geq \mathrm{NTIME}(2^{2^{c \cdot n}})$