

Theoretische Informatik

K.J. Lange
Wintersemester 2004/2005

26. Oktober 2004

Inhaltsverzeichnis

1	Automaten und Formale Sprachen	2
1.1	Formale Sprachen und Grammatiken	3
1.1.1	Die Chomsky - Hierarchie	4
1.2	Reguläre Sprachen	6
1.2.1	Endliche Automaten	6
1.2.2	Nichtdeterministische endliche Automaten	7
1.2.3	Reguläre Ausdrücke	10
1.2.4	Das Pumping Lemma	14
1.2.5	Der Satz von Myhill-Nerode und endliche Automaten	15
1.2.6	Abschlusseigenschaften	18
1.2.7	Entscheidbarkeit	19
1.3	Kontextfreie Sprachen	19
1.3.1	Normalformen	20
1.3.2	Das uvwxy-Theorem	21
1.3.3	Abschlusseigenschaften	24
1.3.4	Der CYK-Algorithmus	25
1.3.5	Kellerautomaten	26
1.3.6	Deterministisch kontextfreie Sprachen	30
1.3.7	Entscheidbarkeit bei kontextfreien Sprachen	32
1.4	Kontextsensitive und Typ-0-Sprachen	33
1.5	Tabellarischer Überblick	35
2	Entscheidbarkeit und Komplexität	37
2.1	Der intuitive Begriff der Berechenbarkeit und die Churchsche These	37
2.2	Turingberechenbarkeit	38
2.3	LOOP-, WHILE-, und GOTO-Berechenbarkeit	39
2.4	Primitive und Partielle Rekursion	44
2.5	Die Ackermannfunktion	45
2.6	Halteproblem, Unentscheidbarkeit, Reduzierbarkeit	48
2.7	Das Post'sche Korrespondenzproblem	54
2.8	Anwendung des PCP	56
3	Komplexitätstheorie	57
3.1	Klassen	57
3.2	NP-Vollständigkeit	59
3.3	Weitere NP-vollständige Probleme	62
	Index	67

1 Automaten und Formale Sprachen

Grundlegende Begriffe:

- Eine endliche nicht-leere Menge von Zeichen oder Symbolen wird **Alphabet** genannt und mit dem Zeichen Σ beschrieben.
- Die Menge der möglichen Zeichenketten aus Elementen von Σ wird Σ^* geschrieben.
- Der zweistellige Operator \circ bezeichnet die Konkatenation, d.h. das Aneinanderhängen zweier Zeichenketten. Statt $u \circ v$ wird oft einfach uv geschrieben.
- Das Zeichen λ steht für das **Leere Wort**
- Die algebraische Struktur $(\Sigma^*, \circ, \lambda)$ ist ein **Monoid**, d.h. für die Menge der Zeichenketten gilt bzgl. der Konkatenation und dem leeren Wort :
 - Abgeschlossenheit v. Σ^* unter Konkatenation
 $\forall u, v \in \Sigma^* : u \circ v \in \Sigma^*$
 - Assoziativität der Konkatenation
 $\forall u, v, w \in \Sigma^* : (u \circ v) \circ w = u \circ (v \circ w)$
 - Das leere Wort ist neutrales Element bzgl. der Konkatenation
 $\forall w \in \Sigma^* : \lambda \circ w = w = w \circ \lambda$
- Für ein Wort w stellt $|w|$ die **Länge** des Wortes, also die Anzahl der Zeichen in w dar. Es gilt : $|\lambda| = 0$.
- Für $w \in \Sigma^*$, $a \in \Sigma$ ist $\#_a(w)$ die Anzahl der Vorkommen des Zeichens a in w .
- Für $\Sigma^* \ni w = a_1 \dots a_n$ bezeichnet $w^R = a_n \dots a_1$ das gespiegelte Wort.
- Abbildungen (also auch Funktionen) werden oft durch ihre Graphen (also als Menge) dargestellt. Die Funktion $F = \begin{cases} \Sigma \rightarrow \Sigma^2 \\ \sigma \mapsto \sigma\sigma \end{cases}$ wird dann als $F \subseteq \Sigma \times \Sigma^2$ geschrieben.

1.1 Formale Sprachen und Grammatiken

Allgemein ist eine formale Sprache L eine Menge von Zeichenketten über einem Alphabet Σ , formal $L \subseteq \Sigma^*$. Die Elemente dieser Mengen können durch **Grammatiken** gebildet werden, in perfekter Analogie zu den Grammatiken, die natürlich gesprochene Sprachen generieren (z.B. Satz = Subjekt-Verb-Objekt, Subjekt = Artikel-Adjektiv-Nomen, usw ...). Im folgenden wird unter "Sprache" immer eine formale Sprache (d.h. eine Menge) verstanden.

Definition 1

Eine Grammatik ist ein Vier-Tupel $G = (N, T, P, S)$, das die formale Sprache $L(G)$ über einem Alphabet Σ bildet, also $L(G) \subseteq \Sigma^*$. Die Elemente der Grammatik sind :

- N , eine nichtleere Menge von Variablen, auch **Nonterminale** genannt
- $T \subseteq \Sigma$, die nichtleere Menge der Zeichen oder **Terminale**, aus denen die Elemente der Sprache zusammengesetzt sind.
- $P \subseteq (N \cup T)^+ \times (N \cup T)^*$, die Menge der **Produktionen** (auch: Überführungsregeln), die angibt, welche Zeichen(folgen) durch welche anderen Zeichen(folgen) ersetzt werden können.
- $S \in N$, das **Startsymbol**, dies ist die Variable, von der ausgehend alle Produktionsfolgen beginnen

Formal gilt für die Produktionen P einer jeden Grammatik $G = (N, T, P, S) : P \subseteq (N \cup T)^+ \times (N \cup T)^*$. Oft schreibt man auch $P : (N \cup T)^+ \rightarrow (N \cup T)^*$. Einzelne Produktionen werden statt $(\alpha, \beta) \in P$ auch $\alpha \rightarrow \beta \in P$ geschrieben. Letztere Schreibweise erlaubt das Zusammenfassen von Produktionen; statt :

$$P = \{\alpha \rightarrow \beta, \alpha \rightarrow \beta', \alpha \rightarrow \beta'', \dots\}$$

schreibt man :

$$P = \{\alpha \rightarrow \beta \mid \beta' \mid \beta'' \mid \dots\}$$

Die Elemente von $L(G)$ werden ausgehend vom Startsymbol meist in mehreren Schritten, einzelnen Anwendungen der Produktionen, erzeugt. Die Zeichenfolgen, die dabei auftreten, $\alpha \in (N \cup T)^+$, werden *Satzformen* genannt. Der Übergang einer Satzform in eine andere wird mit \Rightarrow dargestellt, $\alpha \Rightarrow \alpha'$ gilt, wenn Umformen eines Teils von α gemäß der Produktionen zu α' führt. Formal:

$$\alpha \Rightarrow \alpha' \iff \alpha = \alpha_1 \alpha_2 \alpha_3 \wedge \alpha' = \alpha_1 \alpha'_2 \alpha_3 \wedge (\alpha_2, \alpha'_2) \in P$$

Die transitive Hülle von \Rightarrow wird mit \Rightarrow^+ dargestellt, die reflexive und transitive Hülle mit \Rightarrow^* . Die durch eine Grammatik erzeugte Sprache sind die Zeichenfolgen über dem Terminalphabet, die ausgehend vom Startsymbol durch mehrfache Anwendung von Produktionen abgeleitet werden können : $L(G) = \{w \mid w \in T^* \wedge S \Rightarrow^+ w\}$.

Beispiel 1

Eine mögliche Grammatik, um die Sprache der arithmetischen Ausdrücke zu erzeugen,

ist :

$$\begin{aligned}
 G &= (\{S, Z, N, A\}, \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, \div, (,)\}, P, S) \\
 P &= \{S \rightarrow N \mid S * S \mid S \div S \mid (S + S) \mid (S - S) \\
 &\quad N \rightarrow 0 \mid Z \mid ZN \\
 &\quad Z \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9\}
 \end{aligned}$$

Eine (von vielen möglichen) Ableitung des Wortes '14 * (2 + 1)':

$$\begin{aligned}
 S &\Rightarrow S * S \Rightarrow S * (S + S) \Rightarrow N * (S + S) \Rightarrow N * (N + S) \Rightarrow N * (N + N) \\
 &\Rightarrow ZN * (N + N) \Rightarrow ZZ * (N + N) \Rightarrow ZZ * (Z + N) \Rightarrow ZZ * (Z + Z) \\
 &\Rightarrow 1Z * (Z + Z) \Rightarrow 14 * (Z + Z) \Rightarrow 14 * (2 + Z) \Rightarrow 14 * (2 + 1)
 \end{aligned}$$

Zur Erinnerung : es werden (an sich bedeutungslose) Zeichenketten betrachtet; i.d.S. ist z.B. $3 * 4 \neq 4 * 3$.

1.1.1 Die Chomsky - Hierarchie

Durch stufenweises Einschränken der Produktionen auf bestimmte Teilmengen von $(N \cup T)^+ \times (N \cup T)^*$ erhält man eine Hierarchie von Grammatikklassen. Die Abstufung wird *Chomsky-Hierarchie* genannt, die einzelnen Klassen sind die :

- **Typ 0** - Grammatiken
Die Produktionen dieser Grammatiken haben die bereits angegebene, allgemeinste Form : $P \subseteq (N \cup T)^+ \times (N \cup T)^*$. Jede Grammatik ist vom Typ 0.
- **Typ 1** - Grammatiken
Bei diesen Grammatiken sind die Produktionen dahingehend eingeschränkt, daß eine Zeichenkette nicht verkürzt werden darf. Es gilt : $(\alpha, \beta) \in P \rightarrow |\alpha| \leq |\beta|$. Typ 1 - Grammatiken heißen auch *kontextsensitiv*.
- **Typ 2** - Sprachen
Die Einschränkung wird darauf ausgedehnt, daß nur aus einem einzelnen Nichtterminal abgeleitet werden darf : $P \subseteq N \times (N \cup T)^+$. Man nennt solche Grammatiken auch *kontextfrei*. Die in Beispiel 1 gegebene Grammatik fällt in diese Klasse.
- **Typ 3** - Sprachen
Hier wird zusätzlich die Zeichenfolge, auf die ein Nichtterminal abgeleitet werden kann, eingeschränkt. Es ist $P \subseteq N \times (T \cup TN)$. Jede Produktion überführt ein Nichtterminal entweder in ein Terminal oder in ein Terminal gefolgt von einem Nichtterminal. Solche Grammatiken heißen auch *regulär*.

Es wird sich zeigen, daß bestimmte Sprachen nicht durch jeden der Grammatiktypen erzeugbar sind. Die Einschränkungen der Produktionen beeinflussen die Ausdrucksmächtigkeit der Grammatiken. Analog den Typ i - Grammatiken unterteilt man auch die Sprachen in Typ i - Sprachen : Der Typ einer Sprache ist der größte Typ aller Grammatiken, die die Sprache erzeugen.

Die Grammatikklassen sind ineinander enthalten. Produktionen einer Typ i - Grammatik erfüllen auch die Bedingungen an Produktionen einer Grammatik kleineren Typs. Somit gilt ebenfalls für eine Typ i - Sprache, daß sie in den Klassen kleineren Typs enthalten ist.

Insgesamt ergibt sich für die Sprachklassen eine Kette echter Teilmengen :

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

Oft soll eine Sprache auch das leere Wort λ enthalten. Es soll also gelten : $S \Rightarrow^* \lambda$. Nach den beschriebenen Eigenschaften verschiedener Grammatiken müsste eine Sprache, die das leere Wort enthält, also zwangsläufig vom Typ 0 sein, da die Satzform während der Ableitung mindestens einmal verkürzt werden muß. Verkürzende Produktionen sind jedoch bereits für Typ 1 - Grammatiken nicht mehr möglich.

Man kann allerdings zeigen, daß

1. eine Grammatik $G = (N, T, P, S)$ mit $(X, \lambda) \in P$ für bel. $X \neq S$ umgeformt werden kann in eine Grammatik $G' = (N, T, P', S)$, die keine solchen Regeln mehr enthält, und $L(G) = L(G')$ gilt.
2. jede Typ 1,2,3 - Sprache L mit $\lambda \in L$ durch eine Grammatik erzeugt werden kann, in der nur eine verkürzende Produktion vorkommt, nämlich aus dem Startsymbol heraus.

Daher werden Produktionen der Form $X \rightarrow \lambda$ als einzige verkürzende Regeln in Typ 1,2,3 - Grammatiken zugelassen.

1.2 Reguläre Sprachen

Charakterisierung von TYP-3-Sprachen durch:

- Endliche Automaten
- Nichtdeterministische Automaten
- Reguläre Ausdrücke

1.2.1 Endliche Automaten

Definition 2 1. Ein **deterministischer endlicher Automat** (kurz: DEA; bzw. DFA für 'deterministic finite automaton') ist ein 5-Tupel $M = \{Z, \Sigma, \delta, z_0, E\}$ vermöge:

- Z ist eine nichtleere endliche Menge von Zuständen
 - Σ ist ein Alphabet von Eingabezeichen mit $\Sigma \neq \emptyset$
 - δ ist die Überföhrungsfunktion $\delta : Z \times \Sigma \rightarrow Z$
 - $z_0 \in Z$ ist der Startzustand
 - $E \subseteq Z$ ist die Menge der Endzustände
2. Zu $M = (Z, \Sigma, \delta, z_0, E)$ konstruieren wir die **Abbildung** $\hat{\delta} : Z \times \Sigma^* \rightarrow Z$ induktiv durch
- $\hat{\delta}(z, \lambda) := z \quad \forall z \in Z$
 - $\hat{\delta}(z, ax) := \hat{\delta}(\delta(z, a)x) \quad \text{für } z \in Z, a \in \Sigma, x \in \Sigma^*$
(äquivalent: $\hat{\delta}(z, ax) := \delta(\hat{\delta}(z, x)a)$)
3. Die von einem endlichen Automaten $M = (Z, \Sigma, \delta, z_0, E)$ **akzeptierte Sprache** ist $L(M) = T(M) = \{x \in \Sigma^* \mid \hat{\delta}(z_0, x) \in E\}$

Beispiel 2

$$Z = \{z_0, z_1, z_2\}, \Sigma = \{1, 0\}, E = \{z_2\}$$

Wir stellen uns die Aufgabe, Binärzahlen daraufhin zu überprüfen, ob sie bei ganzzahliger Division durch 3 den Rest 2 aufweisen. Die Bits werden in "natürlicher Reihenfolge von links nach rechts" gelesen, also das höchstwertige Bit zuerst. Die drei Zustände des Automaten entsprechen den Restklassen modulo 3.

Die Überföhrungsfkt. wird hier als Tabelle dargestellt, Zeilen- und Spaltennamen stellen die Argumente von δ , der Eintrag den Funktionswert dar.

δ	z_0	z_1	z_2
0	z_0	z_2	z_1
1	z_1	z_0	z_2

Die Konstruktion beruht auf folgender Beobachtung: wenn wir eine Folge von Bits gelesen haben, die einer Zahl n entsprechen und wir lesen das Eingabezeichen 0, so entspricht nunmehr die gesamte gelesene Bitfolge der Zahl $2n$. Lesen wir dagegen eine 1, so entspricht dies jetzt der Zahl $2n + 1$. Waren wir beispielsweise in dem Zustand "Rest 2" und lesen eine 1 so müssen wir in dem Zustand $2 * 2 + 1$ modulo 3 landen, bleiben also in "Rest 2".

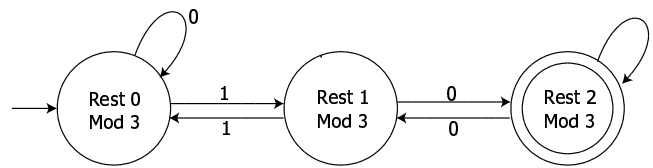


Abbildung 1: Mod 3 - Automat

Fragen:

- Warum ist "Rest 0" der Startzustand?
- Was bewirken führende Nullen?

In den folgenden Sätzen betrachten wir äquivalente Modelle. Zunächst:
DEA = TYP-3-Sprache

Satz 1

Zu jedem DEA M gibt es eine TYP-3-Grammatik G mit $T(M) = L(G)$

Beweis 1

Zu $M = (Z, \Sigma, \delta, z_0, E)$ konstruiere $G = (Z, \Sigma, P, z_0)$ mit:

$$P := \{z_i \rightarrow az_j \mid \delta(z_i, a) = z_j\} \cup \{z_i \rightarrow a \mid \delta(z_i, a) \in E\} \cup \{z_0 \rightarrow \lambda \mid z_0 \in E\}$$

□

Die konstruierte Typ-3-Grammatik simuliert den DEA, indem sie für jeden Zustand ein Nonterminal bereitstellt. Startsymbol ist das dem Startzustand zugeordnete Nonterminal. Der Übergang von einem Zustand z_i in einen Zustand z_j mit Lesen eines Eingabesymbols a wird durch die Produktion $z_i \rightarrow az_j$ dargestellt. Sollte z_j ein Endzustand sein, so ist das bislang gelesene Eingabewort ein Element der von M akzeptierten Sprache. G enthält daher in diesem Fall noch zusätzlich die terminierende Regel $z_i \rightarrow a$.

Beispiel 3

Die Anwendung von Satz 1 auf Beispiel 2 ergibt folgende TYP-3-Grammatik :
 $G = (\{S, Z, E\}, \{0, 1\}, P, S)$, mit den Produktionen:

$$\begin{aligned} S &\rightarrow 0S \mid 1E \\ E &\rightarrow 1S \mid 0Z \mid 0 \\ Z &\rightarrow 1Z \mid 0E \mid 1 \end{aligned}$$

1.2.2 Nichtdeterministische endliche Automaten

Häufig sind wir im Alltag mit nichtdeterministischen Anweisungen der Art "Zwei Kilometer vor der letzten Ampel links abbiegen!" konfrontiert. Diese stellen eigentlich keine Anweisungen, sondern Spezifikationen dar. Sie helfen also, eine Aufgabe zu beschreiben, stellen jedoch im eigentlichen Sinne keine Lösung dar. Dementsprechend gibt es das Konzept des nichtdeterministischen endlichen Automaten, der auf einem Eingabewort mehrere Verhaltensmöglichkeiten aufweisen kann. Ein Wort gilt genau dann als Element der Sprache des Automaten, wenn es eine akzeptierende Berechnung des Automaten auf diesem Wort gibt.

Definition 3 1. Ein nichtdeterministischer, endlicher Automat (kurz: NEA oder NFA) ist ein 5-Tupel $M = (Z, \Sigma, \delta, S, E)$ mit:

- Z ist eine endliche Menge von Zuständen
- Σ ist eine zu Z disjunkte endliche Menge von Eingabezeichen
- $\delta : Z \times \Sigma \rightarrow \mathcal{P}(Z)$ ist Überföhrungsfunktion
- $S \subseteq Z$ ist Menge der Startzustände
- $E \subseteq Z$ ist Menge der Endzustände

2. Zu einem NEA $M = (Z, \Sigma, \delta, S, E)$ definieren wir $\hat{\delta} : \mathcal{P}(Z) \times \Sigma^* \rightarrow \mathcal{P}(Z)$ wieder induktiv durch

- $\hat{\delta}(Y, \lambda) := Y$
- $\hat{\delta}(Y, aw) := \bigcup_{z \in Y} \hat{\delta}(\delta(z, a)w)$

also $(z, \lambda, z') \in \hat{\delta} \iff z' \in \delta(z, \lambda)$ und

$(z, ax, z') \in \hat{\delta} \iff \exists z'' \in Z$ mit
 $(z, a, z'') \in \delta$ und $(z'', x, z') \in \hat{\delta}$ für $z, z' \in Z, a \in \Sigma, x \in \Sigma^*$

Für (z, x, z') schreiben wir $z \vdash^x z'$, für $z, z' \in Z$ und $x \in \Sigma^*$.

3. Die von einem NEA $M = (Z, \Sigma, \delta, S, E)$ akzeptierte Sprache ist:

$$T(M) := \{x \in \Sigma^* \mid \delta(S, x) \cap E \neq \emptyset\}$$

Beispiel 4

Der Nichtdeterminismus in der Anweisung ‘‘Zwei Kilometer vor der letzten Ampel links abbiegen!’’ entspricht in diesem Beispiel der Aufgabe, ein Binärwort darauf zu prüfen, ob das k -letzte Zeichen vor Ende der Eingabe eine Null ist. Während ein DEA in einem Puffer die letzten k Zeichen speichern müßte, was größenordnungsmäßig 2^k Zustände erfordert, kann der folgende NEA ‘‘raten’’, ob das folgende Zeichen das k -letzte ist.

Für festes $k > 0$ wird $L_k := \{x \in \{0, 1\}^* \mid \text{das } k\text{-te Zeichen von rechts in } x \text{ ist } 0\}$ von folgendem Automaten erkannt:

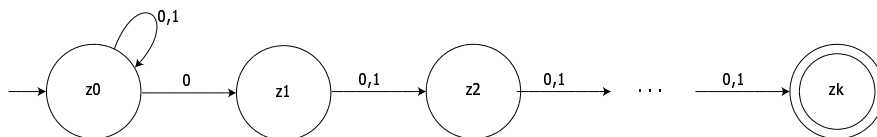


Abbildung 2: k -letzte Element ist 0 Automat

Dieses Beispiel macht die Vorteile deutlich, die ein NEA gegenüber einem DEA hat, und man könnte sich vorstellen, dass es Mengen gibt, die zu ihrer Beschreibung durch endliche Automaten Nichtdeterminismus benötigen. Der folgende Satz beweist, dass das erstaunlicherweise nicht der Fall ist, daß also deterministische und nichtdeterministische endliche Automaten die gleiche Beschreibungsfähigkeit aufweisen.

Satz 2

(von Rabin und Scott:)

Jede von einem NEA akzeptierte Sprache ist auch von einem DEA akzeptierbar.

Die Konstruktion beruht auf der Endlichkeit der Zustände von M : zu jedem Zeitpunkt der Simulation von M hat M' die Menge der "derzeit aktiven" Zustände von M gespeichert. Bei Beginn der Berechnung ist dies gerade die Menge der Startzustände S . Das Lesen eines Eingabesymbols führt nun dazu, dass wir für jeden derzeit aktiven Zustand, der nun deaktiviert wird, jeden bei dieser Eingabe möglichen Folgezustand aktivieren. Sollte es geschehen, dass ein Zustand dabei auf mehrere Weisen aktiviert wird, so braucht dies nur einmal geschehen. Die Zustände von M' sind also Mengen von Zuständen von M . Daher hat diese Konstruktion auch ihren Namen: Potenz(mengen)konstruktion. Da ein Wort dann in $L(M)$ ist, wenn es eine akzeptierende Berechnung gibt, möglicherweise neben vielen ablehnenden, akzeptiert M' genau dann, wenn unter den derzeit aktiven Zuständen von M mindestens ein Enzustand ist. Die Konstruktion sieht demgemäß folgendermaßen aus:

Beweis

Zu gegebenem NEA $M = (Z, \Sigma, \delta, S, E)$ konstruiere einen DEA $M' = (Z', \Sigma, \delta', z_0, E')$ vermöge:

$$Z' = \mathcal{P}(Z)$$

$$\delta'(z', a) = \bigcup_{z \in z'} \delta(z, a) = \hat{\delta}(z', a)$$

$$z_0 = S$$

$$E' = \{T \subseteq Z \mid T \cap E \neq \emptyset\}$$

Wir zeigen nun: $L(M) = L(M')$

Für $x = a_1 a_2 \dots a_n \in \Sigma^*$ gilt:

$$\begin{aligned} x \in T(M) &\iff \hat{\delta}(S, x) \cap E \neq \emptyset \\ &\iff \exists Z_1, Z_2, \dots, Z_n \subseteq Z \text{ mit :} \\ &\quad \hat{\delta}(S, a_1) = Z_1, \hat{\delta}(Z_1, a_2) = Z_2, \dots, \hat{\delta}(Z_{n-1}, a_n) = z_n \text{ und} \\ &\quad Z_n \cap E \neq \emptyset \\ &\iff \hat{\delta}(z_0, x) \in E' \\ &\iff x \in T(M') \quad \square \end{aligned}$$

Beispiel 5

Fortsetzung von Beispiel 4 für $k = 2$

DFA für L_1 :	δ'	0	1
	\emptyset	\emptyset	\emptyset
	z_0	z_1	z_0
	z_1	z_2	z_2
	z_2	z_2	z_0

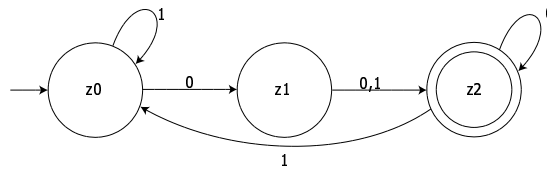


Abbildung 3:

Bemerkung

- Ein Automat kann viele überflüssige Zustände enthalten.
- Ein Automat ist i.a. nicht minimal.
- Es gibt Beispiele, in denen der DEA bei der Konstruktion 2^n Zustände hat.

Satz 3

Für jede reguläre Grammatik G gibt es einen NEA, mit $L(G) = T(M)$

Beweis

Zu einer regulären Grammatik $G = (V, T, P, S)$ definiere einen NEA $M = (Z, \Sigma, \delta, S', E)$ vermöge:

$$Z = V \cup \{F\}$$

$$S' = S$$

$$E = \begin{cases} \{S, F\} & \text{falls } S \rightarrow \lambda \in P \\ \{F\} & \text{sonst} \end{cases}$$

$$\delta(A, a) \ni B, \text{ falls } A \rightarrow aB \in P$$

$$\delta(A, a) \ni F, \text{ falls } a \rightarrow a \in P$$

In Umkehrung der entsprechenden Konstruktion in Satz 1 wird hierbei eine Regel der Form $A \rightarrow aB$ umgeformt in $B \in \delta(A, a)$, wobei die Nonterminale als Zustände des konstruierten Automaten dienen. Eine terminierende Regel $A \rightarrow a$ wird simuliert durch eine Transition $F \in \delta(a, A)$, wobei F ein neuer (End)zustand ist. Der Nichtdeterminismus des konstruierten Automaten dient also u.a. dazu, das Ende des gelesenen Wortes zu raten. Liegt das leere Wort λ in $L(G)$ so ist aus S ein Endzustand. \square

Der obige Satz zeigt zusammen mit den Sätzen 1 und 2 die formalsprachliche Äquivalenz von deterministischen und nichtdeterministischen endlichen Automaten und regulären Grammatiken.

1.2.3 Reguläre Ausdrücke

Definition 4 - \emptyset ist ein regulärer Ausdruck

- λ ist ein regulärer Ausdruck
- Für jedes $a \in \Sigma$ ist a ein regulärer Ausdruck

- Sind α und β reguläre Ausdrücke, so auch $\alpha\beta$, $(\alpha|\beta)$ und $(\alpha)^*$

Induktiv über seinem Aufbau wird jedem regulären Ausdruck eine Sprache $L(\gamma) \subseteq \Sigma^*$ zugeordnet:

$$\begin{array}{ll} -L(\emptyset) := \emptyset & -L(\alpha\beta) := L(\alpha) * L(\beta) = \{ab | a \in L(\alpha) b \in L(\beta)\} \\ -L(\lambda) := \{\lambda\} & -L((\alpha|\beta)) := L(\alpha) \cup L(\beta) \\ -L(a) := \{a\} \text{ für } a \in \Sigma & -L((\alpha)^*) := L(\alpha)^* \end{array}$$

Beispiel 6

Die Menge aller ungeraden Binärzahlen wird durch $(0|1)^*1$ beschrieben. Sollen keine führenden Nullen zugelassen sein, ergibt sich der Ausdruck $1(0|1)^*1$.

Satz von Kleene

Die Menge der durch reguläre Ausdrücke beschreibbaren Sprachen stimmt genau mit der der TYP-3-Sprachen überein.

Beweis Teil I($RA \rightarrow NFA$)

Wir konstruieren zu jedem regulären Ausdruck R eine reguläre Grammatik, die dieselbe Sprache erzeugt, oder einen äquivalenten endlichen Automaten M . Wir gehen dabei induktiv über die Termstruktur von R vor – behandeln also zunächst den Fall daß R atomar ist (Induktionsverankerung), dann daß R zusammengesetzt ist. In letzterem Fall können wir davon ausgehen, daß für die Teilausdrücke, aus denen R aufgebaut ist, schon äquivalente Automaten konstruiert worden sind (Induktionsbehauptung). Da das Zusammensetzen von regulären Ausdrücken mittels der Operationen Konkatenation, Vereinigung und Sternbildung geschieht, genügt es, in Schritt 2 zu zeigen, daß die von nichtdeterministischen endlichen Automaten beschriebenen Sprachen unter diesen drei Operationen abgeschlossen sind (Induktionsschluss).

Schritt 1

Die atomaren regulären Ausdrücke beschreiben TYP 3 - Sprachen:

- $L(\emptyset) = \emptyset$ wird durch folgenden DEA erkannt:
 $M_\emptyset = (\{z_0\}, \Sigma, \{\}, z_0, \{\})$, hier ist δ für alle möglichen Argumentpaare undefiniert.
- $L(\lambda) = \{\lambda\}$ durch:
 $M_\lambda = (\{z_0\}, \Sigma, \{\}, z_0, \{z_0\})$, dito.
- $L(a) = \{a\}$ durch:
 $M_a = (\{z_0, z_1\}, \Sigma, \delta, z_0, \{z_1\})$, mit $\delta(z_0, a) = z_1$ als einzig definiertem Übergang.

Schritt 2

Gezeigt wird der Abschluss TYP 3 - Sprachen unter Konkatenation, Vereinigung und Kleene'scher Hüllenbildung. Seien $M_1 = (Z_1, \Sigma, \delta_1, S_1, E_1)$ und $M_2 = (Z_2, \Sigma, \delta_2, S_2, E_2)$ zwei NFAs mit $Z_1 \cap Z_2 = \emptyset$

Dann gilt für:

- $L(M_1) \cdot L(M_2)$

$M_3 := ((Z_1 \setminus E_1) \cup Z_2, \Sigma, \delta, S, E_2)$ mit :

$$S = \begin{cases} S_1 & \text{für } \lambda \notin T(M_1) \\ S_1 \cup S_2 & \text{sonst} \end{cases}$$

$$\delta(z, a) := \begin{cases} \delta_1(z, a) & \text{für } z \in Z_1 \setminus E_1, \delta_1(z, a) \cap E_1 = \emptyset \\ (\delta_1(z, a) \setminus E_1) \cup S_2 & \text{für } z \in Z_1 \setminus E_1, \delta_1(z, a) \cap E_1 \neq \emptyset \\ \delta_2(z, a) & \text{für } z \in Z_2 \end{cases}$$

$$T(M_3) = T(M_1) \cdot T(M_2) \text{ und } \delta(E_2, \Sigma) = \emptyset$$

- $L(M_1) \cup L(M_2)$

$M_4 := (Z_1 \cup Z_2, \Sigma, \delta, S_1 \cup S_2, E_1 \cup E_2)$ mit:

$$\delta(z, a) := \begin{cases} \delta_1(z, a) & \text{für } z \in Z_1 \\ \delta_2(z, a) & \text{für } z \in Z_2 \end{cases}$$

$$T(M_4) = T(M_1) \cup T(M_2) \text{ und } \delta(E_1 \cup E_2, \Sigma) = \emptyset$$

- $M_5 := (Z_1 \cup \{z_0\}, \Sigma, \delta, S_1 \cup \{z_0\}, E_1 \cup \{z_0\})$ mit:

$$\delta(z, a) \begin{cases} \delta_1(z, a) & \text{für } z \in Z_1, \delta_1(z, a) \cap E_1 = \emptyset \\ \delta_1(z, a) \cup S_1 & \text{für } z \in Z_1, \delta_1(z, a) \cap E_1 \neq \emptyset \\ \emptyset & \text{für } z = z_0 \end{cases}$$

$$T(M_5) = T(M_1)^* \text{ und } \delta(E_1 \cup \{z_0\}, \Sigma) = \emptyset$$

Beweis Teil II ("DFA \rightarrow RA ")

Zu einem DFA $M = (Z, \Sigma, \delta, z_1, E)$ konstruieren wir einen RA γ mit:

$$L(\gamma) = T(M)$$

Der Beweis der Rückrichtung ist etwas umfangreicher, da der endliche Automat, zu dem ein äquivalenter Ausdruck konstruiert werden soll, über keine Termstruktur verfügt und daher sich kein direkter Induktionsbeweis anbietet. Die Beweisidee ist etwa wie folgt: Die von einem endlichen Automaten akzeptierte Sprache besteht genau aus den Wörtern, die vom Startzustand in einen Endzustand führen. Wir versuchen daher, zu jedem Paar z_i, z_j von Zuständen die Menge der Wörter, die von z_i nach z_j führen, durch einen regulären Ausdruck darzustellen. Dazu "kontrollieren" wir diese Menge dadurch, dass wir die Menge der auf dem Weg von z_i nach z_j besuchten Zustände einschränken. Es wird also nicht die Länge des Weges kontrolliert, sondern in gewisser Hinsicht seine Reichhaltigkeit. Lassen wir auf dem Weg keine Zwischenzustände zu, so ist die Menge der erlaubten Wörter so stark eingeschränkt, dass die sich ergebende Wortmenge auf einfache Weise durch einen regulären Ausdruck dargestellt werden kann. Die wesentliche Beweisidee der folgenden Konstruktion ist nun, nach und nach mehr Zwischenzustände zuzulassen und die Vergrößerung der sich ergebenden Wortmenge durch die regulären Operationen Konkatenation, Vereinigung und Sternbildung zu beschreiben.

Es sei $n := |Z|$ und Z sei angeordnet: $Z = \{z_1, z_2, \dots, z_n\}$. Für $i, j \in \{1, \dots, n\}$ und $k \in \{0, 1, \dots, n\}$ betrachte die Menge

$$R_{ij}^k := \{x \in \Sigma^* \mid \hat{\delta}(z_i, x) = z_j \wedge \forall x = yz, y \neq \lambda, z \neq \lambda, \hat{\delta}(z_i, y) \in \{z_1, z_2, \dots, z_k\}\}$$

Es gilt $R_{ij}^n = \{x \in \Sigma^* \mid \hat{\delta}(z_i, x) = z_j\}$ und daher $T(M) = \bigcup_{\{i \mid z_i \in E\}} R_{1i}^n$

Es reicht zu zeigen: Jedes $R_{i,j}^K$ ist regulär. Beweis durch Induktion über K .

Für $i \neq j$ gilt $R_{ij}^0 = \{a \in \Sigma \mid \delta(z_i, a) = z_j\}$ und $R_{ii}^0 = \{a \in \Sigma \mid \delta(z_i, a) = z_i\} \cup \{\lambda\}$
Die $R_{i,j}^0$ sind offenbar regulär.

Betrachte ein $x = a_1 a_2 \dots a_m \in R_{ij}^K$. Bei der Verarbeitung des Wortes x können jetzt zwei Fälle unterschieden werden:

Fall 1: Der zusätzlich erlaubte Zustand z_k wird gar nicht benutzt:

$$\begin{aligned} \text{also } x &\in R_{ij}^{k-1} \\ \hat{\delta}(z_i, a_1) &\in \{z_1, z_2, \dots, z_{k-1}\} \\ \hat{\delta}(z_i, a_1 a_2) &\in \{z_1, z_2, \dots, z_{k-1}\} \\ &\vdots \\ \hat{\delta}(z_i, a_1 a_2 \dots a_{m-1}) &\in \{z_1, z_2, \dots, z_{k-1}\} \\ v_i \in R_{kk}^{k-1}, v_1, v_2, \dots, v_{t-1} &\in R_{kk}^{k-1} \text{ und } v_t \in R_{kj}^{k-1} \end{aligned}$$

Fall 2: Der Zustand z_k wird beim Lesen von x mindestens einmal angesprungen.

$x \notin R_{ij}^{k-1}$ Also kommt z_k vor!
Markiere genau die Vorkommen von z_k

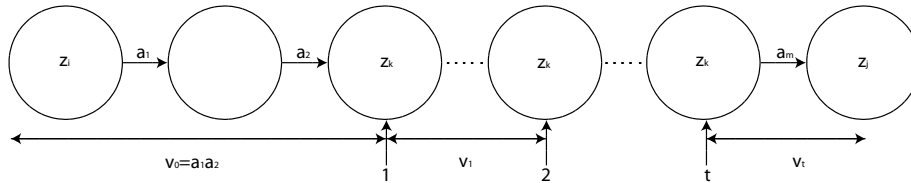


Abbildung 4:

Wenn wir x genau an den t Stellen zerschneiden, an denen der Zustand z_k eingenommen wird, so erhalten wir dadurch eine Zerlegung $x = v_0 v_1 \dots v_t$ mit folgenden Eigenschaften:

Ist also R_{ij}^{k-1} durch ein α_{ij}^{k-1} beschreibbar, so gilt für $\alpha_{ij}^k := (\alpha_{ik}^{k-1} \alpha_{kk}^{k-1} \alpha_{kj}^{k-1}) | \alpha_{ij}^{k-1}$
Ist $E = \{Z_{i_1}, Z_{i_2}, \dots, Z_{i_m}\}$ so gilt: $T(M) = R_{1i_1}^n \cup R_{1i_2}^n \cup \dots \cup R_{1i_m}^n$

Beispiel: Sei $L := b^* a \{a, b\}^*$ die Menge aller Wörter über a und b , die mindestens ein a enthalten. Der (kleinste) deterministische endliche Automat hat dann folgende Gestalt:

Die Mengen R_{ij}^k ergeben sich dann (nach Kürzungen!) zu:

$k = 0$	$j = 1$	$j = 2$
$i = 1$	λ, b	a
$i = 2$	$-$	λ, a, b
$k = 2$	$j = 1$	$j = 2$
$i = 1$	b^*	$b^*a\{a, b\}^*$
$i = 2$	$-$	$\{a, b\}^*$

$k = 01$	$j = 1$	$j = 2$
$i = 1$	b^*	b^*a
$i = 2$	$-$	λ, a, b

1.2.4 Das Pumping Lemma

Wir haben in den vorangehenden Abschnitten verschiedene Beschreibungsmechanismen für formale Sprachen kennen gelernt, die sich alle als äquivalent herausstellten und die Menge der regulären oder Typ-3 Sprachen beschreiben. In diesem Abschnitt werden nun Grenzen dieses Ansatzes herausgearbeitet. Der folgende Satz wird es gestatten, für gewisse Sprachen zu zeigen, daß sie nicht regulär und damit nicht durch endliche Automaten beschreibbar sind.

Satz 4

Zu jeder regulären Sprache $L \subseteq \Sigma^*$ existiert eine Zahl $n \in \mathbb{N}$ derart, daß sich alle $z \in L$ der Länge $|z| \geq n$ zerlegen lassen in $z = uvw$ für gewisse $u, v, w \in \Sigma^*$, so daß gilt :

1. $|v| \neq \lambda$,
2. $|uv| \leq n$,
3. für alle $i \in \mathbb{N}$ ist $uv^i w \in L$.

Beweis

Der Beweis macht sich die Endlichkeit von Z zunutze. Wenn es n verschiedene Zustände gibt, so muss bei Eingabe eines Wortes z , welches länger als n ist, ein Zustand z' mehrfach besucht werden. Der Automat enthält also einen Zyklus (vgl Abb. 5). Somit lässt sich z zerlegen in drei Teilwörter u, v, w , also $z = uvw$, wobei u vom Startzustand zum Zustand z' führt und v von z' wiederum zu z' . Wird z nun akzeptiert von M , führt also w von z' zu einem Endzustand, so führt das Weglassen oder Neueinfügen des Teilwortes v (das in dem Zyklus gelesen wird) zur Konstruktion weiterer Wörter, die von M akzeptiert werden.

Es sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DEA mit $T(M) = L$. Wir setzen $n := |Z|$. Es sei z ein beliebiges Element von L mit $|z| \geq n$.

Also $z = a_1 a_2 \dots a_n z'$ mit $a_i \in \Sigma$ und $z' \in \Sigma^*$.

Für $i = 0, 1, \dots, n$ setze $q_i := \hat{\delta}(z_0, a_1 \dots a_i)$.

Da $|Z| = n$, existieren $0 \leq i < j \leq n$ mit $q_i = q_j$.

Wir setzen $u := a_1 a_2 \dots a_i$, $v := a_{i+1} \dots a_j$ und $w := a_{j+1} \dots a_n z'$.

Dann gilt für $q := \hat{\delta}(z_0, u) : \hat{\delta}(z_0, v) = q$. Also $\hat{\delta}(z_0, u) = \hat{\delta}(z_0, uv) = q$.

Und induktiv über $i : \hat{\delta}(z_0, uv^i) = q$. Also gilt auch für alle $i \in \mathbb{N}$ $\hat{\delta}(z_0, uv^i w) = \hat{\delta}(\hat{\delta}(z_0, uv^i) w) = \hat{\delta}(\hat{\delta}(z_0, uv) w) = \hat{\delta}(z_0, z) \in E$. Mithin $uv^i w \in L$. \square

Beispiel 7

$L_1 := \{a^n b^n \mid n \geq 1\}$ ist nicht regulär

Indirekter Beweis:

Wäre L_1 regulär, so gäbe es ein n mit den Eigenschaften des obigen Satzes. Betrachte

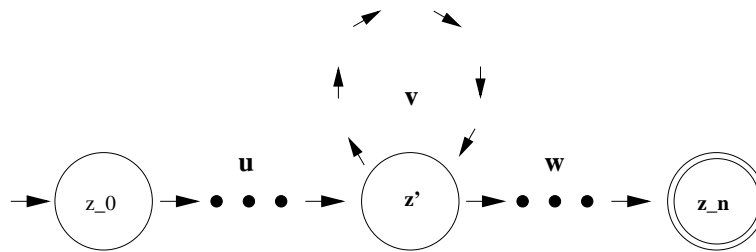


Abbildung 5: Lesen von $z = uvw$

$z := a^n b^n$. Da $|z| = 2n \geq n$ ist würde es $u, v, w \in \{a, b\}^*$ geben mit $z = uvw$ und $uv^i w \in L_1$. Da $|uv| \leq n$ und $v \neq \lambda$ gilt $v \in a^+$. Dann gilt $uw = a^{n-|v|} b^n \notin L_1$ im Widerspruch zur Aussage des Satzes. \square

Beispiel 8

$L_2 := \{0^{2^n} \mid n \geq 0\}$ ist nicht regulär

Indirekter Beweis:

Wäre L_2 regulär, so gäbe es ein n mit den Eigenschaften des obigen Satzes. Betrachte $z := 0^{2^m}$, wobei m die kleinste natürliche Zahl mit $2^m > n$ sei. Da $|z| \geq n$ ist würde es $u, v, w \in 0^*$ geben mit $z = uvw$ und $uv^i w \in L_2$. Da $|uv| \leq n$ und $|v| \geq 1$ gilt $2^m < 2^m + |v| \leq 2^m + n < 2^{m+1}$, also $uvv^i w = 0^{2^m + |v|} \notin L_2$ im Widerspruch zur Aussage des Satzes. \square

1.2.5 Der Satz von Myhill-Nerode und endliche Automaten

Eine Alternative zum im Gebrauch etwas schwerfälligen Pumping-Lemma besteht in der Charakterisierung der regulären Sprachen von Myhill und Nerode, die in diesem Unterabschnitt behandelt werden soll. Sie wird auch der Ausgangspunkt sein für ein Verfahren, das es uns gestattet wird, deterministische endliche Automaten zu minimieren und auf Äquivalenz zu testen.

Definition 5

Zu jedem $L \subseteq \Sigma^*$ definieren wir die Äquivalenzrelation R_L (auch \sim_L) vermöge

$$xR_L y \Leftrightarrow [\forall z \in \Sigma^* : xz \in L \Leftrightarrow yz \in L]$$

Satz von Myhill-Nerode

$L \subseteq \Sigma^*$ ist regulär genau dann, wenn der Index (d.h. die Anzahl der Äquivalenzklassen) von R_L endlich ist.

Beweis:

" \Rightarrow "

Sei L regulär, dann wird L von einem DEA M erkannt. Aus M konstruieren wir eine Relation R_M auf Σ^* dadurch, daß wir alle Zeichenketten, die M in denselben Zustand überführen, als derselben Klasse zugehörig ansehen.

Sei $M = (Z, \Sigma, \delta, z_0, E)$. Setze für $x, y \in \Sigma^* : xR_M y \Leftrightarrow \hat{\delta}(z_0, x) = \hat{\delta}(z_0, y)$.

Zunächst halten wir fest, daß R_M Äquivalenzrelation ist, also reflexiv, symmetrisch und transitiv. Dies ergibt sich aus den selben Eigenschaften der Gleichheit (von erreichten Zuständen).

Sei nun $xR_M y$. Nach Def. von R_M gilt $\hat{\delta}(z_0, x) = \hat{\delta}(z_0, y)$, also ergibt sich :

$$\begin{aligned}xz \in L &\Leftrightarrow \hat{\delta}(z_0, xz) \in E \\ &\Leftrightarrow \hat{\delta}(\hat{\delta}(z_0, x), z) \in E \\ &\Leftrightarrow \hat{\delta}(\hat{\delta}(z_0, y), z) \in E \\ &\Leftrightarrow \hat{\delta}(z_0, yz) \in E \\ &\Leftrightarrow yz \in L\end{aligned}$$

Somit ist R_M eine Verfeinerung der Relation R_L , d.h. daß jede Äquivalenzklasse von R_M vollständig in einer der Äquivalenzklassen von R_L enthalten ist. Da R_M endlich viele Äquivalenzklassen bildet (so viele, wie der (endliche !) Automat Zustände hat), kann auch R_L nur endlich viele Äquivalenzklassen haben, d.h. der Index von R_L ist endlich.

" \Leftarrow "

Hier konstruieren wir aus R_L einen DEA M , indem wir die Äquivalenzklassen von R_L als Zustände identifizieren und die Überföhrungsfunktion δ geeignet bilden.

Sei R_L Äquivalenzrelation auf Σ^* mit endlichem Index und der Menge der Äquivalenzklassen $\Sigma^*/R_L = \{[x_1]_{R_L}, [x_2]_{R_L}, \dots, [x_n]_{R_L}\}$.

Definiere $M = (Z, \Sigma, \delta, z_0, E)$ vermöge :

- $Z = \Sigma^*/R_L$
- $\delta([x]_{R_L}, a) = [xa]_{R_L}$
- $z_0 = [\lambda]_{R_L}$
- $E = \{[x]_{R_L} \mid x \in L\}$

Dann gilt

$$\begin{aligned}x \in T(M) &\Leftrightarrow \hat{\delta}(z_0, x) \in E \\ &\Leftrightarrow \hat{\delta}([\lambda]_{R_L}, x) \in E \\ &\Leftrightarrow [x]_{R_L} \in E \\ &\Leftrightarrow x \in L\end{aligned}$$

Da die Sprache L von einem DEA akzeptiert wird, ist sie regulär. \square

Für eine reguläre Sprache L ist der akzeptierende DEA i.A. nicht eindeutig festgelegt. Insbesondere ergeben sich Automaten mit unterschiedlich vielen Zuständen. Unter allen DEAs, die L akzeptieren, gibt es jedoch einen (bis auf Umbenennung der Zustände) eindeutigen DEA mit minimaler Zustandmenge, den sog. Minimalautomaten. Minimalautomaten stellen somit eine Normalform für DEAs dar.

Definition 6

Sei L eine reguläre Sprache, die von einem DEA $M = (Z, \Sigma, \delta, z_0, E)$ akzeptiert wird.

- Zwei Zustände $z, z' \in Z$ heißen **äquivalent**, $z \sim_M z'$, wenn gilt:

$$\forall x \in \Sigma^* : \hat{\delta}(z, x) \in E \Leftrightarrow \hat{\delta}(z', x) \in E$$

- M heißt **reduziert**, wenn M keine äquivalenten Zustände enthält.

Ein Algorithmus zur Bestimmung von \sim_M und damit der Konstruktion eines reduzierten Automaten :

- $R : Z \times Z \longrightarrow \{0, 1\}$
- $R ::= 1$
- Für $(z, z') \in E \times (Z \setminus E) \cup (Z \setminus E) \times E$ setze $R(z, z') := 0$
- Setze $R(z, z') := 0$ wenn es ein $a \in \Sigma$ mit $R(\delta(z, a), \delta(z', a)) = 0$ gibt, solange, bis sich keine Änderungen mehr ergeben.

Danach gilt: $z \sim_M z' \Leftrightarrow R(z, z') = 1!$. Der reduzierte Automat wird gebildet, indem man äquivalente Zustände zu einem einzigen zusammenfasst.

Definition 7

Sei wieder L Sprache, M DEA mit $L = T(M)$. Dann heißt M **Minimalautomat** für L, wenn M die kleinstmögliche Zustandsmenge in der Klasse aller DEAs, die L akzeptieren, hat.

Satz 5

Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DEA. Dann gilt : M ist minimal \Leftrightarrow M ist reduziert

Beweis:

" \Rightarrow "

Sei M minimal. Angenommen es gibt zwei äquivalente Zustände in M, so können diese nach obigem Algorithmus zu einem einzelnen zusammengefasst werden, wobei für den dadurch entstehenden DEA M' gilt : $T(M) = T(M')$. Da M' einen Zustand weniger als M hat, ist M nicht minimal. Also muß M reduziert sein.

" \Leftarrow "

Sei für einen DEA $M = (Z, \Sigma, \delta, z_0, E)$ wieder die Relation R_M definiert : $\forall x, y \in \Sigma^* : xR_M y \Leftrightarrow \hat{\delta}(z_0, x) = \hat{\delta}(z_0, y)$. Es gilt, daß für jeden Automaten, der L erkennt, R_M eine Verfeinerung von R_L ist, also $R_M \subseteq R_L$. Somit hat R_L die geringste Anzahl Äquivalenzklassen in Σ^* bzw der zugehörige DEA (wie im Beweis des Satzes von Myhill-Nerode) minimale Zustandsmenge.

Wir wollen zeigen, daß für einen reduzierten DEA M' gilt : $R_{M'} = R_L$.

$R_{M'} \subseteq R_L$ gilt bereits, zu zeigen ist $R_L \subseteq R_{M'}$ bzw. $\forall x, y \in \Sigma^* : xR_L y \Rightarrow xR_{M'} y$. Der Beweis folgt durch Kontraposition, wir betrachten Worte x,y, die einen reduzierten DEA in nicht-äquivalente Zustände überführen. Sei $M' = (Z', \Sigma, \delta', z'_0, E')$ reduzierter DEA, $x, y, z \in \Sigma^*$:

$$\begin{aligned}
\neg x R_{M'} y &\Leftrightarrow \neg(\forall z : \hat{\delta}'(\hat{\delta}'(z'_0, x), z) \in E' \Leftrightarrow \hat{\delta}'(\hat{\delta}'(z'_0, x), z)) \in E' \\
&\Leftrightarrow \exists z : \hat{\delta}'(\hat{\delta}'(z'_0, x), z) \in E' \Leftrightarrow \hat{\delta}'(\hat{\delta}'(z'_0, x), z) \notin E' \\
&\Leftrightarrow \exists z : xz \in T(M') \Leftrightarrow yz \notin T(M') \\
&\Leftrightarrow \exists z : xz \in L \Leftrightarrow yz \notin L \\
&\Leftrightarrow \neg(\forall z : xz \in L \Leftrightarrow yz \in L) \\
&\Leftrightarrow \neg x R_L y
\end{aligned}$$

Da $R_{M'} = R_L$, und der durch R_L gebildete DEA M_L minimal ist, ist auch der reduzierte DEA minimal. \square

Da reduzierter und Minimalautomat gleich dem durch R_L gebildeten Automaten sind, sind sie auch (bis auf Umbenennung der Zustände) eindeutig.

1.2.6 Abschlusseigenschaften

Aufgrund der Charakterisierung der regulären Sprachen durch reguläre Ausdrücke, die zu diesem Namen führten, wissen wir, daß diese Sprachfamilie abgeschlossen ist unter Konkatenation, Vereinigung und Kleene'scher Hüllenbildung. Der folgende Satz zeigt dies noch für Komplementbildung und wegen deMorgan daher auch für Schnittbildung. Letztere ist als eine Art Filtervorgang häufig recht nützlich. Beispielsweise kann es die Anwendung des Pumping-Lemmas vereinfachen: so ist der Schnitt der Sprache $L = \{\text{alle Wörter über } a \text{ und } b, \text{ die die gleiche Anzahl von } a\text{'s und } b\text{'s enthalten}\}$ mit der regulären Sprache a^*b^* die schon behandelte Menge $\{a^n b^n \mid n \geq 0\}$, von der wir schon wissen, daß sie nicht regulär ist. Also kann auch $L =$ nicht regulär sein.

Satz 6

Die regulären Sprachen sind konstruktiv abgeschlossen unter:

- Vereinigung, Schnitt, Komplement
- Produkt
- Kleene'scher Hüllenbildung

Beweis

Die Abgeschlossenheit gegenüber Vereinigung, Konkatenation und Kleenescher Hüllenbildung folgt aus der Definition regulärer Ausdrücke.

Die Abgeschlossenheit gegenüber Komplementbildung lässt sich über DEAs zeigen, indem man End- und Nicht-Endzustände vertauscht: Sei $M = (Z, \Sigma, \delta, z_0, E)$ und $L = T(M)$, setze $\bar{M} = (Z, \Sigma, \delta, z_0, Z \setminus E)$. Für $x \in \Sigma^*$ ergibt sich:

$$\begin{aligned}
x \notin L &\Leftrightarrow \hat{\delta}(z_0, x) \notin E \\
&\Leftrightarrow \hat{\delta}(z_0, x) \in Z \setminus E \\
&\Leftrightarrow x \in T(\bar{M})
\end{aligned}$$

Die Abgeschlossenheit gegenüber Schnittbildung folgt jetzt nach der Regel von deMorgan; denn für zwei Sprachen $L_1, L_2 \in \Sigma^*$ gilt: $L_1 \cap L_2 = \Sigma^* \setminus (\Sigma^* \setminus L_1 \cup \Sigma^* \setminus L_2)$
 \square

Nachtrag zu Abschlusseigenschaften:

Der Beweis des Schnittabschlusses verwendete die Komplementbildung, wäre also für nichtdeterministische Automaten eine sehr aufwendige Operation. Es gibt jedoch auch einen direkten Weg, den Schnitt- und Vereinigungsabschluss zu zeigen, über das sogenannte Kreuzprodukt (oder auch direktes Produkt). Hierbei konstruieren wir aus zwei endlichen Automaten einen neuen, dessen Zustandsmenge aus dem Produkt der beiden einzelnen Zustandsmengen besteht.

Gegeben: $M_i = (Z_i, \Sigma, \delta_i, S_i, E_i)$, $\delta_i : Z_i \times \Sigma_i \rightarrow \mathcal{P}(Z_i)$

Ziel: $M_i = (Z, \Sigma, \delta, S, E)$ mit $L(M) = L(M_1) \cap L(M_2)$ (DeMorgan direkt)

definiere: $Z := Z_1 \times Z_2$

$S := S_1 \times S_2$

$E := E_1 \times E_2$

$\delta : Z \times \Sigma \rightarrow \mathcal{P}(Z)$, $\delta(\langle z_1, z_2 \rangle, a) := \{(z'_1, z'_2) \mid z'_1 \in \delta_1(z_1, a), z'_2 \in \delta_2(z_2, a)\} = \delta_1(z_1, a) \times \delta_2(z_2, a)$

$\forall Y_1 \subseteq Z_1, Y_2 \subseteq Z_2$ gilt: $\hat{\delta}(Y_1 \times Y_2, w) = \hat{\delta}_1(Y_1, w) \times \hat{\delta}_2(Y_2, w)$

Damit:

$w \in L(M) \Leftrightarrow \hat{\delta}(S, w) \cap E \neq \emptyset \Leftrightarrow (DEF) \hat{\delta}_1(S_1, w) \times \hat{\delta}_2(S_2, w) \cap E_1 \times E_2 \neq \emptyset \Leftrightarrow$

$\hat{\delta}_1(S_1, w) \cap E_1 \neq \emptyset \wedge \hat{\delta}_2(S_2, w) \cap E_2 \neq \emptyset \Leftrightarrow w \in L_1(M_1) \wedge w \in L_2(M_2) \Leftrightarrow$

$w \in L(M_1) \cap L(M_2)$

1.2.7 Entscheidbarkeit

Im Zuge der Verwendung endlicher Automaten bei der Modellierung von Problemen stellen sich meist Probleme folgender Art ein: entscheide zu einem Automaten A , der beispielsweise die Menge der möglichen Ereignisabläufe beschreibt, und einem Wort v , das einen gewünschten Prozess beschreibt, ob $v \in L(A)$ gilt. Oder sei B ein Automat, der die verbotenen weil fehlerhaften Abläufe beschreibt. Dann gibt es aufgrund der Ausführungen des vorherigen Abschnitts einen endlichen Automaten C , der $L(A) \cap L(B)$ erkennt. Entscheide nun, ob $L(C)$ leer ist. In diesem Unterabschnitt werden einige derartige Fragetypen formalisiert und auf ihre Entscheidbarkeit hin betrachtet. Das Wortproblem für TYP-3-Sprachen ist entscheidbar.

Das Leerheitsproblem - gegeben M , gilt $T(M) = \emptyset$? - ist entscheidbar für Typ-3-Sprachen.

Das Endlichkeitsproblem - gegeben M , gilt $|T(M)| < \infty$ - ist entscheidbar für Typ-3-Sprachen.

Das Schnittleerheitsproblem - gegeben M_1, M_2 , gilt $T(M_1) \cap T(M_2) = \emptyset$? - ist entscheidbar für Typ-3-Sprachen.

Das Äquivalenzproblem - gegeben M_1, M_2 , gilt $T(M_1) = T(M_2)$? - ist entscheidbar für Typ-3-Sprachen.

1.3 Kontextfreie Sprachen

Nach der Untersuchung der regulären Sprachen gehen wir in der Chomsky-Hierarchie eine Stufe nach oben und führen in diesem Abschnitt entsprechende Betrachtungen für kontextfreie Sprachen durch. Dazu stellen wir zunächst fest, daß es kontextfreie Sprachen gibt, die nicht regulär sind.

Bsp: $\{a^n b^n \mid n \geq 1\} = L(\{\{S\}, \{a, b\}, \{S \rightarrow aSb, S \rightarrow ab\}, S)$

1.3.1 Normalformen

Wir beginnen unsere Untersuchungen kontextfreier Sprachen mit der Herleitung sogenannter *Normalformen*. Hierbei geht es darum zu zeigen, daß beliebige kontextfreie Sprachen schon von recht eingeschränkten Typ-2-Grammatiken erzeugt werden können. Ziel dieses Abschnittes ist es nun zu zeigen, dass kontextfreie Sprachen (ohne λ) von Typ-2-Grammatiken erzeugt werden können, die nur Regeln der Form $A \rightarrow a$ und $A \rightarrow BC$ enthalten. Zur Vorbereitung dieser *Chomsky-Normalform* dient das folgende Lemma.

Lemma

Zu jeder kontextfreien Grammatik $G = (V, \Sigma, P, S)$ existiert eine kontextfreie Grammatik $G' = (V', \Sigma, P', S')$, für die gilt: $P' \cap (V' \times V') = \emptyset$ und $L(G') = L(G)$.

Beweisskizze und Konstruktion

Gegeben: Typ-2-Grammatik $G = (V, \Sigma, P, S)$. $P \subseteq V \times (V \cup \Sigma)^+$.

Ziel: Elimination von Regeln in $V \times V$. Definiere die Relation \leq über V vermöge:

$$A \leq B : \Leftrightarrow A \rightarrow B \in P$$

$$A \equiv B : \Leftrightarrow A \leq^* B \text{ und } B \leq^* A$$

Schritt 1 $G' = (V' := V / \equiv, \Sigma, P', S' := [S]_{\equiv})$

$$P' := \{[A]_{\equiv} \rightarrow [V]_{\equiv} \mid A \rightarrow B \in P\}$$

$$[v]_{\equiv} \text{ Kanonische Erweiterung auf } (V \cup \Sigma)^*. [a]_{equiv} := \{a\},$$

$$[vw]_{\equiv} := [v]_{\equiv}[w]_{\equiv}$$

Schritt 2 Nach Schritt 1 gibt es keine Kreise $A \xrightarrow{*} B \xrightarrow{*} A$

Dann ist \leq partielle Ordnung auf V . Dann lassen sich die Elemente von V so durchnummerieren $V = \{A_1, \dots, A_N\}$, daß aus $A_i \rightarrow A_j \in P$ folgt $i < j$. Beginnend mit $j = n$ wird für abnehmendes j bei Auftreten einer Regel $A_i \rightarrow A_j$ die Regelmenge P ersetzt durch:

$$(P \setminus \{A_i \rightarrow A_j\}) \cup \{A_i \rightarrow v \mid A_j \rightarrow v \in P\}.$$

Definition 8

Eine Grammatik $G = (V, \Sigma, P, S)$ befindet sich in *Chomsky-Normalform*, wenn gilt: $P \subseteq V \times (V^2 \cup \Sigma)$.

Satz 7

Zu jeder kontextfreien Grammatik $G = (V, \Sigma, P, S)$ existiert ein $\hat{G} = (\hat{V}, \Sigma, \hat{P}, \hat{S})$ in Chomsky-Normalform mit $L(G) = L(\hat{G})$.

Beweis

Gemäß der Vorbemerkung können wir $P \subseteq N \times (\Sigma \cup (V \cup \Sigma)^2 (V \cup \Sigma)^*)$ annehmen. Der Beweis verläuft in zwei Schritten. Zuerst wird dafür gesorgt, dass in den rechten Seiten der Regeln Terminale und Nichtterminale nicht mehr gemeinsam auftreten können.

Schritt 1 Trennung von Nonterminalen und Terminalen:

Einführung einer "Kopie" $\hat{\Sigma}$ von Σ .

Erweiterung von V zu $V_1 := V \cup \hat{\Sigma}$, $h : (V \cup \Sigma)^+ \Rightarrow (V_1)^*[A \rightarrow A, a \rightarrow \hat{a}]$

Bilde $P_1 := \{A \rightarrow h(v) \mid A \rightarrow v \in P, |v| \geq 2\} \cup (P \cap (N \times \Sigma)) \cup \{\hat{a} \rightarrow a \mid a \in \Sigma\}$

Offenbar gilt: $L(G) = L(V_1, \Sigma, P_1, S)$.

Schritt 2 Verkürzung der rechten Seiten: Nach diesen Umformungen ist sichergestellt, dass jede rechte Seite einer Regel entweder aus einem Terminalsymbol oder aus mindestens zwei Nonterminalen besteht. Die folgende einfache Umformung ersetzt Regeln, deren rechte Seiten zu lang sind, durch mehrere Regeln mit richtiger Länge der rechten Seiten.

Für jede Regel $A \rightarrow B_1 B_2 \dots B_n$ in P_1 mit $k \geq 3$ erweitere V_1 um neue Symbole C_2, \dots, C_{k-1} und ersetze $A \rightarrow B_1 B_2 \dots B_n$ durch:

$$A \rightarrow B_1 C_2$$

$$C_2 \rightarrow B_2 C_3$$

$$\vdots$$

$$C_{k-2} \rightarrow B_{k-2} C_{k-1}$$

$$C_{k-1} \rightarrow B_{k-1} C_k$$

Diese Manipulationen verändern nicht die erzeugte Sprache, führen aber zu einer Grammatik in Chomsky Normalform. \square

Neben der eher einfach herzuleitenden Chomsky-Normalform gibt es noch die ungleich mächtigere Greibach-Normalform, benannt nach Sheila Greibach, der groß en alten Dame der formalen Sprachen.

Definition 9

Eine kontextfreie Grammatik befindet sich in Greibach-Normalform, wenn für die Produktionsmenge P gilt: $P \subseteq V \times \Sigma V^*$.

Mitteilung

Zu jeder kontextfreien Grammatik existiert eine äquivalente in Greibach-Normalform.

Der Beweis, daß sich jede kontextfreie Sprache durch eine Typ-2-Grammatik in Greibach-Normalform erzeugen lässt, übersteigt die uns hier zu Verfügung stehenden Mitteln, weswegen diese Tatsache hier nur ohne Beweis mitgeteilt wird.

1.3.2 Das uvwxy-Theorem

Ganz entsprechend dem Abschnitt 1.2.4, in dem Grenzen der Beschreibungsfähigkeit endlicher Automaten herausgearbeitet wurden, sollen jetzt die entsprechenden Ergebnisse für kontextfreie Grammatiken hergeleitet werden. Da kontextfreie Grammatiken mächtiger sind als reguläre, ist das entsprechende Werkzeug, das sogenannte uvwxy-Theorem, umfangreicher. Beim näheren Vergleich der beiden Pumpingtheoreme lassen sich jedoch weitgehende Ähnlichkeiten feststellen.

Satz 8

Zu jeder kontextfreien Sprache $L \subseteq \Sigma^*$ existiert eine Zahl $n \in \mathbb{N}$ derart, daß sich alle

$z \in L$ der Länge $|z| \geq n$ zerlegen lassen in $z = uvwxy$ für gewisse $u, v, w, x, y \in \Sigma^*$, so daß gilt:

1. $|vx| \geq 1$
2. $|vwx| \leq n$
3. Für alle $i \in \mathbb{N}$ ist $uv^iwx^iy \in L$.

Beweis

Es sei $G = (V, \Sigma, P, S)$ Grammatik in Chomsky-Normalform, die L erzeugt. Sei $k := |V|$ die Anzahl der Variablen von G . Wir setzen $n := 2^k$.

Ist nun $z \in L$ mit $|z| \geq n$, so existiert ein Ableitungsbaum T in G für z . Dieser hat $|z|$ Blätter, $|z|$ Knoten mit Ausgangsgrad 1 und $|z| - 1$ Knoten mit Ausgangsgrad 2. Da z von einer Grammatik in Chomsky-Normalform erzeugt wird, ist T bis auf die Ableitungsschritte, die die Terminale bilden, ein Binärbaum.

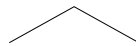
Behauptung: In T existiert ein Pfad der Länge $\geq k$ der zu einem Knoten mit Ausgangsgrad 2 führt.

(Beweis)

Da T Binärbaum ist, zeigen wir, daß jeder Binärbaum mit mind. 2^k Blättern wenigstens einen Pfad der Länge $k+1$ hat. Ein Pfad ist dabei eine von der Wurzel ausgehende Folge adjazenter Knoten, wobei kein Knoten mehrfach auftreten darf. Die Länge eines Pfades ist die Anzahl der auf ihm liegenden Knoten. Der Beweis wird durch Induktion über k geführt :

Induktionsverankerung:

$k = 1$, d.h. der Baum hat 2 Blätter. Behauptung klar, da eine von der Wurzel ausgehende Kante Pfad der Länge 2 ist.

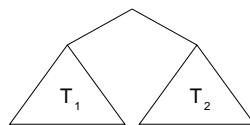


Induktionsannahme:

Die Behauptung gelte für Binärbäume mit 2^k Blättern.

Induktionsschluss:

seien T_1, T_2 Binärbäume mit gemeinsam mind. $2 * 2^k$ Blättern. Bilde aus T_1 und T_2 einen neuen Binärbaum, indem die beiden Wurzeln an einen neuen Knoten 'gehängt' werden, der Wurzel des Baumes T sei. Da T Binärbaum ist, muß ein Teilbaum T_i wenigstens die Hälfte aller Blätter, also 2^k , tragen. Nach Induktionsannahme hat T_i dann einen Pfad der Länge $k+1$. Somit hat T einen Pfad der Länge $k+2$, gebildet durch Hinzunahme Wurzel von T zum Pfad in T_i .



Also hat ein Ableitungsbaum eines Wortes z mit $|z| \geq 2^{|V|}$ einen Pfad mit mind. $|V|+1$ Variablen. Somit muß wenigstens eine Variable mehrfach im Pfad vorkommen. Das

erste Vorkommen dieser Variablen leitet auf zwei andere Variablen ab, denn sonst sind nur Ableitungen auf Terminale möglich, die einen Pfad beenden (und die betrachtete Variable daher nicht mehrfach auftreten kann).

Sei C besagte Variable. Das letzte Vorkommen von C im Pfad erzeugt das Teilwort w in $z=uvwxy$. Das vorletzte Vorkommen von C erzeugt vwx , also

$$S \Rightarrow^* uCy \Rightarrow^* uvCxy \Rightarrow^* uvwxy$$

Die Ableitung $C \Rightarrow vCx$ kann beliebig oft (insbes. überhaupt nicht) ausgeführt werden; da nur Ableitungen aus G verwendet werden, ergeben sich nur Worte aus $L = L(G)$:

$$S \Rightarrow^* uCy \Rightarrow^* uvCxy \Rightarrow^* \dots \Rightarrow^* uv^iCw^i x \Rightarrow^* uv^iwx^i y$$

Da $C \Rightarrow^* C$ in G nicht möglich ist, folgt $vx \neq \lambda$. $|vwx| \leq n = 2^{|V|}$ kann o.B.d.A. angenommen werden, wäre $|vwx| > 2^{|V|}$, so würde mit gleicher Argumentation wie zuvor folgen, daß vwx zerleg- und pumpbar ist. \square

Beispiel 9

$L := \{a^n b^n c^n | n \geq 1\}$ ist nicht kontextfrei:

Wäre L kontextfrei, so gäbe es ein n mit den Eigenschaften des obigen Satzes. Wähle $z := a^n b^n c^n$. Wegen $|z| = 3n \geq n$ gäbe es eine Zerlegung $z = uvwxy$. Wegen $|vwx| \leq n$ kann vx nicht zugleich a's und c's enthalten.

Also gilt: $\#_a(vx) \neq \#_b(vx)$ oder $\#_b(vx) \neq \#_c(vx)$. Daher gilt $\#_a(uwy) \neq \#_b(uwy)$ oder $\#_b(uwy) \neq \#_c(uwy)$, daher $uwy \notin L$. Widerspruch zur Annahme. \square

Da L kontextsensitiv ist folgt die Ungleichheit der Typ-2-Sprachen und der Typ-1-Sprachen.

Mit den Methoden, die schon bei Typ-3-Sprachen die Anwendung des uvw-Theorems ermöglichten, kann auch gezeigt werden, daß die folgenden Sprachen nicht kontextfrei sind:

- $\{0^p | p \text{ ist Primzahl}\}$
- $\{0^{2^n} | n \geq 1\}$
- $\{0^{n^2} | n \geq 1\}$.

Diese Beispiele sind Ausdruck des folgenden allgemeinen Ergebnisses.

Satz 9

Jede kontextfreie Sprache über einem einelementigen Alphabet ist regulär.

Dieser Satz wiederum ist ein Spezialfall des *Satzes von Parikh*, der die nahe Verwandtschaft von regulären und kontextfreien Sprachen beleuchtet.

Definition 10

Sei $L \subseteq \Sigma^*$ kontextfrei, $\Sigma = \{a_1, a_2, \dots, a_n\}$. Die **Parikh-Abbildung** ist wie folgt definiert:

$$\Psi : \begin{cases} \Sigma^* & \rightarrow \mathbb{N}^n \\ w & \mapsto (\#_{a_1}(w), \#_{a_2}(w), \dots, \#_{a_n}(w)) \end{cases}$$

Ψ ist ein Homomorphismus, der jedem Wort seinen Parikh-Vektor, d.h. die Angabe, welches Zeichen in dem Wort mit welcher Häufigkeit auftritt, zuordnet.

Satz von Parikh

Zu jedem kontextfreien $L \subseteq \Sigma^*$ existiert ein reguläres $L' \subseteq \Sigma^*$ so daß gilt :
 $\Psi(L) := \{\Psi(w) | w \in L\} = \Psi(L')$.

Die gemeinsam von den regulären als auch von den kontextfreien Sprachen durch die Parikh-Abbildung beschriebenen Mengen ganzzahliger Vektoren stellen neue Charakterisierungen lang bekannter mathematischer bzw. logischer Konstrukte dar. Sie sind die (nichtnegativen) Teile der *semilinearen Mengen*, die auch als *Presburger Arithmetik* bekannt sind.

1.3.3 Abschlusseigenschaften

Wie schon für die regulären Sprachen (in Abschnitt 1.2.6) wollen wir als technisches Hilfsmittel für spätere Abschnitte nun die Abschlusseigenschaften der kontextfreien Sprachen herausarbeiten. Es wird sich herausstellen, daß diese vergleichsweise schwächer sind.

Satz

- a.) Die kontextfreien Sprachen sind konstruktiv abgeschlossen unter
- Vereinigung
 - Produkt
 - Stern
- b.) Die kontextfreien Sprachen sind nicht abgeschlossen unter
- Schnitt
 - Komplement

Beweis

- a.) Sind $G_1 = (V_1, \Sigma, P_1, S_1)$ und $G_2 = (V_2, \Sigma, P_2, S_2)$ zwei kontextfreie Grammatiken mit $V_1 \cap V_2 = \emptyset$ so gilt für:
- $G_{\cup} := (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S)$
 $L(G_{\cup}) = L(G_1) \cup L(G_2)$
 - $G_{\cdot} := (V_1 \cup V_2 \cup \{S\}, \Sigma, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, S)$
 $L(G_{\cdot}) = L(G_1) \cdot L(G_2)$
 - $G_{*} := (V_1 \cup \{S\}, \Sigma, P_1 \cup \{S_1 \rightarrow \lambda, S \rightarrow S S_1\}, S)$
 $L(G_{*}) = L(G_1)^*$.
- b.) $\{a^i b^j c^k | i, j \geq 1\}$ und $\{a^i b^j c^k | i, j \geq 1\}$ sind beide kontextfrei. Ihr Schnitt $\{a^i b^j c^k | i \geq 1\}$ jedoch nicht! Da die TYP-2-Sprachen unter Vereinigung, nicht aber unter Schnittbildung abgeschlossen sind, können sie nicht gegen Komplementbildung stabil sein. \square

Beispiel 10

Wir betrachten die Sprache $L := \{w \in \{a, b, c\}^* | \#_a(w) = \#_b(w) = \#_c(w)\}$. Es ist nicht schwierig, mit Hilfe des uvwxy-Theorems zu zeigen, daß L nicht kontextfrei ist. Unter Benutzung der Abschlusseigenschaften ist das jedoch noch leichter, da es

oft möglich ist, vorherige Anwendungen des uvwxy-Theorems zu "recyclen". So gilt offenbar $L \cap a^*b^*c^* = \{a^n b^n c^n \mid n \geq 0\}$. Von dieser Sprache wissen wir, daß sie nicht kontextfrei ist. Da die kontextfreien Sprachen unter Schnitt mit regulären Sprachen abgeschlossen sind, kann auch L nicht kontextfrei sein.

1.3.4 Der CYK-Algorithmus

In diesem Abschnitt wird ein Verfahren vorgestellt, das zu einer kontextfreien Grammatik G in Chomsky-Normalform und einem Eingabewort x ermittelt, ob $x \in L(G)$ gilt. Ausgedrückt mit den Begriffen des späteren Kapitels 2 lässt sich also sagen, daß das Wortproblem für kontextfreie Sprachen entscheidbar ist.

Der folgende Algorithmus, der von Cocke und Younger sowie unabhängig davon von Kasami gefunden wurde, setzt durch dynamisches Programmieren größere Ableitungen aus kleineren zusammen. Dies geschieht durch Füllen einer Tabelle T deren Feld in der i -ten Zeile und der j -ten Spalte die Menge aller Nonterminale angibt, die das an der i -ten Stelle des Eingabewortes beginnende Teilwort der Länge j ableiten können. Das Grundprinzip besteht darin, aus den Einträgen B in $T[i, j]$ und C in $T[i+j, k]$ sowie der Existenz einer Regel $A \rightarrow BC$ zu schliessen, daß A das Teilwort der Länge $j+k$ beginnend an der i -ten Stelle der Eingabe ableiten kann; A wird also in $T[i, j+k]$ eingetragen.

$G = (V, \Sigma, P, S)$ in CNF, $A \in V, x = a_1 a_2 \dots a_n \in \Sigma$

Frage: $A \Rightarrow^* x$?

Fall 1: $n = 1 : A \Rightarrow^* x = a_1 \Leftrightarrow A \rightarrow a_1 n P$

Fall 2: $n \geq 1 : A \Rightarrow^* a_1, a_2 \dots a_n \Leftrightarrow \exists B, C \in V, 1 \leq j < n$

$A \rightarrow BC \in P, B \Rightarrow^* a_1 \dots a_j, C \xrightarrow{*} a_{j+1} \dots a_n$

$1 \leq i < j \leq n, x_{i,j} := a_i a_{i+1} \dots a_{i+j}$

Halbmatrix $T[1 \dots n, 1 \leq j \leq n-j+1], (i, j)$ mit: $1 \leq i \leq n, i+j-1 \leq n$

$T[i, j] = \{A \in V \mid A \Rightarrow^* x_{i,j}\}$.

Dann gilt $x \in L \Leftrightarrow S \in T[0, n]$!

Der CYK-Algorithmus:

Eingabe: $x = a_1 a_2 \dots a_n$

```

for  $i := 1$  TO  $n$  do
   $T[i, 1] := \{A \in V \mid A \rightarrow a_i \in P\}$ 
end for;
for  $j := 2$  TO  $n$  do
  for  $i := 1$  TO  $n+1-j$  do
     $T[i, j] := \emptyset;$ 
    for  $k := 1$  TO  $j-1$  do
       $T[i, j] := T[i, j] \cup \{A \in V \mid A \rightarrow BC \in P \wedge B \in T[i, k] \wedge C \in T[i+k, j-k]\}$ 
    end for;
  end for;
end for;
if  $S \in T[1, n]$ 
  WriteString('x liegt in L(G)')
else

```

```
WriteString('x liegt nicht in L(G)')
endif
```

Beispiel 11

$E \rightarrow E + T | T, T \rightarrow T * F | F, F \rightarrow (E) | a$, in Chomsky-Normalform.

$G = (V, \Sigma, P, E)$

$V = \{E, T, F, A, b; c; \hat{+}, \hat{*}, \hat{()}\}$

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

$P = \{E \rightarrow a | \hat{()}; A | TB | EC;$
 $T \rightarrow a | \hat{()}; A | TB;$
 $F \rightarrow a | \hat{()}; A;$
 $C \rightarrow \hat{+} T;$
 $B \rightarrow \hat{*} F;$
 $A \rightarrow E \};$
 $\hat{() \rightarrow (; \hat{) \rightarrow)};$
 $\hat{+ \rightarrow + ; \hat{* \rightarrow *}; \}$

$x = a + (a + a) * a \in L?$

$n = 9$

	1	2	3	4	5	6	7	8	9
	a	+	(a	+	a)	*	a
1	E,T,F	$\hat{+}$	$\hat{()}$	E,T,F	$\hat{+}$	E,T,F	$\hat{)}$	$\hat{*}$	E,T,F
2	-	-	-	-	C	A	-	B	
3	-	-	-	E	-	-	-		
4	-	-	-	A	-	-			
5	-	-	E,T,F	-	-				
6	-	C	-	-					
7	E	-	E,T						
8	-	C							
9	E								

Also $x \in L$ wg. $E \in T[1, 9]$.

1.3.5 Kellerautomaten

Als das wichtigste Darstellungsmittel regulärer Sprachen stellten sich im Abschnitt 1.2.1 die endlichen Automaten heraus. In diesem Abschnitt wird ein Automatenmodell vorgestellt, das genau die kontextfreien Sprachen charakterisiert. Leider gehen im kontextfreien Fall wichtige Eigenschaften verloren; so sind Determinismus und Nicht-determinismus nicht äquivalent und es gibt im Allgemeinen keinen minimalen Automaten, der sich algorithmisch ermitteln ließe.

Das Automatenmodell für die kontextfreien Sprachen lässt sich anschaulich als endlicher Automat schildern, dem zusätzlich ein last-in-first-out-Speicher zur Verfügung steht, auf dem allerdings nur das letztespeicherte Zeichen und nicht die darunter liegenden gelesen werden darf. Da ein derartiger Speicher oft als Kellerspeicher bezeichnet wird, heißt dieser Automatentyp *Kellerautomat*. (Im Englischen *push-down auto-*

maton). Dieser ist zu unterscheiden vom *stack automaton*, im Deutschen häufig *Stapelautomat*, bei dem der Inhalt des kellerartig bearbeiteten Stapels zusätzlich noch zerstörungsfrei gelesen werden darf.

Kellerautomat = Endliche Kontrolle + Keller

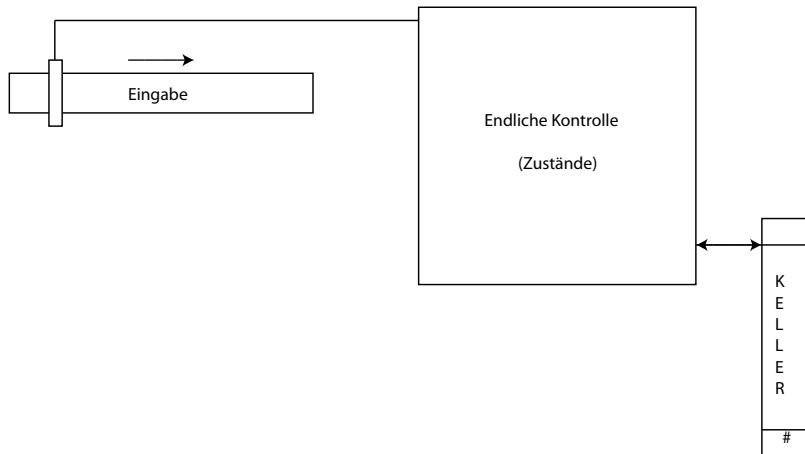


Abbildung 6: Kellerautomat

Definition 11

Ein (nichtdeterministischer) Kellerautomat, kurz: (N)PDA, ist ein 6-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ mit:

- Z einer endlichen Menge von Zuständen
- Σ dem Eingabealphabet
- Γ dem Kelleralphabet
- $z_0 \in Z$ dem Startzustand
- $\# \in \Gamma$ dem Kellerbodenzeichen und
- $\delta \subseteq Z \times (\Sigma \cup \{\lambda\}) \times \Gamma \times Z \times \Gamma^*$ einer endlichen Mengen von Transitionen

Anschauliche Bedeutung von $(q, a, A, q', B_1 \dots B_n) \in \delta$:

Im Zustand $q \in Z$ und mit oberstem Kellerzeichen A kann M bei Lesen des Eingabeteils $a \in \Sigma \cup \{\lambda\}$ in den Zustand q' übergehen und A durch $B_1 \dots B_n$ ersetzen:

Ein Kellerautomat akzeptiert durch Endzustände oder durch leeren Keller.

Eine Konfiguration eines Kellerautomaten $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ ist ein Tripel $k \in Z \times \Sigma^* \times \Gamma^*$

Übergangsrelation \vdash_M auf der Menge der Konfigurationen:

$$(z, a_1 \dots a_n, A_1 \dots A_m) \vdash_M (z', a_2 \dots a_n, B_1 \dots B_k A_2 \dots A_m),$$

falls $(z, a_1, A_1, z', B_1 \dots B_k) \in \delta$.

$(z, a_1 \dots a_n, A_1 \dots A_m) \vdash_M (z', a_1 \dots a_n, B_1 \dots B_k A_1 \dots A_m)$,
 falls $(z, \lambda, A_1, z', B_1 \dots B_k) \in \delta$; hierbei handelt es sich um einen *spontanen Übergang*, da das oberste Kellerzeichen nicht beachtet wird.

Sei nun \vdash_M^* die reflexiv-transitive Hülle von $\vdash \subseteq Z \times \Sigma^* \times \Gamma^*$.

Die von einem Kellerautomaten $M = (Z, \Sigma, \Gamma, \delta, z_0, \#)$ (mit leerem Keller) akzeptierte Sprache ist:

$$N(M) := \{x \in \Sigma^* \mid (z_0, x, \#) \vdash_M^* (z, \lambda, \lambda) \text{ für ein } z \in Z\}$$

Der folgende Kellerautomat erkennt die sogenannte Spiegelsprache, deren Wörter aus einem Wortanfang w bestehen, der von einem Trennzeichen $\$$ gefolgt wird und daran anschließend das Wort w in umgekehrter Leserichtung.

Beispiel 12

Ein Kellerautomat für: $L = \{a_1 \dots a_n \$ a_n \dots a_1 \mid a_i \in \{a, b\}\}$

$M = (Z = \{z_0, z_1\}, \Sigma = \{a, b, \$\}, \Gamma = \{\#, A, B\}, \delta, z_0, \#)$ mit:

$\delta(z_0, \cdot, \cdot)$:	#	A	B
a	$(z_0, A\#)$	(z_0, AA)	(z_0, AB)
b	$(z_0, B\#)$	(z_0, BA)	(z_0, BB)
\$	$(z_1, \#)$	(z_1, A)	(z_1, B)
λ	-	-	-

$\delta(z_1, \cdot, \cdot)$:	#	A	B
a	-	(z_1, λ)	-
b	-	-	(z_1, λ)
\$	-	-	-
λ	(z_1, λ)	-	-

Wir haben hier die Menge der Transitionen tabellenartig aufgelistet. Im Zustand z_0 liest der Automat Eingabesymbole a und B ein und legt jeweils ein A oder B zusätzlich auf dem Keller ab, der zunächst nur mit dem Symbol $\#$ gefüllt ist. Bei Lesen des Trennzeichens $\$$ wechselt der Automat in den Zustand z_1 , in dem die obersten Kellersymbole mit dem nächsten Eingabesymbol verglichen werden. Bei nichtpassendem Eingabesymbol blockiert der Automat. Erreicht er das Kellerbodensymbol $\#$, so löscht er dieses ohne ein Eingabesymbol zu lesen und hält akzeptierend mit leerem Keller. Eine akzeptierende Rechnung auf der Eingabe $aab\$baa$ sieht wie folgt aus:

$$(z_0, aab\$baa, \#) \vdash^1 (z_1, ab\$baa, A\#) \vdash^2 (z_0, \$baa, BAA\#) \vdash^1 (z_1, baa, BAA\#) \vdash^3 (z_1, \lambda, \#) \vdash^1 (z, \lambda, \lambda)$$

Bei der Arbeitsweise dieses Kellerautomaten fällt auf, daß es zu jedem Zeitpunkt höchstens eine anwendbare Transition gibt; der Automat arbeitet also *deterministisch*. Dieses

Thema wird uns im folgenden Unterabschnitt beschäftigen. Betrachten wir die Spiegelsprache ohne Trennzeichen $\$, L' = \{a_1 \dots a_n a_{n+1} a_n \dots a_1 \mid a_i \in \{a, b, \lambda\}\}$ so muß die Wortmitte erraten werden : Zu jedem Zeitpunkt der Berechnung wird nur ein einzelnes Zeichen der Eingabe betrachtet, während des Lesens der ersten Worthälfte ist dabei nicht klar, ob ein Zeichen auf den Keller gelegt oder mit ihm verglichen werden soll. Der Übergang zum 'vergleichenden' Zustand z_1 muß zu jedem Zeitpunkt möglich sein. Beispielhaft :

$$\text{ND: } \langle z_0, a, A \rangle \begin{cases} \langle z_0, AA \rangle \\ \langle z_1, \lambda \rangle \end{cases}$$

Wie angekündigt erweisen sich die Kellerautomaten als äquivalent zu den kontextfreien Grammatiken. Es stellt sich jedoch heraus, dass die dabei konstruierten Automaten notwendigerweise nichtdeterministisch sind.

Satz 10

Eine Sprache $A \subseteq \Sigma^*$ ist genau dann kontextfrei, wenn A von einem nicht deterministischen Kellerautomaten erkannt wird.

Beweis

” \Rightarrow ”

Der im folgende zu einer beliebigen Grammatik konstruierte *Top-Down-Automat* startet mit dem Startsymbol der Grammatik als unterstem Kellerzeichen. Befindet sich oben auf dem Keller ein Nonterminal A , so wählt der Automat nichtdeterministisch eine Regel, deren linke Seite A ist und ersetzt auf dem Keller A durch die rechte Seite der Regel. Befindet sich ein Terminalzeichen oben auf dem Keller, so wird dieses mit dem nächsten Eingabesymbol verglichen und bei Übereinstimmung gelöscht. Ansonsten blockiert der Automat. Der so konstruierte Kellerautomat weist nur einen Zustand auf und akzeptiert bei vollständig gelesener Eingabe mit leerem Keller.

Sei $G = (V, \Sigma, P, S)$ vom Typ-2 mit $L(G) = A$.

Wir setzen $M := (Z = \{z\}, \Sigma, \Gamma := V \cup \Sigma, \delta, z, S)$

mit: $\delta(z, a, a) = \{(z, \lambda)\}$

und: $\delta(z, \lambda, A) = \{(z, \alpha) \mid A \rightarrow \alpha \in P\}$

Beh. (ohne Beweis): $x \in L(G) \Leftrightarrow \langle z, x, S \rangle \vdash^* \langle z, \lambda, \lambda \rangle$

” \Leftarrow ”

Sei $M = (Z, \sigma, \Gamma, \delta, z_0, \#)$, o.B.d.A. hat jede Transition die Form $(z, a, A, z', B_1 \dots B_k)$ mit $k \leq 2$. Das stellt keine Einschränkung dar, da jede Regel $(z, a, A, z', B_1 \dots B_k)$ mit $k > 2$ ersetzt werden kann durch eine Menge von Regeln, die die B_i einzeln auf den Keller legen :

Ersetze dazu $(z, a, A, z', B_1 \dots B_k)$ durch

$$(z, a, A, z_1, B_1), (z_1, \lambda, B_1, z_2, B_1 B_2) \dots, (z_{k-1}, \lambda, b_{k-1}, z', B_{k-1} B_k)$$

wobei die Zustände z_i für jede Ersetzung neu zur Zustandsmenge hinzugefügt werden.

Bilde dann $G = (V, \Sigma, P, S)$

mit $V := \{S\} \cup Z \times \Gamma \times Z$

und $P := \{S \rightarrow \langle z_0, \#, z \rangle \mid z \in \delta\} \cup$
 $\{\langle z, A, z' \rangle \rightarrow a \mid \langle z', \lambda \rangle \in \delta(z, a, A)\} \cup$
 $\{\langle z, A, z' \rangle \rightarrow a \langle z_1, B, z' \rangle \mid \langle z_1, B \rangle \in \delta(z, a, A), z \in Z\} \cup$
 $\{\langle z, A, z' \rangle \rightarrow a \langle z_1, B, z_2 \rangle \langle z_2, C, z' \rangle \mid \langle z_2, BC \rangle \in \delta(z, a, A), z, z_2 \in Z\} \quad \square$

Bemerkungen:

a.) Geht man im obigen Beweis von einer kontextfreien Grammatik in Greibach-Normalform aus, in der also alle Regeln die Gestalt $A \rightarrow a\alpha$ mit $\alpha \in V^*$ haben, so können wir daraus in sehr ähnlicher Weise einen Kellerautomaten definieren vermöge $\delta(z, a, A) := \{(z, \alpha) \mid A \rightarrow a\alpha \in P\}$, der keine λ -Übergänge besitzt. Kontextfreie Sprachen lassen sich also von Kellerautomaten erkennen, die in jedem Schritt ein Eingabezeichen lesen.

b.) Neben dem oben skizzierten *Top-Down-Automat* gibt es auch den *Bottom-Up-Automat*, der auf dem Keller rechte Seiten durch linke ersetzt und akzeptiert, wenn am Ende der Berechnung auf dem Keller das Startsymbol steht. Obwohl diese Konstruktion etwas umständlicher zu formalisieren ist, erweist sie sich bei der Behandlung deterministischer Automaten als leistungsfähiger.

1.3.6 Deterministisch kontextfreie Sprachen

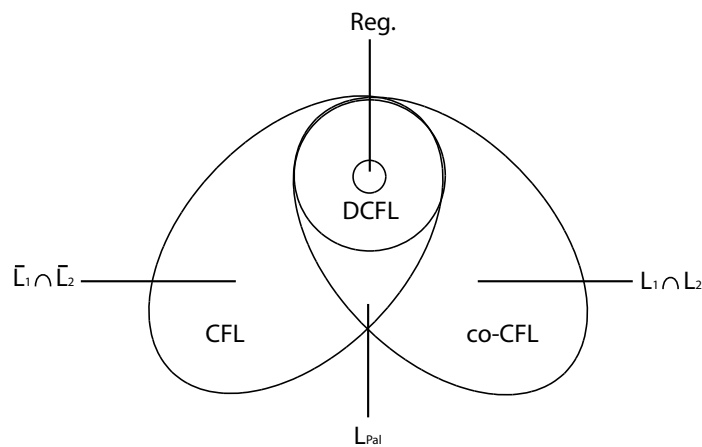


Abbildung 7:

$co-X = \{L \mid \bar{L} \in X\}$ [CFL= Kontextfrei, co-CFL = KomplementCFL ; DCFL = Deterministisch CFL, REG = Regulär, DPDA = Deterministischer Kellerautomat]

Definition 12

Ein Kellerautomat heißt **deterministisch**, wenn die Folgekonfiguration einer jeden Konfiguration eindeutig bestimmt ist (falls sie existiert) : $|\delta(z, a, A)| + |\delta(z, \lambda, A)| \leq 1$ für alle $z \in Z, a \in \Sigma, A \in \Gamma$.

Dazu kommt, daß deterministische Kellerautomaten per Endzustand akzeptieren und nicht per leerem Keller (für nicht-det. ist beides äquivalent). Sie werden dargestellt durch ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E)$

Eine Sprache $L \subseteq \Sigma^*$ heisst deterministisch kontextfrei (\in DCFL), falls sie von einem deterministischen Kellerautomaten erkannt wird.

Folgerung

Die Berechnung eines deterministischen Kellerautomaten entspricht einer verzweigungsfreien Konfigurationsfolge. (Im Gegensatz zu möglichen 'Konfigurationsbäumen' bei Berechnungen nichtdet. Kellerautomaten.)

Satz 11

Die deterministisch kontextfreien Sprachen sind unter Komplement abgeschlossen (DCFL=co-DCFL)

Beispiel 13 - $L_1 = \{a^n b^m c^n | n, m \geq 0\}$
 $z_0 a \# \rightarrow z_0 A \#, z_0 a A \rightarrow z_0 A A, z_0 b A \rightarrow z_1 A,$
 $z_1 b A \rightarrow z_1 A, z_1 c A \rightarrow z_2 \lambda, z_2 c A \rightarrow z_2 \lambda, z_2 \lambda \# \rightarrow z_3 \lambda (z_3 \text{ Endzustand})$
 - $L_2 = \{a^n b^n c^m | n, m > 0\}$
 analog

Satz 12

Die deterministisch kontextfreien Sprachen sind nicht abgeschlossen unter Schnitt und Vereinigung.

Beweis

- Schnitt
 $L_1 \cap L_2 = \{a^n b^n c^n | n, m > 0\}$ nicht kontextfrei
- Vereinigung
 angenommen DCFL abgeschlossen unter Vereinigung, dann :
 $\overline{L_1}, \overline{L_2} \in DCFL \Rightarrow \overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2 \in DCFL$, im Widerspruch zur nicht-Abgeschlossenheit unter Schnitt. \square

Satz 13

Der Schnitt einer deterministisch kontextfreien Sprache mit einer regulären Sprache ist wieder deterministisch kontextfrei.

Beweis

Sei $M_1 = (Z; \Sigma, \Gamma, \delta_1, z_{01}, \#, E_1)$ ein Kellerautomat für L_1 (E_1 Endzustand) und $M_2 = (Z, \Sigma, \delta_2, z_{02}, E_2)$ ein DFA für L_2 . Wir konstruieren einen Kreuzproduktautomaten $M_3 = (Z_1 \times Z_2, \Sigma, \Gamma, \delta_3, (z_{01}, z_{02}), \#, E_1 \times E_2)$ wobei für die Überföhrungsfkt. gilt :

$$\delta_3((z_1, z_2), a, A) \ni ((z'_1, z'_2), B_1 \dots B_k) \quad \text{wenn} \quad \delta_1(z_1, a, A) \ni (z'_1, B_1 \dots B_k) \\ \text{und} \quad \delta_2(z_2, a) = z'_2 \quad \square$$

Definition 13

Für $L \subseteq \Sigma^*$ ist $MIN(L) := \{w \in L | w = uv, u \in L \Rightarrow v = \lambda\}$ d.h. w hat kein echtes Präfix in L .

Satz 14

$L \in DCFL \Rightarrow MIN(L) \in DCFL$

Beweis

Streiche im Automaten alle Übergänge, die von Endzuständen ausgehen. \square

Satz 15

Es gibt Sprachen, die im Schnitt der Kontextfreien Sprachen mit deren Komplement enthalten, jedoch nicht deterministisch kontextfrei sind.

Beweis

Sei $L_{Pal} = \{ww^R | w \in \{a, b\}^*\}$ [Palindrome gerader Buchstabenzahl, z.B. Reittier]
 L_{Pal} ist kontextfrei, da die Sprache von der kontextfreien Grammatik $G = (\{S\}, \{a, b\}, \{S \rightarrow aSa | bSb | aa | bb\}, S)$ gebildet wird.

Um zu zeigen, daß L_{Pal} auch im Komplement der Kontextfreien Sprachen enthalten ist, geben wir eine kontextfreie Grammatik an, die das Komplement von L_{Pal} generiert (d.h. alle Zeichenfolgen über $\{a, b\}$, die keine Palindrome der Form ww^R sind). Diese ist $G = (\{S, T\}, \{a, b\}, P, S)$, wobei

$$P = \{S \rightarrow aSa | bSb | aTb | bTa | ab | ba | a | b, T \rightarrow aT | bT | a | b\}$$

Zuletzt bleibt zu zeigen, daß L_{Pal} nicht deterministisch kontextfrei ist:
 Angenommen $L_{Pal} \in DCFL$, dann gilt nach Satz 13 :

$$L_1 = L_{Pal} \cap (ab)^+(ba)^+(ab)^+(ba)^+ = \{(ab)^m (ba)^n (ab)^n (ba)^m | m > 0, n \geq 0\} \in DCFL$$

und nach Satz 14

$$L_2 = MIN(L_1) = \{(ab)^m (ba)^n (ab)^n (ba)^m | 0 \leq n < m\} \in DCFL$$

Betrachte $z = (ab)^{n+1} (ba)^n (ab)^n (ba)^{n+1} \in L_2$: Nach Pumping-Lemma existiert eine Zerlegung $z = uvwxy$ mit $|vx| \geq 1$, $|vwx| \leq n$ und $\forall i \geq 0 : uv^iwx^iy \in L_2$. vwx muß nun Infix von $(ba)^n (ab)^n$ sein, denn wegen $|vwx| \leq n$ können nicht gleichzeitig Zeichen aus $(ab)^{n+1}$ und $(ba)^{n+1}$ gepumpt werden (was für Symmetrieerhalt nötig wäre). Weiterhin müssen v und x jeweils sowohl aus a's als auch aus b's bestehen, wobei $v = x^R$. Dann verletzt jedoch bereits uv^2wx^2y die an Worte aus L_2 gestellte Bedingung $n < m$. Somit kann L_2 nicht kontextfrei sein, im Widerspruch zu $L_2 \in DCFL$. Also ist L_2 nicht deterministisch kontextfrei.

Insgesamt gilt : $L_{Pal} \in (CFL \cap co-CFL) \setminus DCFL$. \square

Bemerkung

- DCFL nicht abgeschlossen unter Spiegelung $L^R = \{w^R | w \in L\}$
- $\{a^n b^m c^n | n, m \geq 0\} \cup \{a^n b^m c^m d | n, m \geq 0\} \notin DCFL$
- $\{a^n b^m c^n | n, m > 0\} \cup \{a^n b^m d^m | n, m \geq 0\} \in DCFL \setminus (DCFL \lambda\text{-frei})$

1.3.7 Entscheidbarkeit bei kontextfreien Sprachen**Wortproblem**

ist bereits durch CYK-Algorithmus positiv entschieden

Leerheitsproblem

Betrachte die zur Sprache gehörende Grammatik $G = (V, \Sigma, \delta, S)$. Wir gehen davon aus, daß G in CNF vorliegt. Die Nonterminale werden nun danach unterschieden, ob sie vollständig auf Terminalfolgen abgeleitet werden können, d.h. ob für $A \in V$ mit Produktionen aus δ gilt: $A \rightarrow^+ \alpha$ mit $\alpha \in \Sigma^*$. Derartige Terminale nennt man produktiv. Die produktiven Terminale werden wie folgt gefunden:

1. markiere $A \in V$ als produktiv, wenn $A \rightarrow a \in \delta$ für ein $a \in \Sigma$
2. markiere $A \in V$ als produktiv, wenn $A \rightarrow BC \in \delta$ und B und C bereits als produktiv markiert sind.
3. wiederhole Schritt 2, bis sich keine Änderungen mehr ergeben

Es gilt: Die erzeugte Sprache ist leer $\Leftrightarrow S$ wurde nicht markiert.

Endlichkeitsproblem

Sei n die Pumping-Zahl.

$|L| = \infty \Leftrightarrow \exists z \in L$ mit $n \leq |z| < 2n$

Beweis:

" \Rightarrow "

Falls $|z| \geq 2n \Rightarrow z = uvwxy$, $uvw \in L$, $|uvw| < |z|$ und $|uvw| \geq n$.

Setze $z := uvw$ und wiederhole bis $n \leq |z| < 2n$.

" \Leftarrow "

Direkt mit Pumping-Lemma.

oder effizienter:

Bilde Graphen (V, E) wobei V aus den produktiven Nonterminalen besteht und die Kantenmenge definiert ist als $E = \{(A, B) | A \rightarrow \alpha B \beta, \alpha, \beta \in (V \cup \Sigma)^*\}$. Suche denn Zyklen im Graphen, ein Zyklus entspricht einer Ableitung z.B. $A \rightarrow^* \alpha_1 \alpha_2 A \beta_2 \beta_1$. \square

1.4 Kontextsensitive und Typ-0-Sprachen

Turingmaschine = endliche Kontrolle + Turingband

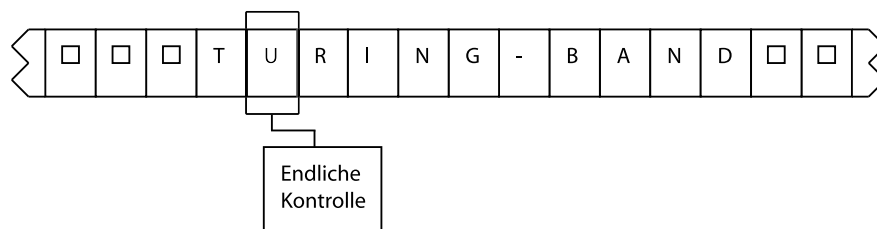


Abbildung 8:

Definition 14

Eine Turingmaschine (TM) ist ein 7-Tupel $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$, wobei

- Z eine endlichen Menge von Zuständen

- Σ das Eingabealphabet
- $\Gamma \supseteq \Sigma$, das Arbeits- oder Bandalphabet
- $\delta : Z \times \Gamma \xrightarrow{PAR} Z \times \Gamma \times \{L, R, N\}$ (det. Fall) bzw.
 $\delta \subseteq Z \times \Gamma \longrightarrow P(Z \times \Gamma \times \{L, R, N\})$ (nicht-det. Fall) die Überföhrungsfunktion
- $z_0 \in Z$ der Startzustand
- $\square \in \Gamma \setminus \Sigma$ das Blanksymbol
- $E \subseteq Z$ die Menge der Endzustände

ist.

Wenn $\delta(z, a) = (z', b, x)$ bzw. $\delta(z, a) \ni (z', b, x)$ gilt, so muss bzw. kann M im Zustand Z bei Lesen von a in den Zustand z' übergelien, auf dem Band a mit b überschreiben und den Lese/Schreibkopf nach x bewegen. $x = L$: links, $x = R$: rechts, $x = N$: keine Bewegung.

Definition 15

Eine Konfiguration einer TM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ ist ein Wort $\alpha z \beta \in \Gamma^* Z \Gamma$.

Interpretation:

Durch eine Konfiguration $\alpha z \beta$ ist eine Momentaufnahme der TM gegeben. Dabei ist α der Bandinhalt links des Lese-/Schreibkopfes, β der Bandinhalt ab dem Kopf bis zum rechten Ende. z gibt den Zustand der TM an.

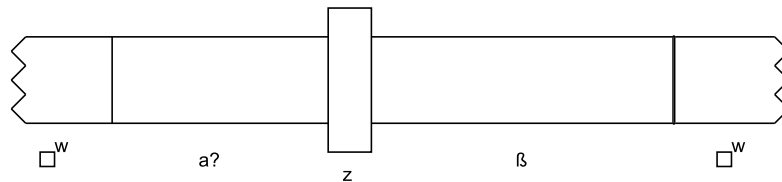


Abbildung 9:

Startkonfiguration zu einem Wort: $x \in \Sigma^* z_0 x$

Definition 16

Für eine Konfiguration $k = a_1 a_2 \dots a_m z b_1 \dots b_n$ setzen wir:

Fall 1 $k \vdash_M^1 a_1 \dots a_m z' c b_2 \dots b_n$ wenn $\delta(z, b_1) = (z', c, N)$

Fall 2 $k \vdash_M^1 a_1 \dots a_m c z' b_2 \dots b_n$ wenn $\delta(z, b_1) = (z', c, R)$

Fall 3 a) $m > 0$

$k \vdash_M^1 a_1 \dots a_{m-1} z' a_m c b_2 \dots b_n$

b) $m = 0$

$k \vdash_M^1 z' \square c b_2 \dots b_n$ wenn $\delta(z, b_1) = (z', c, L)$

Beispiel 14

Inkrementieren einer Binärzahl:

$M = (\{z_0, z_1, z_2, z_e\}, \{1, 0\}, \{0, 1, \square\}, \delta, z_0, \square, \{z_e\})$ wobei δ gegeben ist durch :

δ	0	1	\square
z_0	$(z_0, 0, R)$	$(z_0, 1, R)$	(z_1, \square, L)
z_1	$(z_2, 1, L)$	$(z_1, 0, L)$	$(z_e, 1, N)$
z_2	$(z_2, 0, L)$	$(z_2, 1, L)$	(z_e, \square, R)

Bei Eingabe 101 ergibt sich als Konfigurationsfolge:

$$z_0101 \vdash 1z_001 \vdash 10z_01 \vdash 101z_0 \vdash 10z_11 \vdash 1z_100 \vdash z_2100 \vdash z_2\square110 \vdash \square z_e110$$

Definition 17

Die von einer TM M akzeptierte Sprache ist definiert als:

$$T(M) := \{x \in \Sigma^* \mid \exists \alpha, \beta \in \Gamma^*, \exists z \in E, z_0x \vdash^* \alpha z \beta\}$$

Vordefinition

Eine NTM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ heisst linear beschränkt, wenn für $x \in \Sigma^+$ und alle $\alpha, \beta \in \Gamma^*$ mit $z_0x \vdash_M^* \alpha z' \beta$ gilt: $|\square^* \setminus \alpha \beta / \square^*| \leq |x|$

$$\square^* \setminus v := w \text{ mit } \exists i v = \square^* w \text{ w} \in (\Gamma \times \square) \Gamma^*$$

Im weiteren sei $\hat{\Sigma} := \{\hat{a} \mid a \in \Sigma\}$ eine zu Σ disjunkte Kopie von Σ .

Definition 18

Eine NTM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ heisst linear beschränkt (kurz: LBA, v. engl. 'linear bound automaton'), wenn für alle $a_1 \dots a_n \in \Sigma^+$ und alle aus der Konfiguration $z_0 a_1 \dots a_n$ erreichbaren Konfigurationen $\alpha z \beta$ gilt: $|\alpha \beta| = n$.

Die von einer linear beschränkten Turingmaschine akzeptierte Sprache ist wie folgt definiert:

$$T(M) = \{a_1 a_2 \dots a_n \in \Sigma^+ \mid z_0 a_1 \dots \hat{a}_n \vdash_M^* \alpha z \beta \text{ für } \alpha, \beta \in \Gamma^* \text{ und } z \in E\}$$

Satz 16

Die von linear beschränkten, nichtdeterministischen Turingmaschinen akzeptierten Sprachen sind genau die Typ-1-Sprachen

Ohne Beweis

Satz 17

Die von Turingmaschinen akzeptierten Sprachen sind die Typ-0-Sprachen.

Bemerkung: Nichtdeterministische und deterministische Turingmaschinen sind äquivalent. Allerdings sind die Typ-0-Sprachen nicht unter Komplement abgeschlossen. DLBAs und LBAs sind beide unter Komplement abgeschlossen; ihre Äquivalenz ist aber noch offen.

1.5 Tabellarischer Überblick

	Grammatik	Automat	Sonstiges
TYP-3	einseitig linear	endlicher Automat	Reg. Ausdruck
TYP-2	kontextfrei	Kellerautomat	
TYP-1	kontextsensitiv	LBA	
TYP-0	allg. Regelgrammatik	Turingmaschine	

Äquivalente Darstellungen

	U	∩	¬	•	*
TYP-3	+	+	+	+	+
Det. KF	-	-	+	-	-
TYP-2	+	-	-	+	+
TYP-1	+	+	+	+	+
TYP-0	+	+	-	+	+

Abschlusseigenschaften

	U	∩	¬	•	*
TYP-3	+	+	+	+	+
Det. KF	+	+	-	+	-
TYP-2	+	+	-	-	-
TYP-1	+	-	-	-	-
TYP-0	-	-	-	-	-

Entscheidbarkeit

2 Entscheidbarkeit und Komplexität

2.1 Der intuitive Begriff der Berechenbarkeit und die Churchsche These

Berechenbarkeit von Funktionen



Abbildung 10:

Vordefinition

Eine partielle Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}^m$ heisst *berechenbar*, wenn es einen Algorithmus gibt, der bei Eingabe $(n_1, n_2 \dots n_k) \in \mathbb{N}^k$ genau dann nach endlicher Zeit mit einer Ausgabe m zu einem Halt kommt, wenn f an der Stelle $(n_1, n_2 \dots n_k)$ definiert ist und $f(n_1, n_2 \dots n_k) = m$ gilt.

Beispiel 15

Der Algorithmus

⌈ *INPUT*(n);

⌋ *WHILE true DO*;

berechnet die an keiner Stelle definierte Funktion Ω mit leerem Wertebereich.

$$\Omega : \begin{cases} \mathbb{N} \rightarrow \emptyset \\ n \mapsto \uparrow \end{cases}$$

Beispiel 16

f mit $f(n) = 1$ ist berechenbar

Beispiel 17

Von der Funktion

$$g(n) = \begin{cases} 1 & \text{falls die Dezimalbruchdarstellung von } n \text{ in der} \\ & \text{Dezimalbruchentwicklung von } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

ist nicht bekannt, ob sie berechenbar ist.

Beispiel 18

Die Funktion

$$h(n) = \begin{cases} 1 & \text{falls } n \text{ aufeinanderfolgende } 7\text{'en in der Dezimalbruchdarstellung von } \pi \\ & \text{auftreten} \\ 0 & \text{sonst} \end{cases}$$

ist dagegen berechenbar.

2.2 Turingberechenbarkeit

Definition 19

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heisst turingberechenbar genau dann, wenn es eine deterministische Turingmaschine M gibt derart, daß für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, n_2 \dots n_k) = m \quad \text{gdw.} \quad z_0 \text{bin}(n_1) \# \text{bin}(n_2) \# \dots \# \text{bin}(n_k) \vdash_M^* \square^* z \text{bin}(n) \square^*$$

für ein $z \in E$

Definition 20

Eine Funktion $f : \Sigma^* \rightarrow \Sigma^*$ heisst turingberechenbar genau dann, wenn es eine deterministische Turingmaschine gibt derart, daß für alle $x, y \in \Sigma^*$ gilt:

$$f(x) = y \quad \text{gdw.} \quad z_0 x \vdash^* \square^* z y \square^* \quad \text{für ein } z \in E$$

Beispiel 19

Die Nachfolgerfunktion $\text{succ} : \begin{cases} \mathbb{N} \rightarrow \mathbb{N} \\ n \mapsto n + 1 \end{cases}$ bzw. $\text{succ} : \begin{cases} \{0, 1\}^* \rightarrow \{0, 1\}^* \\ \text{bin}(n) \mapsto \text{bin}(n + 1) \end{cases}$ ist nach Bsp. 14 turingberechenbar.

Beispiel 20

Ω aus Bsp. 15 ist turingberechenbar.

Satz 18

Eine formale Sprache $L \subseteq \Sigma^*$ ist genau dann vom Typ-0, wenn die Funktion :

$$\chi'_L : \begin{cases} \Sigma^* \rightarrow \{0, 1\} \\ w \mapsto \begin{cases} 1, w \in L \\ \text{undef}, w \notin L \end{cases} \end{cases}$$

turingberechenbar ist.

ohne Beweis

Man beachte, daß von der entsprechenden TM nur für Worte, die in der Sprache liegen, ein definierter Bandinhalt im Endzustand verlangt wird. In Abschnitt 1.2.7 wird auf dieses Verhalten näher eingegangen.

Mehrbandturingmaschinen

Mehrbandturingmaschinen arbeiten auf beliebig vielen Bändern (was die Einband-TM zu einer speziellen Mehrband-TM macht). Die einzelnen Bänder, bzw. die zugehörigen Schreib-/Leseköpfe können dabei unabhängig voneinander gesteuert werden. Die formale Definition unterscheidet sich von der der Einband-TM (Def. 14) nur in der Überföhrungsfunktion. Diese hat für eine n-Band-TM die Form : $\delta : Z \times \Gamma^n \rightarrow Z \times (\Gamma \times \{L, R, N\})^n$. Die Konfiguration einer Mehrband-TM kann entsprechend definiert werden, etwa $k \in K \subseteq \Gamma^n Z \Gamma^n$. Daraus ergibt sich auch die Definition der Übergangsrelation $\vdash_n \subseteq K \times K$.

Berechnung von Funktionen - Akzeptanz von Sprachen

Der folgende Satz besagt, daß eine Mehrband-TM nicht mehr berechnen kann als eine Einband-TM.

Satz 19

Zu jeder Mehrband-Turingmaschine M gibt es eine Einband-Turingmaschine M' , für die $T(M) = T(M')$ (bzw. $f_n = f_{n'}$) gilt.

Ohne Beweis

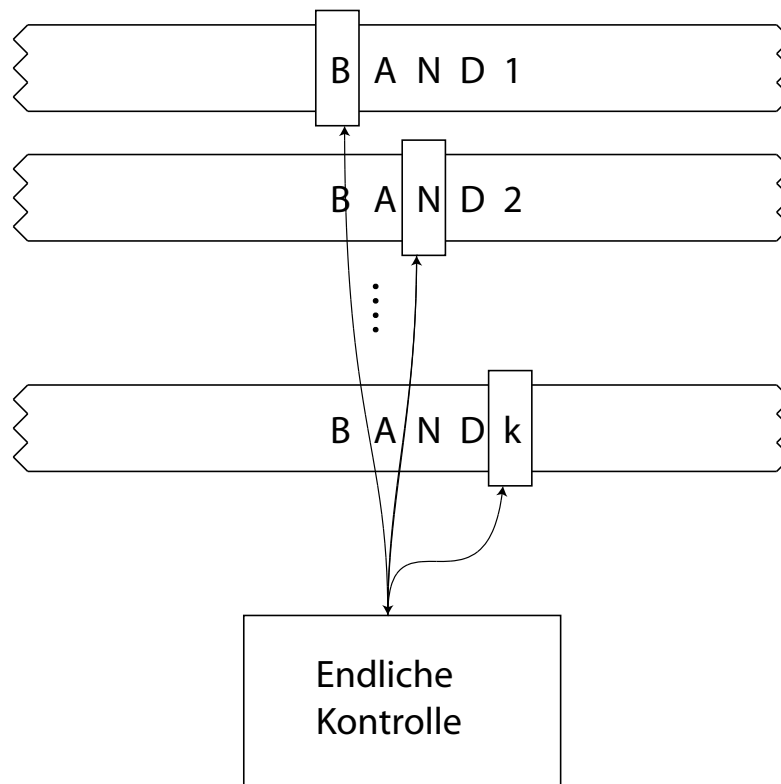


Abbildung 11: Mehrbandturingmaschine

2.3 LOOP-, WHILE-, und GOTO-Berechenbarkeit

LOOP-Berechenbarkeit

"Programmiersprache" LOOP:

- Variablen: x_0, x_1, x_2, \dots
- Konstanten: $0, 1, 2, \dots$
- Trennsymbol: $;$, $:=$
- Operatoren: $+$, $-$
- Schlüsselworte: *LOOP*, *DO*, *END*

Kontextfreie Sprache der syntaktisch korrekten LOOP-Programme:

$$\langle LP \rangle ::= \langle LP \rangle ; \langle LP \rangle \mid \text{LOOP } x_i \text{ DO } \langle LP \rangle \text{ END} \mid x_i := x_j + c \mid x_i := x_j - c$$

Wir betrachten die Arbeitsweise von LOOP-Programmen anhand einer *Registermaschine*, kurz RAM (v. engl. 'random access machine'). Eine RAM besteht aus einer endlichen Anzahl von Registern und einem unendlichen Speicher. In den Registern werden dabei elementare Anweisungen ausgeführt, bei LOOP-Programmen Addition und Subtraktion. Die zu manipulierenden Werte werden aus dem Speicher gelesen, dorthin werden sie auch wieder zurückgeschrieben. Der unendliche Speicher besteht aus

indizierten Zellen, jede Zelle kann eine beliebig große natürliche Zahl speichern. Wir betrachten lediglich die Werte in den Speicherzellen, und unterschlagen den zur Operation nötigen Verkehr zwischen Speicher und Registern. Die Variable x_i eines LOOP-Programms wird mit der i -ten Speicherzelle einer RAM identifiziert.

LOOP-Programme berechnen Funktionen von $\mathbb{N}^k \rightarrow \mathbb{N}$ mit k beliebig. Per Konvention haben die Variablen x_1, x_2, \dots, x_k , also die Speicherzellen der RAM, die Werte der k Argumente. Alle anderen Variablen haben den Anfangswert 0. Das Resultat der Berechnung eines LOOP-Programms ist der Wert von x_0 bzw. Zelle 0.

Zu jedem LOOP-Programm P geben wir eine 'Funktion' F_P an, die alle Zellen einer RAM gemäß P verändert. F_P ist keine Funktion im Sinne einer rechtseindeutigen Abbildung! $F_P(i) = n$ bedeutet, den Wert n in Zelle i zu schreiben. Die Lesefunktion auf dem Speicher gibt den Inhalt einer Zelle zurück; $f(i) = n$ bedeutet, daß in Zelle i der Wert n steht. Die Konstruktion folgt der Definition der Sprache $\langle LP \rangle$.

- Zuweisung Addition : $P = 'x_i := x_j + c'$

$$F_P(n) = F_{x_i := x_j + c}(n) = \begin{cases} f(n) & \text{für } n \neq i \\ f(j) + c & \text{für } n = i \end{cases}$$

- Zuweisung Subtraktion : $P = 'x_i := x_j - c'$

$$F_P(n) = F_{x_i := x_j - c}(n) = \begin{cases} f(n) & \text{für } n \neq i \\ f(j) \hat{-} c & \text{für } n = i \end{cases}$$

Dabei ist $\hat{-}$ die modifizierte Subtraktion, definiert als :

$$a \hat{-} b = \begin{cases} a - b & \text{für } a \leq b \\ 0 & \text{sonst} \end{cases}$$

- Komposition zweier Programme : $P = 'P_1; P_2'$

$$F_P(n) = F_{P_1; P_2} = F_{P_2}(F_{P_1}(n))$$

- Schleife : $P = 'LOOP x_i DO P''$

$$F_P = F_{\text{LOOP} \dots} = F_{P'}(F_{P'}(\dots F_{P'}(n))) = F_{P'}^{f(i)}$$

Das Programm P' wird dabei so oft ausgeführt, wie der Wert in Zelle i vor Beginn von P angibt. Eine Veränderung des Zelleninhalts während der Ausführung hat keinen Einfluß auf die Anzahl der Wiederholungen.

Definition 21

Eine Funktion $f : \mathbb{N}^k \rightarrow \mathbb{N}$ heisst LOOP-berechenbar, falls es ein LOOP-Programm P gibt, das f im obigen Sinne berechnet.

Hinweis

Mit LOOP-Programmen lassen sich keine Endlosschleifen ausdrücken. Alle LOOP-berechenbaren Funktionen sind also total.

Erweiterungen

- Simulation von Verzweigungen
IF $x_i = 0$ THEN P END wird simuliert durch :

```

 $x_t := 1;$ 
LOOP  $x_i$  DO  $x_t := 0$  END;
LOOP  $x_t$  DO P END
Dabei sei  $x_t$  eine Variable, die in P nicht vorkommt.

```

- Die Addition $x_0 := x_1 + x_2$ wird simuliert durch
 $x_0 := x_1;$
LOOP x_2 DO $x_0 := x_0 + 1$ END

- Die Multiplikation ergibt sich aus der Addition
 $x_0 := 0;$
LOOP x_2 DO $x_0 := x_0 + x_1$

Entsprechend ist es möglich, die MOD- und DIV-Funktion zu berechnen.

WHILE-Berechenbarkeit

WHILE-Schleifen sind eine Erweiterung des LOOP-Konzeptes mit variablen Laufzeiten.

$\langle \text{WP} \rangle ::= \langle \text{WP} \rangle ; \langle \text{WP} \rangle \mid \langle \text{WHILE } x_i \neq 0 \text{ DO } \langle \text{WP} \rangle \text{ END} \mid \langle \text{Zuweisung} \rangle$

Die Anweisung WHILE $x_i \neq 0$ DO P END bewirkt die wiederholte Ausführung des Programnteils P, bis x_i den Wert 0 annimmt.

LOOP-Programme können durch WHILE - Programme simuliert werden; LOOP x_i DO P END ist äquivalent zu:

```

 $x_j := x_i$ 
WHILE  $x_j \neq 0$  DO  $x_j := x_j - 1$ ; P END
wobei  $x_j$  eine neue in P nicht benutzte Variable ist.

```

Definition 22

Eine Funktion $f : \mathbb{N}^n \rightarrow \mathbb{N}$ heisst WHILE-berechenbar, falls es ein WHILE-Programm P gibt, das f im obigen Sinn berechnet.

Satz 20

Jede WHILE-berechenbare Funktion ist Turingberechenbar

Beweis-Idee

Es genügt zu zeigen, daß die (partielle!) Speicherüberföhrungsfunktion F_P turingberechenbar ist. Desweiteren lässt sich zu P eine Turingmaschine M konstruieren derart, daß

$$z_0 \text{BIN}(f(0)) \# \text{BIN}(f(1)) \# \dots \# \text{BIN}(f(m)) \vdash_M^* f_f \text{BIN}(F_P(f)(0)) \# \text{BIN}(F_P(f)(1)) \# \dots \# \text{BIN}(F_P(f)(m))$$

gilt genau dann, wenn F_P an der Stelle f definiert ist. Hierbei sei m der grösste Index eines von P benutzten Registers.

Beweis durch Induktion über die Termstruktur von P:

Fall 1 $P = x_i := x_j \pm c$

$$\begin{aligned}
z_0 \text{BIN}(f(0)) \# \text{BIN}(f(1)) \# \dots \# \text{BIN}(f(m)) &\vdash_M^* \\
\text{BIN}(f(j)) z_1 \# \text{BIN}(f(0)) \# \dots \# \text{BIN}(f(m)) &\vdash_M^* \\
\text{BIN}(f(j) \pm c) z_2 \# \vdash_M^* \\
z_f \text{BIN}(f(0)) \# \dots \text{BIN}(f(i-1)) \# \text{BIN}(f(j) \pm c) \# \text{BIN}(f(i+1)) \dots \# \text{BIN}(f(m))
\end{aligned}$$

Fall 2 $P = P_1; P_2$

F_{P_i} werde berechnet durch M_i . Identifizieren wir den Endzustand f von M_1 mit dem Startzustand von P_2 , so berechnet die neue Maschine offenbar F_P .

Fall 3 $P = \text{WHILE } x_i \neq 0 \text{ DO } P_1 \text{ END}$

F_{P_1} werde berechnet durch M_1 . Wir erweitern M_1 durch einen neuen Startzustand, aus dem heraus geprüft wird, ob $x_i \neq 0$ ist.

Falls ja: Übergang in den Startzustand von M_1 , sonst STOP. Aus dem Endzustand von M_1 Übergang in den neuen Startzustand. \square

GOTO-Berechenbarkeit

GOTO-Programme bestehen aus einer Folge von Paaren aus Marken und Anweisungen:

$$\begin{aligned}
P = & M_1 : A_1; \\
& M_2 : A_2; \\
& \vdots \\
& M_k : A_k;
\end{aligned}$$

Die Anweisungen A_i sind von der Gestalt:

- $x_i := x_j \pm c$
- $GOTO M_i$
- $IF x_i = c THEN GOTO M_j$
- $HALT$

Semantik: Konfiguration = Zustand \times Registerinhalte

Satz 21

Jedes GOTO-Programm kann durch ein WHILE-Programm mit nur einer "echten" WHILE-Schleife simuliert werden. Jede GOTO-berechenbare Funktion ist auch WHILE-berechenbar.

Beweis

Es sei $P := M_1 : A_1; M_2 : A_2; \dots; M_k : A_k$ ein GOTO-Programm und m der grösste Index einer von P benutzten Variable. Das WHILE-Programm hat unter Benutzung des IF-THEN-Konstruktes folgendes Aussehen:

$$\begin{aligned}
&x_{m+1} := 1; \\
&\mathbf{while } x_{m+1} \neq 0 \mathbf{ do} \\
&\quad \mathbf{IF } x_{m+1} = 1 \mathbf{ THEN } A'_1; \\
&\quad \mathbf{IF } x_{m+1} = 2 \mathbf{ THEN } A'_2;
\end{aligned}$$

```

⋮
⊥ IF  $x_{m+1} = k$  THEN  $A'_k$ 
end while
END

```

Hierbei hat A'_i in Abhängigkeit von A_i folgende Gestalt:

A_i	A'_i
$x_j := x_n \pm c$	$x_j := x_n \pm c;$ $x_{m+1} := x_{m+1} + 1$
GOTO M_j	$x_{m+1} := j$
HALT	$x_{m+1} := 0$
IF $x_j = c$ THEN GOTO M_n	IF $x_j = c$ THEN $x_{m+1} := n$; ELSE $x_{m+1} := x_{m+1} + 1$;

□

Satz 22

Jede turingberechenbare Funktion ist GOTO-berechenbar.

Beweis-Idee

Gegeben DTM $M = (Z, \Sigma, \Gamma, \delta, z_1, \square, E)$ mit $\Gamma = \{a_1, a_2, \dots, a_m\}$.
Idee: Darstellung einer Konfiguration $a_{i_1} \dots a_{i_p} z_l a_{j_1} \dots a_{j_q}$ mit $z_l \in Z$ und $a_{i_\mu}, a_{j_\mu} \in \Gamma$ durch drei Zahlen in $b := m+1$ -närer Darstellung.

$$x = \text{C-Wert}(i_1 \dots i_p) := \sum_{\mu=1}^p i_\mu b^{p-\mu}$$

$$y := \text{C-Wert}(j_q \dots j_1) := \sum_{\mu=1}^q j_\mu b^\mu$$

$$\text{UND } z := l$$

Das GOTO-Programm hat nun folgende Gestalt:

$$M_1 : P_1;$$

$$M_2 : P_2;$$

$$M_3 : P_3;$$

P_1 und P_3 sind Konvertierungsprogramme, die die b -näre (De-)Codierung bewirken.
Das Simulationsprogramm P_2 hat folgende Struktur:

```

M2:   $x_4 := y \bmod b$ 
      IF  $z = 1$  AND  $a = 1$  THEN GOTO  $M_{\langle 1,1 \rangle}$ ;
      IF  $z = 1$  AND  $a = 2$  THEN GOTO  $M_{\langle 1,2 \rangle}$ ;
      ⋮
      IF  $z = k$  AND  $a = m$  THEN GOTO  $M_{\langle k,m \rangle}$ ;

```

An der Stelle $M_{\langle i,j \rangle}$ erfolgt Simulation von $\delta(z_i, a_j)$ mit anschließendem Rücksprung nach M_2 . Es treten folgende Fälle auf:

Fall 1: $\delta(z_i, a_j) = (z'_j, a'_j, R)$
 $z := i'$
 $y := (y \text{ div } b)$
 $x := b \cdot x + j'$

Fall 2: $\delta(z_i, a_j) = (z'_j, a'_j, N)$

$z := i'$

$y := b \cdot (y \text{ div } b) + j$

$x := x$

Fall 3: $\delta(z_i, a_j) = (z'_j, a'_j, L)$

$z := i'$

$y := b(b(y \text{ div } b) + j') + (x \text{ mod } b)$

$x := x \text{ div } b$

Ist z_j ein akzeptierender Endzustand, so wird an allen Stellen $M_{\langle j, i \rangle}$ ein Sprung nach M_3 ausgeführt.

Ist $\delta(z_i, a_j)$ nicht definiert, so wird bei $M_{\langle i, j \rangle}$ die Endlosschleife GOTO $M_{\langle i, j \rangle}$ installiert.

Also: WHILE-Berechenbarkeit \approx GOTO-Berechenbarkeit \approx Turing-Berechenbarkeit

2.4 Primitive und Partielle Rekursion

Die Klasse der primitiv-rekursiven Funktionen ist die kleinste Klasse von Funktionen von $\mathbb{N} \rightarrow \mathbb{N}$, die...

a.) folgende Grundfunktionen enthält:

i Die konstanten Funktionen

$$f : \begin{cases} \mathbb{N}^k \rightarrow \mathbb{N} \\ f(n_1 \dots n_k) = c \end{cases}$$

ii Die Projektion

$$\pi_i^j : \begin{cases} \mathbb{N}^j \rightarrow \mathbb{N} \\ f(n_1 \dots n_j) \rightarrow n_i \end{cases}$$

iii Die Nachfolgefunktion

$$succ : \begin{cases} \mathbb{N} \rightarrow \mathbb{N} \\ f : n \rightarrow n + 1 \end{cases}$$

b.) abgeschlossen ist unter folgenden Operationen :

i Komposition

Seien $f_1, \dots, f_m : \mathbb{N}^k \rightarrow \mathbb{N}$ und $g : \mathbb{N}^m \rightarrow \mathbb{N}$ primitiv rekursiv. Dann ist auch

$$(f_1 \dots f_m) \circ g : \begin{cases} \mathbb{N}^k \rightarrow \mathbb{N} \\ (n_1 \dots n_k) \rightarrow g(f_1(n_1, \dots, n_k), \dots, f_m(n_1, \dots, n_k)) \end{cases}$$

primitiv rekursiv.

ii primitive Rekursion

Seien $g : \mathbb{N}^k \rightarrow \mathbb{N}$, $h : \mathbb{N}^{k+2} \rightarrow \mathbb{N}$ primitiv rekursiv. Dann ist auch $f : \mathbb{N}^{k+1} \rightarrow \mathbb{N}$, definiert durch:

$$\begin{aligned} f(0, n_1, \dots, n_k) &= g(n_1, \dots, n_k) \\ f(n+1, n_1, \dots, n_k) &= h(n, f(n, n_1, \dots, n_k), n_1, \dots, n_k) \end{aligned}$$

primitiv rekursiv.

Beispiel 1

Die Summenfunktion $\text{add}: \mathbb{N}^2 \rightarrow \mathbb{N}$ ist primitiv rekursiv :

$$\begin{aligned} \text{add}(0, x) &:= \pi_1^1(x) \\ \text{add}(n + 1, x) &:= \text{succ}(\text{add}(n, x)) \end{aligned}$$

Beispiel 2

$\text{mult}: \mathbb{N}^2 \rightarrow \mathbb{N}$ ist somit auch primitiv rekursiv.

$$\begin{aligned} \text{mult}(0, x) &:= 0 \\ \text{mult}(n + 1, x) &:= \text{add}(\text{mult}(n, x), x) \end{aligned}$$

Alle primitiv-rekursiven Funktionen sind total.

Mitteilung

Die primitiv-rekursiven Funktionen sind genau die LOOP-berechenbaren.

Definition 23

Die Klasse der partiell-rekursiven Funktionen ist die kleinste Klasse von Funktionen von $\mathbb{N}^+ \rightarrow \mathbb{N}$, die

- a) die Grundfunktionen enthält
- b) abgeschlossen ist unter folgenden Operationen:
 - i) Komposition
 - ii) primitiver Rekursion
 - iii) dem μ -Operator

Mitteilung

Die partiell-rekursiven Funktionen sind genau die Turing- (bzw. WHILE-, GOTO-) berechenbaren.

Mitteilung (Kleene)

Zu jeder μ -rekursiven Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ existieren zwei primitiv-rekursive Funktionen $p, q: \mathbb{N}^{k+1} \rightarrow \mathbb{N}$ derart, daß für alle $(x_1, \dots, x_k) \in \mathbb{N}^k$ gilt: $f(x_1, \dots, x_k) = p(x_1, \dots, x_k, \mu(p)(x_1, \dots, x_k))$

2.5 Die Ackermannfunktion

Die Ackermannfunktion (Version: Hermes):

$$\begin{aligned} \text{ack}(0, y) &:= y + 1 \\ \text{ack}(x, 0) &:= \text{ack}(x - 1, 1) \\ \text{ack}(x, y) &:= \text{ack}(x - 1, \text{ack}(x, y - 1)) \end{aligned}$$

Entwicklung der Rekursion :

$$ack(x, y) = \underbrace{ack(x-1, ack(x-1, \dots ack(x-1, 1) \dots))}_{y\text{-mal}}$$

Modernisierte Version :

$$a(0, y) := a(x, 0) := 1$$

$$a(1, y) := 3y + 1$$

$$a(x, y) := \underbrace{a(x-1, a(x-1, \dots a(x-1, y) \dots))}_{y\text{-mal}}$$

Induktiv lässt sich zeigen, dass $a : \mathbb{N}^2 \rightarrow \mathbb{N}$ total ist.

Feststellung: a ist in beiden Argumenten monoton wachsend.

Für ein LOOP-Programm P , welches die Variablen x_0, x_1, \dots, x_k verwendet mit der "Semantik" $F_P : \mathbb{N}^{k+1} \rightarrow \mathbb{N}^{k+1}$ definieren wir die Funktion $f_P : \mathbb{N} \rightarrow \mathbb{N}$ vermöge

$$f_P(n) := \max\left\{\sum_{i=0}^k F_P(x_0, \dots, x_k) \mid \sum_{i=0}^k x_i \leq n\right\}$$

Lemma

Zu jedem LOOP-Programm P existiert eine Konstante k derart, dass für alle $n \geq k$ gilt: $f_P(n) < a(k, n)$.

Beweis

Durch Induktion über die Termstruktur des LOOP-Programms P .

Induktionsverankerung:

P hat die Gestalt $x_i := x_j + c$, dann gilt $f_P(n) \leq 2n + c < 3n + 1 = a(1, n)$
für $n \geq c$. Wähle also $k = c$.

Induktionsschluss:

Fall 1 P hat die Gestalt $P_1; P_2$

Nach Induktionsvoraussetzung existieren k_1, k_2 derart, daß für $n \geq \max\{k_1, k_2\} =:$

k_3 gilt $f_{P_1}(n) < a(k_1, n)$ und $f_{P_2}(n) < a(k_2, n)$

es gilt: $f_P(n) \leq f_{P_2}(f_{P_1}(n))$

also folgt:

$$\begin{aligned} f_P(n) &< a(k_2, f_{P_1}(n)) \\ &\leq a(k_2, a(k_1, n)) \\ &\leq a(k_3, a(k_3, n)) \\ &\leq a(k_3, \underbrace{a(k_3, \dots a(k_3, n) \dots)}_{n\text{-mal}}) = a(k_3 + 1, n) \end{aligned}$$

wähle also $k := \max(k_3 + 1, 2)$.

Fall 2 P hat die Gestalt LOOP x_i DO P' END.

Nach Induktionsvoraussetzung existiert eine Konstante k' derart, daß für alle $n \geq k'$ gilt: $f_{P'}(n) < a(k', n)$.

Wegen $F_P(n_0 \dots n_k) = F_{P'}^{n_i}(n_0 \dots n_k)$ gilt:

$$\begin{aligned} f_P(n) &\leq f_{P'}^n(n) \\ &< \underbrace{a(k', a(k', \dots, a(k', n) \dots))}_n = a(k' + 1, n) \text{ für } n \geq k' \end{aligned}$$

Wähle also $k := k' + 1$. \square

Satz 23

Die Ackermannfunktion a ist nicht LOOP-berechenbar.

Beweis

Wenn a LOOP-berechenbar ist, dann auch $g(n) := a(n, n)$.

Sei also P ein LOOP-Programm zur Berechnung von g . Dann existiert eine Konstante k derart, daß für alle $n \geq k$ gilt: $f_P(n) < a(k, n)$. Für $n = k$ ergibt das aber den Widerspruch $g(k) \leq f_P(k) < a(k, k) = g(k)$. \square

Zur Berechenbarkeit von a :

Problem: Berechnung mit Hilfe einer konstanten Anzahl von Registern.

Lösung: Speichern von mehreren Zahlen auf einem Register. Dazu wird eine bijektive Abbildung von \mathbb{N}^2 nach \mathbb{N} benötigt, also eine Kodierung zweier Zahlen in einer einzelnen. Dies wird realisiert durch die Fkt. c mit $c(x, y) := 2^{x+y} + x$. Die Umkehrfunktionen werden c_1 und c_2 genannt, es gilt $n = c(c_1(n), c_2(n))$.

Input: x, y

```

Init(Stack)
Push (x, Stack)
Push (y, Stack)
while Size (Stack)  $\neq$  1 do
  y := Pop(Stack)
  x := Pop(Stack)
  if  $x = 0$  or  $y = 0$  then Push (1, Stack)
  elif  $(x = 1)$  then Push ( $3 * y + 1$ , Stack)
  else Loop y do
    Push (y-1, Stack)
  end;
  Push (y, Stack)
end
end while
Result := Pop(Stack);
Output := (Result)

```

Der gesamte Kellerinhalt kann vermittle c in einer einzelnen Zahl kodiert werden, $N := c(n_1, c(n_2, c(\dots c(n_k, 0)) \dots))$. Die Kelleroperationen werden wie folgt simuliert :

Init(Stack): $n:=0$
 Push (x,Stack): $n:= c(x,n)$
 Pop (Stack): Result:= $c_1(n)$
 $n:= c_2(n)$
 Return Result

Satz 24

Es gibt totale (WHILE-)berechenbare Funktionen, die nicht LOOP-berechenbar sind.

2.6 Halteproblem, Unentscheidbarkeit, Reduzierbarkeit

Definition 24

a) Eine Sprache $A \in \Sigma^*$ heisst *entscheidbar*, falls die charakteristische Funktion

$$\chi_A : \begin{cases} \Sigma^* \rightarrow \{0, 1\} \\ w \mapsto \begin{cases} 1, \text{ falls } w \in A \\ 0, \text{ falls } w \notin A \end{cases} \end{cases}$$

berechenbar ist.



Abbildung 12:

b) $A \in \Sigma^*$ heisst *semi-entscheidbar*, falls die eingeschränkte charakteristische Funktion

$$\chi_A : \begin{cases} \Sigma^* \rightsquigarrow \{1\} \\ w \mapsto \begin{cases} 1, \text{ falls } w \in A \\ \perp, \text{ falls } w \notin A \end{cases} \end{cases}$$

berechenbar ist.

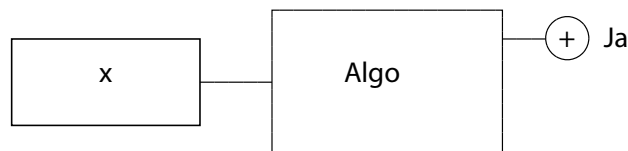


Abbildung 13:

Satz 25

Eine Sprache $A \in \Sigma^*$ ist genau dann entscheidbar, wenn sowohl A als auch ihr Komplement $\Sigma^* \setminus A =: \bar{A}$ semi-entscheidbar sind.

Beweis

"⇒"
 A entscheidbar $\Rightarrow A$ semi-entschiedbar
 \Downarrow
 $\Sigma^* \setminus A$ entscheidbar $\Rightarrow \Sigma^* \setminus A$ semi-entschiedbar
 "⇐"

Seien χ'_A und $\chi'_{\Sigma^* \setminus A}$ berechenbar durch zwei TM M_1 und M_2 . Dann entscheidet folgender Algorithmus A:

```

Input(x);
while Result undefined do
  ⌈ Simuliere einen Schritt von  $M_1$ ;
  if  $x \xrightarrow{M_1} 1$  then Result:=1
  Simuliere einen Schritt von  $M_2$ ;
  ⌋ if  $x \xrightarrow{M_2} 1$  then Result:=0
  
```

Output(Result). \square

Definition 25

Eine Sprache $A \in \Sigma^*$ heisst rekursiv aufzählbar, falls $A = \emptyset$, oder falls es eine totale und berechenbare Funktion $f : \mathbb{N} \rightarrow \Sigma^*$ gibt, derart, daß $A = \{f(0), f(1), f(2), \dots\} = f(\mathbb{N})$. Man sagt, f zählt A auf.

Satz 26

Eine Sprache ist genau dann rekursiv aufzählbar, wenn sie semi-entscheidbar ist.

Beweis

"⇒"
 Ist A rekursiv aufzählbar, d.h. gilt $f(\mathbb{N}) = A$ für eine totale, berechenbare Funktion f , so wird A semi-entschieden durch folgende Turingmaschine:

```

Input(x)
n:=0
while Result undefined do
  if  $f(n) = x$  then Result:=1 else n:=n+1
  
```

Output(Result) \square

"⇐"

Sei A nun semi-entscheidbar durch eine TM M . Berechnung einer Funktion f durch eine TM M mit zweidimensionalem Arbeitsband (das stellt keine Erweiterung bisheriger TM dar, da eine Bijektion von \mathbb{N}^2 nach \mathbb{N} existiert). In M' liegt zu Beginn der $m+1$ -ten Phase folgende Situation vor :

	Zähler	
'Spur 1'	v_1	m Schritte bearbeitet
'Spur 2'	v_2	m-1 Schritte bearbeitet
'Spur 3'	v_3	m-2 Schritte bearbeitet
'Spur 4'		
⋮	⋮	
'Spur m'	v_m	1 Schritt bearbeitet

Sollte M das Wort v_j in höchstens $m+1-j$ Schritten akzeptiert haben, so ist die Spur j "geschlossen".

Phase $m+1$: Ergänze jede nicht geschlossene Simulation um einen Schritt. Eröffne auf Spur $m+1$ die Simulation von M auf $v_m + 1$ und führe einen Schritt aus. Wird dabei ein $v_j, j \in m + 1$ auf einer nicht geschlossenen Spur akzeptiert, so schliesse die Spur und führe $t := t + 1$ aus.

Wird dabei $t=n$ so gebe dieses v_j als Ausgabe aus und stoppe. Sonst nächste Phase.

Das so konstruierte $f = f_{n'}$ ist injektiv. Aber das so konstruierte $f = f_{n'}$ ist nur genau für unendliches A total!

Zusammenfassung

Für beliebiges $A \in \Sigma^*$ gelten folgende Äquivalenzen:

- a)
- \Leftrightarrow A ist semi-entscheidbar
 - \Leftrightarrow A ist rekursiv aufzählbar
 - \Leftrightarrow χ'_A ist berechenbar
 - \Leftrightarrow A wird von einer deterministischen oder nichtdeterministischen TM M akzeptiert, $A = T(M)$
 - \Leftrightarrow A ist Definitionsbereich einer partiellen, berechenbaren Funktion $f : \Sigma^* \rightarrow \Pi^*$, d.h. A hat eine Darstellung $A = f^{-1}(\Pi^*)$
 - \Leftrightarrow A ist Wertebereich einer partiellen, berechenbaren Funktion $g : \Pi^* \rightarrow \Sigma^*$, d.h. A hat eine Darstellung $A = g(\Pi^*)$
- b)
- \Leftrightarrow A ist entscheidbar
 - \Leftrightarrow A ist endlich oder aufzählbar durch eine streng monotone, totale und berechenbare Abbildung $h : \mathbb{N} \rightarrow \Sigma^*$
 - \Leftrightarrow χ_A ist berechenbar
 - \Leftrightarrow A wird von einer deterministischen oder nichtdeterministischen TM M akzeptiert, die auf allen Eingaben hält. $A = T(M)$
 - \Leftrightarrow A ist Urbild eines Bildwertes einer totalen, berechenbaren Funktion $f : \Sigma^* \rightarrow \Pi^*$, d.h. A hat eine Darstellung $A = f^{-1}(1)$
 - \Leftrightarrow A ist Wertebereich einer totalen, berechenbaren Funktion $g : \Pi^* \rightarrow \Sigma^*$, d.h. A hat eine Darstellung $A = g(\Pi^*)$

Kodierung und Selbstanwendung

Im folgenden wird eine TM M , genauer : ihre Transitionen, binär kodiert. Dazu müssen das Bandalphabet und die Zustandsmenge geordnet sein, sei also

$$M = (Z = \{z_0, z_1, \dots, z_m\}, \Sigma = \{a_1, a_2, \dots, a_n\}, \Gamma = \Sigma \cup \{a_{n+1}\}, \delta, z_0, \square, \{z_e\})$$

Nun wird jeder Transition von M ein Wort $w_{i,j,i',j',y}$ zugeordnet, daß sie beschreibt :
 $(z_i, a_j, z'_i, a'_j, y) \in \delta \rightarrow w_{i,j,i',j',y} := \#\#\text{BIN}(i)\#\text{BIN}(j)\#\text{BIN}(i')\#\text{BIN}(j')\#\text{BIN}(m)$

$$\text{mit } m = \begin{cases} 0, y = L \\ 1, y = R \\ 2, y = N \end{cases}$$

Die Kodierung von $\{0, 1, \#\}$ in $\{0,1\}$ erfolge durch :

- $0 \rightarrow 00$
- $1 \rightarrow 01$
- $\# \rightarrow 11$

Auf diese Weise kann jeder TM ein eindeutiges binäres Wort zugeordnet werden. Dieses Wort kann als Eingabe einer zweiten TM M' dienen, die sich dann beispielsweise wie die kodierte TM verhält (somit wäre M' eine programmierbare TM). Durch die Kodierung von TM ist allerdings insbesondere beweisbar, daß nicht jede (formal formulierbare) Aufgabe von einer TM gelöst werden kann.

Sei v Code der TM M_v . Unter einem *Problem* versteht man im Kontext der Berechenbarkeit auch eine Menge. Das *spezielle Halteproblem* ist die Menge aller TM, die beim Abarbeiten ihrer eigenen Kodierung irgendwann in einen Endzustand übergehen.

Definition/Satz

Das spezielle Halteproblem, formal: $K := \{w \in \{0, 1\}^* | M_w \text{ angesetzt auf } w \text{ hält}\}$, ist unentscheidbar.

Beweis

Angenommen K ist entscheidbar, dann ist χ_K berechenbar durch eine TM M .

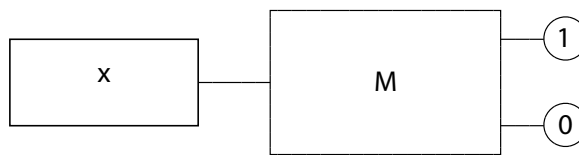


Abbildung 14:

Ergänzung von M zu M' folgender Art: Bei Ausgabe 1 Übergang in eine Endlosschleife.

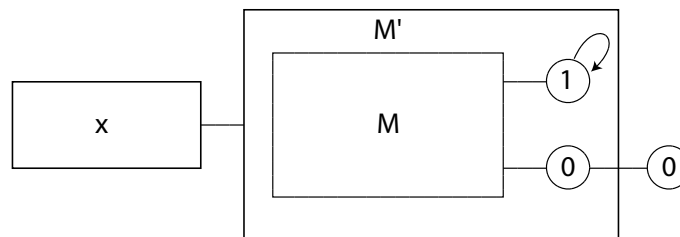


Abbildung 15:

Ist $w' = \langle M' \rangle$ die Codierung von M' , so gilt:

$$\begin{aligned}
M' \text{ angesetzt auf } w' \text{ hält} &\Leftrightarrow M \text{ gibt bei Eingabe } w' \text{ eine 0 aus} \\
&\Leftrightarrow \chi_K(w') = 0 \\
&\Leftrightarrow w' \notin K \\
&\Leftrightarrow M' \text{ angesetzt auf } \langle M' \rangle = w' \text{ hält nicht.}
\end{aligned}$$

M' angesetzt auf w' hält also genau dann, wenn M' angesetzt auf w' nicht hält. Widerspruch! Also kann K nicht entscheidbar sein. \square

Hinweis: K ist semi-entscheidbar

Entscheidbarkeitsbeweis durch Reduktion

Definition 26

Eine formale Sprache $A \subseteq \Sigma^*$ heißt *reduzierbar* auf eine Sprache $B \subseteq \Pi^*$ (in Zeichen $A \leq B$), wenn es eine totale und berechenbare Funktion $f : \Sigma \rightarrow \Pi^*$ mit $A = f^{-1}(B)$ gibt; wenn also gilt: $\forall x \in \Sigma^*: x \in A \Leftrightarrow f(x) \in B$.

Beispiel: für jede entscheidbare Sprache $A \subseteq \Sigma^*$ ist χ_A eine Reduktion von A auf die Menge $\{1\}$

Lemma

Gilt $A \leq B$ und ist B entscheidbar (bzw. semi-entscheidbar), so ist auch A entscheidbar (bzw. semi-entscheidbar).

Beweis

Ist $A = f^{-1}$, so folgt $\chi_A = \chi_B \circ f$
 Sind also χ_B und f berechenbar, so auch χ_A .
 Ist nur das partielle χ'_B berechenbar so bleibt χ'_A berechenbar. \square

Definition 27

Das allgemeine Halteproblem ist die Menge $H := \{w\#x \mid M_w \text{ hält auf Eingabe } x\}$

Satz 27

H ist nicht entscheidbar

Beweis

Durch Nachweis von $K \leq H$:
 wähle $f : \begin{cases} \{0, 1\}^* \rightarrow \{0, 1, \#\}^* \\ w \rightarrow w\#w \end{cases}$
 Dann ist $f^{-1}(H) = K!$ \square

Hinweis: H ist semi-entscheidbar

Definition 28

Das Halteproblem auf leerem Band ist die Menge
 $H_0 := \{x \mid M_x \text{ hält auf leerem Band}\}$

Satz 28

H_0 ist unentscheidbar.

Beweis

Durch Reduktion von H auf H_0

Gegeben $w\#x$, frage hält M_w auf x . Zu x ist eine TM $A(x)$ zu konstruieren, die angesetzt auf das leere Band das Wort x schreibt. Die zu w und x zu konstruierende TM $M(w, x)$ besteht dann in der Anwendung von $A(x)$, gefolgt von M_w . Dann ist die Abbildung $w\#x \mapsto f(w\#x) := \langle M(w, x) \rangle$ berechenbar und es gilt:

$$\forall w\#x : M_w \text{ hält auf } x \Leftrightarrow M(w, x) \text{ hält auf } \lambda.$$

Also folgt $H \leq H_0$ und damit die Unentscheidbarkeit von H_0 . \square

Satz von Rice

Sei \mathcal{R} die Menge aller (turing-) berechenbarer Funktionen. Sei weiter $\mathcal{S} \subseteq \mathcal{R}$ eine nicht-triviale Teilmenge von \mathcal{R} , d.h. $\mathcal{S} \neq \emptyset$, $\mathcal{S} \neq \mathcal{R}$.

Dann ist die Sprache $C(\mathcal{S}) := \{ w \mid \text{die von } M_w \text{ berechnete Funktion liegt in } \mathcal{S} \}$ unentscheidbar.

Beweis

Sei Ω die Funktion mit leerem Definitionsbereich.

Fall 1: $\Omega \in \mathcal{S}$. Dann Reduktion von H_0 auf $C(\overline{\mathcal{S}}) := (\mathcal{R} \setminus \mathcal{S})$. Wegen $\mathcal{S} \neq \mathcal{R}$ existiert eine TM Q , deren berechnete Funktion q nicht in \mathcal{S} liegt.

Zu gegebenem w betrachten wir die Maschine $M(w)$, die auf beliebiger Eingabe y zunächst die Maschine M_w auf der leeren Eingabe simuliert und nur wenn M_w hält, Q auf y simuliert. Die von $M(w)$ berechnete Funktion ist dann Ω , wenn M_w nicht hält und q anderenfalls.

$$w \in H_0 \Leftrightarrow \langle M(w) \rangle \notin C(\mathcal{S})$$

Da offenbar $w \mapsto M(w)$ total und berechenbar ist folgt $\overline{H_0} \leq C(\mathcal{S})$; aus der Unentscheidbarkeit von H_0 folgt dann die Behauptung. \square

Fall 2: $\Omega \in \mathcal{R} \setminus \mathcal{S}$. In diesem Fall ergibt sich analog $H_0 \leq C(\mathcal{S})$ \square

Korollar

Folgende Fragestellungen bezüglich der Leistung von TM sind unentscheidbar:

- a.)
 - Die berechnete Funktion ist konstant
 - Die berechnete Funktion ist total
 - Die berechnete Funktion ist primitiv rekursiv
 - Die berechnete Funktion ist ein Polynom
- b.)
 - Die akzeptierte Sprache ist leer
 - Die akzeptierte Sprache ist endlich
 - Die akzeptierte Sprache ist "total": Σ^*
 - Die akzeptierte Sprache ist regulär, kontextfrei und kontextsensitiv

Bemerkungen

- 1.) Die Fragestellung, ob die akzeptierte Sprache TYP-0 ist, ist trivial und daher entscheidbar.
- 2.) Es gibt einen "zweiten" Satz von Rice, der genau die Teilmenge $\mathcal{S} \subseteq \mathcal{R}$ charakterisiert, für die $C(\mathcal{S})$ semi-entscheidbar ist.
So ist beispielsweise die Frage semi-entscheidbar, ob die berechnete Funktion von Ω verschieden ist, nicht aber, ob sie mit Ω übereinstimmt.
- 3.) Es gibt Probleme, die nachweislich schwieriger sind als das unentscheidbare Halteproblem: So lässt sich H auf das Äquivalenzproblem für TM reduzieren, nicht aber umgekehrt.

2.7 Das Post'sche Korrespondenzproblem

Definition 29

Zu einem Alphabet Σ ist das Post'sche Korrespondenzproblem definiert als die Menge :

$$\text{PCP} := \{ \langle (x_1, y_1), (x_2, y_2), \dots, (x_k, y_k) \rangle \mid k \geq 1; x_i, y_i \in \Sigma^*; \exists i_1, \dots, i_n \leq k \text{ mit} \\ x_{i_1} x_{i_2} \dots x_{i_n} = y_{i_1} y_{i_2} \dots y_{i_n} \}$$

Beispiel 1

$\langle (1, 101), (10, 00), (011, 11) \rangle \in \text{PCP}$: Wähle $c_1 = 1, c_2 = 3, c_3 = 2, c_4 = 3$.

Beispiel 2

$\langle (1, 101), (10, 1), (01, 0), (0, 100) \rangle$ ist nicht lösbar.

Hinweise

- Das PCP ist semi-entscheidbar für beliebiges Σ .
- Das PCP ist entscheidbar für $|\Sigma| = 1$
- Alternative Darstellung ist:

$$\text{PCP} := \{ \langle g, h, \Pi \rangle \mid h, g : \Pi \rightarrow^* \text{ sind Homomorphismen } \exists x \in \Pi^+ : h(x) = g(x) \}$$

Im Weiteren wird die Unentscheidbarkeit des PCP's gezeigt. Dazu benötigt man zunächst das modifizierte Korrespondenzproblem :

$$\text{MPCP} := \{ \langle (x_1, y_1), \dots, (x_k, y_k) \rangle \mid k \geq 1, x_i, y_i \in \Sigma^*, \exists i_2, \dots, i_n \text{ mit} \\ x_1 x_{i_2} \dots x_{i_n} = y_1 y_{i_2} \dots y_{i_n} \}$$

Die Modifikation besteht darin, daß eine Lösung mit dem Paar (x_1, y_1) beginnen muß.

Lemma: MPCP \leq PCP

Beweis

Es seien $\$$ und $\#$ zwei neue, nicht in Σ enthaltene Symbole. Wir betrachten die Homomorphismen φ_l und $\varphi_r : \Sigma^* \rightarrow (\Sigma \cup \{\#\})^*$, definiert durch $\varphi_l(a) := \#a$ bzw. $\varphi_r(a) := a\#$.

Für $w \in \Sigma^*$ gilt dann: $\varphi_l(w)\# = \#\varphi_r(w)$

Die reduzierende Abbildung f ist nun erklärt vermöge:

$$f(\langle (x_1, y_1), \dots, (x_k, y_k) \rangle) := \langle (\varphi_l(x_1)\#, \varphi_l(y_1)), (\varphi_r(x_1), \varphi_l(y_1)), (\varphi_r(x_2), \varphi_l(y_2)), \dots, (\varphi_r(x_k), \varphi_l(y_k)), (\$, \#\$) \rangle =: \langle (x'_1, y'_1), \dots, (x'_{k+2}, y'_{k+2}) \rangle$$

Für "falsche" Eingaben sei f die Identität.

f ist total und berechenbar.

f ist auch Reduktion: $\mathcal{R} \in \text{MPCP} \Leftrightarrow f(\mathcal{R}) \in \text{PCP}$:

" \Rightarrow " $\mathcal{R} \in \text{MPCP} \Rightarrow \exists i_2, \dots, i_n \leq k+2$ mit $x_1, x_{i_2} \dots x_{i_n} = y_1, y_{i_2} \dots y_{i_n}$, dann gilt:

$$\begin{aligned} x'_1 x'_{i_2+1} \dots x'_{i_n+1} x'_{k+2} &= \# \phi_r(x_1 x_{i_2} \dots x_{i_n}) \$ = \phi_l(x_1 x_{i_2} \dots x_{i_n}) \# \$ = \\ &= \phi_l(y_1 \dots y_{i_n}) \# \$ = y'_1 y'_{i_2+1} \dots y'_{i_n+1} y'_{k+2}. \end{aligned}$$

Also ist $(1, i_2 + 1, \dots, i_n + 1, k + 2)$ Lösung von $f(\mathcal{R})$.

" \Leftarrow " Ist $f(\mathcal{R}) \in \text{PCP}$, so existiert eine Lösung (i_1, \dots, i_n)

$$\text{mit } x'_{i_1} x'_{i_2} \dots x'_{i_n} = y'_{i_1} y'_{i_2} \dots y'_{i_n}. \quad \square$$

"Claims":

1.) $i_1 = 1$

2.) $i_n = k + 2$

3.) Ist für $1 < j < n$ bereits $i_j = k + 2$, so ist auch (i_1, \dots, i_j) schon Lösung!

4.) Wähle eine minimale Lösung, also $i_2, i_3 \dots i_{n-1} \leq k + 1$.

$$\text{Dann ist } \varphi_l(x_1 x_{i_2-1} \dots x_{n-1}) \# \$ = \varphi_l(y_1 y_{i_2-1} \dots y_{n-1}) \# \$ x'_1 x'_{i_2} \dots x'_{n-1} x'_{k+2} = y'_1 y'_{i_2} \dots y'_{n-1} y'_{k+2}.$$

Aus der Injektivität von ϕ_l folgt $x_1 x_{k-1} \dots x_{i_{n-1}-1} = \varphi_l$ folgt $y_1 y_{k-1} \dots y_{i_{n-1}-1}$, also folgt $\mathcal{R} \in \text{MPCP}$

Lemma $H_0 \leq \text{MPCP}$

Beweis

Zu gegebener TM $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ setzen wir $\Pi := Z \cup \Gamma \cup \{\#\}$, wobei $\#$ ein neues nicht in $Z \cup \Gamma$ enthaltene Symbol sei. Ziel ist jetzt die Konstruktion eines Kandidaten R aus MPCP über dem Alphabet Π , so daß gilt: $R \in \text{MPCP} \Leftrightarrow \langle M \rangle \in H_0$

Das Wortpaar Nummer 1 ist: $(\#, \#z_0\#)$

Weitere Wortpaare:

- Kopierregeln: (a, a) für beliebiges $a \in \Gamma \cup \{\#\}$

- Überführungsregeln :

$(za, z'c)$	falls	$\delta(z, a) = (z', c, N)$	
(za, cz')	falls	$\delta(z, a) = (z', c, R)$	
$(bza, z'bc)$	falls	$\delta(z, a) = (z', c, L)$, für beliebiges $b \in \Gamma$
$(\#za, \#z'\square c)$	falls	$\delta(z, a) = (z', c, L)$	
$(z\#, z'c\#)$	falls	$\delta(z, \square) = (z', c, N)$	
$(z\#, cz'\#)$	falls	$\delta(z, \square) = (z', c, R)$	
$(bz\#, z'bc\#)$	falls	$\delta(z, \square) = (z', c, L)$, für beliebiges $b \in \Gamma$
- Löschende Regeln : $(az_e, z_e), (z_e a, z_e)$, für alle $a \in \Gamma, z_e \in E$
- Abschlussregeln : $(z_e \# \#, \#)$ für $z_e \in E$

□

Korollar

PCP (und MPCP) sind unentscheidbar

Satz 29Das PCP ist schon für zweielementiges Σ unentscheidbar.Für einelementiges Σ ist das PCP entscheidbar.Offenbar ist das PCP semi-entscheidbar. Aus $H_0 \leq PCP$ folgt die Semi-Entscheidbarkeit von $H_0(H, K, \dots)$.**2.8 Anwendung des PCP****Satz 30**

Das Schnittproblem, gegeben durch :

$$L_{\cap \neq \emptyset} := \{ \langle G_1, G_2 \rangle \mid L(G_1), L(G_2) \in \text{CFL}, L(G_1) \cap L(G_2) \neq \emptyset \}$$

ist nicht entscheidbar.

BeweisideeDurch Nachweis von $\text{PCP} \leq L_{\cap \neq \emptyset}$. Zu gegebenem PCP-Kandidaten $R = \langle (x_1, y_1), \dots, (x_k, y_k) \rangle$ über Σ konstruieren wir die Grammatiken $G_1 = (\{S_1\}, \Sigma \cup \{a_1, \dots, a_k\}, P_1, S_1)$ mit

$$P_1 = \{S_1 \rightarrow a_i S_1 x_i \mid 1 \leq i \leq k\} \cup \{S_1 \rightarrow a_i x_i \mid 1 \leq i \leq k\}$$

und $G_2 = (\{S_2\}, \Sigma \cup \{a_1, \dots, a_k\}, P_2, S_2)$ mit

$$P_2 = \{S_2 \rightarrow a_i S_2 y_i \mid 1 \leq i \leq k\} \cup \{S_2 \rightarrow a_i y_i \mid 1 \leq i \leq k\}$$

Claim: $R \in \text{PCP} \Leftrightarrow f(R) := \langle G_1, G_2 \rangle \in L_{\cap \neq \emptyset}$ "⇒" Wenn $x_{i_1} x_{i_2} \dots x_{i_n} = y_{i_1} y_{i_2} \dots y_{i_n}$ ist, dann liegt $a_{i_n} a_{i_{n-1}} \dots a_{i_1} x_{i_1} \dots x_{i_n} = a_{i_n} \dots a_{i_1} y_{i_1} \dots y_{i_n}$ sowohl in $L(G_1)$ als auch in $L(G_2)$."⇐" Ist $z \in L(G_1) \cap L(G_2)$, so gilt $z \in \{a_1 \dots a_n\}^* \Sigma^*$ und z hat die Gestalt $z = \{a_{j_1}, \dots, a_{j_k}\} w$ für gewisse $a_{j_n} \in \{a_1 \dots a_m\}^{m \geq 1}$ und $w \in \Sigma^*$. Aus der Konstruktion von G_1 und G_2 folgt $x_{j_n} \dots x_{j_m} = w = y_{j_m} \dots y_{j_1}$ also $R \in \text{PCP}$.

Korollar

Das Schnittproblem ist sogar unentscheidbar für deterministische lineare kontextfreie Sprachen.

Korollar

Das Äquivalenzproblem für kontextfreie Sprachen ist unentscheidbar.

Bemerkung

Das Äquivalenzproblems für deterministisch kontextfreie Sprachen ist entscheidbar.

Korollar

Das Leerheitsproblem für TYP-1 Sprachen ist entscheidbar.

Satz 31

Es ist unentscheidbar, ob eine beliebige kontextfreie Grammatik eine reguläre Sprache erzeugt.

3 Komplexitätstheorie

3.1 Klassen

Definition 30

Für eine Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ sei $DTIME(f(n))$ die Mengen aller formalen Sprachen $L \subseteq \Sigma^*$, die von einer Turingmaschine M erkannt werden derart, daß für jedes $x \in \Sigma^*$, die TM M auf x angesetzt höchstens $f(|x|)$ Schritte ausführt.

Sei dabei $time_M(x) :=$ die Anzahl der Schritte von M auf x .

Die Klasse P ist definiert vermöge $P := \bigcup_{k \geq 1} DTIME(n^k)$

Satz 32

Die Elemente von P sind LOOP-berechenbar.

Beweis-Idee

Die Simulation einer beliebigen Turingmaschine M durch ein WHILE-Programm hat folgende Gestalt:

while not Haltezustand **do**

$P_M := \{\text{LOOP-Programm zur Simulation eines Schrittes}\}$

end while

Ist M zeitbeschränkt durch eine Funktion f , so ist M äquivalent zu :

$y := f(x);$

LOOP y **DO**

$P_M := \{\text{LOOP-Programm zur Simulation eines Schrittes}\}$

END

Ist f LOOP-berechenbar, so auch die von M berechnete Funktion.

Problem

Wie definiert man Zeitbedarf einer NTM ?

Einige Möglichkeiten sind :

$$M1: time_M(x) := \begin{cases} \text{Länge der kürzesten akzeptierenden Rechnung von M} \\ \text{auf } x, \text{ wenn } x \in T(M) \\ 0 \text{ sonst} \end{cases}$$

M2: $time_M(x) :=$ Länge der längsten Rechnung von M auf x

M3: $time_M(x) :=$ Alle Rechnungen gleich lang := Laufzeit

Für "vernünftige" Zeitschranken sind diese Maße äquivalent. Vernünftig sind u.a. Polynome, Exponentialfunktionen,...

$NTIME(f(n))$ sei die Mengen aller Sprachen L für die eine NTM M existiert mit $L=T(M)$ und $time_M(x) \leq f(|x|)$ für alle x.

Wir setzen $NP := \bigcup_k NTIME(n^k)$.

Eine der bekanntesten offenen Fragen der (theoretischen) Informatik ist die (Un-?)Gleichung: $P=NP$?

Zur Einordnung von P und NP:

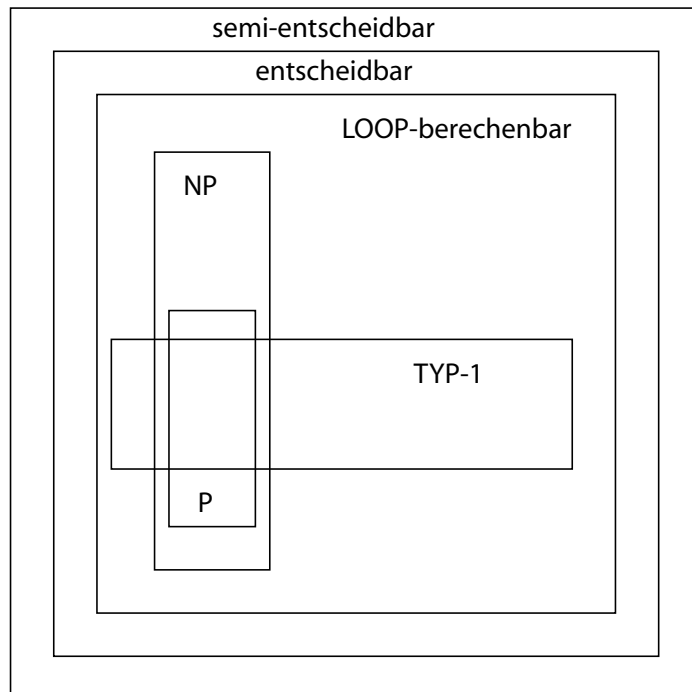


Abbildung 16: Übersicht

3.2 NP-Vollständigkeit

Definition 31

eine Sprache $A \subseteq \Sigma^*$ heisst *polynomiell-reduzierbar* auf eine Sprache $B \subseteq \Pi^*$ (i.Z.: $A \leq_P B$) genau dann, wenn es eine totale, in polynomialer Zeit berechenbare Funktion $f : \Sigma^* \rightarrow \Pi^*$ gibt, so daß gilt $\forall x \in \Sigma^* : x \in A \Leftrightarrow f(x) \in B$.

Hinweis

- 1) $A \leq_P B \Rightarrow A \leq B$
- 2) \leq_P ist transitiv.

Lemma

Wenn $A \leq_P B$ und $B \in P$ (bzw. $B \in NP$), so gilt $A \in P$ (bzw. $A \in NP$).

Beweis

Sei die Reduktion f in $O(p(n))$ Schritten berechenbar durch eine TM M_f und B in $O(q(n))$ Schritten entscheidbar durch eine TM M , p und q Polynome. Wie schon bekannt, gilt $\chi_B \circ f$. Die Berechnung von χ_A erfordert höchstens $p(|x|) + q(p(|x|))$ Schritte. $p + q \circ p$ ist aber polynomiell beschränkt.

Ist M nichtdeterministisch, so auch die Hintereinanderausführung von M_f und M , die aber in diesem Fall polynomiell durch $p + q \circ p$ zeitbeschränkt ist. \square

Definition 32 a) Ein Problem A heisst NP-hart, falls für alle Sprachen $L \in NP$ $L \leq_P A$ gilt.

b) A heisst NP-vollständig, wenn A NP-hart ist und $A \in NP$ gilt.

Hinweis

Ist A NP-hart und $A \leq_P B$, so ist auch B NP-hart.

Satz 33

Ist A NP-vollständig, so gilt: $A \in P \Leftrightarrow P = NP$.

Beweis

" \Rightarrow " $NP \leq_n^P A, A \in P \Rightarrow NP \subseteq P$, mit $P \subseteq NP$ folgt Gleichheit.

" \Leftarrow " $A \in NP, P = NP \Rightarrow A \in P$. \square

Situation für $P \neq NP$

Definition 33

Das Erfüllbarkeitsproblem der Aussagenlogik : *SAT* (von engl. 'satisfiability'), ist die Frage, ob eine gegebene aussagenlogische Formel F erfüllbar ist, ob es also eine $\{0, 1\}$ -wertige Belegung der in F verwendeten Variablen derart gibt, daß F zu 1 ausgewertet wird.

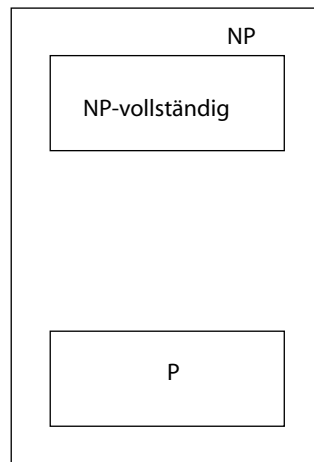


Abbildung 17: Übersicht

Dazu

$\langle \text{Formel} \rangle ::= 0 \mid 1 \mid \langle \text{Variable} \rangle \mid \neg \langle \text{Formel} \rangle \mid (\langle \text{Formel} \rangle \wedge \langle \text{Formel} \rangle) \mid (\langle \text{Formel} \rangle \vee \langle \text{Formel} \rangle)$
 $\langle \text{Literal} \rangle ::= \langle \text{Variable} \rangle \mid \neg \langle \text{Variable} \rangle$
 $\langle \text{Variable} \rangle ::= x_1 \mid x_2 \mid \dots \mid x_n \mid \dots$

$\mathcal{V} := \{x_1, x_2, \dots, x_n, \dots\}$ sei die Menge aller Variablen, \mathcal{F} die Menge aller Formeln.

Eine *Belegung* ist eine Abbildung $\phi : \mathcal{V} \rightarrow \{0, 1\}$. Der Definitionsbereich von ϕ wird induktiv erweitert auf ganz \mathcal{V} vermöge :

$$\begin{aligned}
 \phi(0) &:= 0 \\
 \phi(1) &:= 1 \\
 \phi(\neg F) &:= 1 - \phi(F) \\
 \phi((F \wedge G)) &:= \phi(F) \cdot \phi(G) \\
 \phi((F \vee G)) &:= \phi(F) + \phi(G) - \phi(F) \cdot \phi(G)
 \end{aligned}$$

Also: $\text{SAT} := \{F \in \mathcal{F} \mid \exists \phi \text{ mit } \phi(F) = 1\}$

Satz 34

SAT ist NP-vollständig

Teil 1: Bei Eingabe F kann deterministisch ermittelt werden, welche Variablen x_{i_1}, \dots, x_{i_k} in F vorkommen. Nichtdeterministisch wird nun für jedes x_{i_j} der erfüllende Wahrheitswert $\phi(x) \in \{0, 1\}$ geraten. Abschliessend wird deterministisch $\phi(F)$ ausgerechnet.
d.h.: Wenn $F \in \text{SAT}$, so lässt sich das nicht-deterministisch in Polynomzeit ermitteln.

Teil 2: SAT ist NP-hart.

Sei $L \in NP$. Es gibt dann eine polynomiell zeit-beschränkte NTM M , die L akzeptiert: $L=T(M)$. M sei also laufzeitbeschränkt durch das Polynom $p(n)$ und wir nehmen an, daß die Berechnung von M nach genau $p(|x|)$ Schritten stoppt für eine Eingabe $x = x_1x_2 \dots x_n \in \Sigma^*$

Das Arbeitsalphabet von M sei $\Gamma = \{a_1, a_2 \dots a_l\}$ und die Zustandsmenge $Z = \{z_1 \dots z_k\}$.

Die zu konstruierende Formel F betrifft die folgenden Variablen:

Variable	Indizes	Bedeutung
$Z_{t,j}$	$0 \leq t \leq p(n)$ $j \leq k$	$Z_{t,j} = 1 \Leftrightarrow$ nach t Schritten (bei Eingabe x) befindet sich M im Zustand z_j
$P_{t,i}$	$0 \leq t \leq p(n)$ $-p(n) \leq i \leq p(n)$	$P_{t,i} = 1 \Leftrightarrow$ nach t Schritten (bei Eingabe x) befindet sich der Kopf von M auf Position i
$B_{t,i,j}$	$0 \leq t \leq p(n)$ $-p(n) \leq i \leq p(n)$ $1 \leq j \leq l$	$B_{t,i,j} = 1 \Leftrightarrow$ nach t Schritten (bei Eingabe x) steht in Bandzelle i das Zeichen a_j .

Die zu einer Eingabe $x \in \Sigma^*$ der Länge n zu konstruierende Formel $F = f_n(x)$ hat die Gestalt:

$$F = R \wedge A \wedge \ddot{U}_1 \wedge \ddot{U}_2 \wedge E$$

R dient der Kontrolle, dass durch die Werte der Z , P - und B -Variablen eindeutig $p(n)+1$ Konfigurationen $K_t, t \in [0 \dots p(n)]$ dargestellt werden. R hat die Gestalt $R_Z \wedge R_P \wedge R_B$, wobei :

$$R_Z := \forall_{t=0}^{p(n)} (\exists_{j=1}^k Z_{t,j} \wedge \neg \exists_{(i \neq j) \leq k} (Z_{t,i} \wedge Z_{t,j})) =: \forall_{t=0}^{p(n)} \exists_{i=1}^k z_{t,i}$$

$$R_P := \forall_{t=0}^{p(n)} \exists_{i=-p(n)}^{p(n)} P_{t,i}$$

$$R_B := \forall_{t=0}^{p(n)} \forall_{i=-p(n)}^{p(n)} \exists_{j=0}^l B_{t,i,j}$$

Die Teilformel A stellt sicher, dass für $t=0$ die Startkonfiguration über x eingenommen wird:

$$A := Z_{0,1} \wedge P_{0,0} \wedge \forall_{j=-p(n)}^{-1} B_{0,j,1} \wedge \forall_{j=0}^{n-1} B_{0,j,\mu_j} \wedge \forall^p(n)_{j=n} B_{0,j,1}$$

Hierbei $\square = a_1$ und $x = a_{\mu_0} \dots a_{\mu_{n-1}}, \mu_j \in \{1, \dots, l\}$, Startzustand von M sei z_1 .

Die Teilformel E stellt sicher, dass die letzte Konfiguration akzeptierend ist, also ein Zustand aus $E = \{z_{\alpha_1}, \dots, z_{\alpha_r}\}, r \leq u, 1 \leq \alpha_j \leq k$, angenommen wird.

$$E := Z_{p(n),\alpha_1} \vee Z_{p(n),\alpha_2} \dots Z_{p(n),\alpha_r}$$

\ddot{U}_2 stellt sicher, dass der Inhalt einer Turingzelle, über der nicht der Schreib-/Lesekopf steht, unverändert bleibt:

$$\ddot{U}_2 := \forall_{t=0}^{p(n)} \forall_{i=-p(n)}^{p(n)} \forall_{j=1}^l ((\neg P_{t,i} \wedge B_{t,i,j}) \rightarrow B_{t+1,i,j})$$

Der Kern der Konstruktion liegt in der Beschreibung des korrekten Konfigurationsübergangs durch \ddot{U}_1 :

$$\forall_{t=0}^{p(n)} \forall_{i=-p(n)}^{p(n)} \forall_{j=1}^k \forall_{m=1}^l [(Z_{t,j} \wedge P_{t,i} \wedge B_{t,i,m}) \rightarrow \exists_{(z_j, a_m, z_{j'}, a_{m'}, y)} (Z_{t+1,j'} \wedge P_{t+1,i+y} \wedge B_{t+1,i,m'})]$$

Hierbei wird die Bewegungsrichtung L mit -1 , N mit 0 und R mit $+1$ identifiziert.

”Feststellungen:”

- Die Abbildung $F_m(x)$ ist total.
- Die Größe von $F_m(x)$ ist polynomiell in $|x|$ beschränkt.
- Die Abbildung $x \rightarrow F_n(x)$ ist in Polynomzeit berechenbar.
- Ist $x \in L$, so ist $F_n(x) \in SAT$.
- Ist $F_n(x) \in SAT$, so ist $x \in L$.

Bemerkung:

Die besten bislang bekannten Algorithmen für SAT haben exponentielle Laufzeit.

3.3 Weitere NP-vollständige Probleme

Wir werden im Weiteren die NP-Vollständigkeit folgender Probleme zeigen:

3 SAT:

Gegeben: Eine aussagenlogische Formel F in konjunktiver Normalform mit höchstens 3 Literalen per Klausel.

Frage: Ist F erfüllbar?

Clique:

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$

Frage: Besitzt G eine ”Clique” der Größe k ? (Dies ist eine Teilmenge V' der Knotenmenge mit $|V'|=k$ und für alle $u, v \in V'$ mit $u \neq v$ gilt: $\{u, v\} \in E$)

Rucksack:

Gegeben: Natürliche Zahlen $a_1, a_2, a_3, \dots, a_k \in \mathbb{N}$ und $b \in \mathbb{N}$

Frage: Gibt es eine Teilmenge $I \subseteq \{1, 2, \dots, k\}$ mit $\sum_{i \in I} a_i = b$?

Partition:

Gegeben: Natürliche Zahlen $a_1, a_2, \dots, a_k \in \mathbb{N}$.

Frage: Gibt es eine Teilmenge $J \subseteq \{1, 2, \dots, k\}$ mit $\sum_{i \in J} a_i = \sum_{i \notin J} a_i$?

Bin Packing:

Gegeben: Eine "Behältergrösse" $b \in \mathbb{N}$, die Anzahl der Behälter $k \in \mathbb{N}$, "Objekte" $a_1, a_2, \dots, a_n \leq b$.

Frage: Können die Objekte so auf die k Behälter verteilt werden, dass kein Behälter überläuft? (Das heisst: gefragt ist, ob eine Abbildung $f : \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, k\}$ existiert, so daß für alle $j = 1, \dots, k$ gilt: $\sum_{f(i)=j} a_i \leq b$).

Gerichteter Hamilton-Kreis:

Gegeben: Ein gerichteter Graph $G = (V, E)$.

Frage: Besitzt G einen Hamilton-Kreis? (Dies ist eine Permutation π der Knotenindizes $(v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(n)})$, so daß für $i = 1, \dots, n - 1$ gilt: $(v_{\pi(i)}, v_{\pi(i+1)}) \in E$ und ausserdem $(v_{\pi(n)}, v_{\pi(1)}) \in E$)

Ungerichteter Hamilton-Kreis:

Gegeben: Ein ungerichteter Graph $G = (V, E)$.

Frage: Besitzt G einen Hamilton-Kreis? (Dies ist eine Permutation π der Knotenindizes $(v_{\pi(1)}, v_{\pi(2)}, \dots, v_{\pi(n)})$, so daß für $i = 1, \dots, n - 1$ gilt: $(v_{\pi(i)}, v_{\pi(i+1)}) \in E$ und ausserdem $(v_{\pi(n)}, v_{\pi(1)}) \in E$)

Traveling Salesman

Gegeben: Eine $n \times n$ Matrix $(M_{(i,j)})$ von "Entfernungen" zwischen n "Städten" und eine Zahl k .

Frage: Gibt es eine Permutation π (eine "Rundreise"), so dass $\sum_{i=1}^{n-1} M_{\pi(i), \pi(i+1)} + M_{\pi(n), \pi(1)} \leq k$?

Färbbarkeit

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$

Frage: Gibt es eine Färbung der Knoten in V mit k verschiedenen Farben, so daß keine zwei benachbarten Knoten in G dieselbe Farbe haben?

Knotenüberdeckung

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$.

Frage: Besitzt G eine "überdeckende Knotenmenge" der Grösse höchstens k ? (Dies ist eine Teilmenge $V' \subseteq V$ mit $|V'| \leq k$, so daß für all Kanten $\{u, v\} \in E$ gilt: $u \in V'$ oder $v \in V'$)

Für all diese Probleme ist es einfach, einen NP-Algorithmus zu finden. In jedem Fall geht es darum, eine "Lösung" zu raten und danach deterministisch die Korrektheit zu prüfen.

Die NP-Vollständigkeit dieser Probleme wird durch folgende Reduktionsstruktur gezeigt werden:

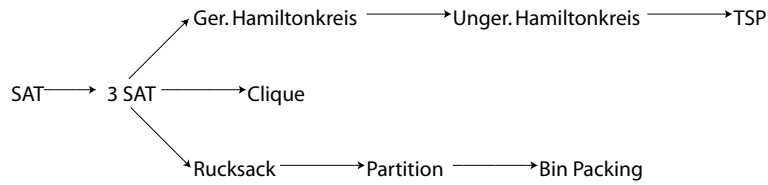


Abbildung 18: Übersicht

Reduktion 1 $SAT \leq_P 3SAT$

Zu einer Formel F ist eine Formel $g(F)$ in 3-konjunktiver Normalform derart zu konstruieren, dass $F \in SAT$ gilt, wenn $g(F) \in SAT$ gilt.

Das geschieht durch schrittweises Umformen der Formel F in eine 3 CNF-Formel, wobei in jedem Schritt die (Un)Erfüllbarkeit der Formel erhalten bleibt.

Schritt 1

Durch Anwendung der De Morgan'schen Regeln werden die Negationen zu den Blättern=Literals bewegt.

Schritt 2

Wir ordnen den inneren Knoten (also jedem \vee und \wedge) von F_1 jeweils neue Variablen $y_0, y_1 \dots$ zu. (Die Wurzel erhalten den Wert y_0).

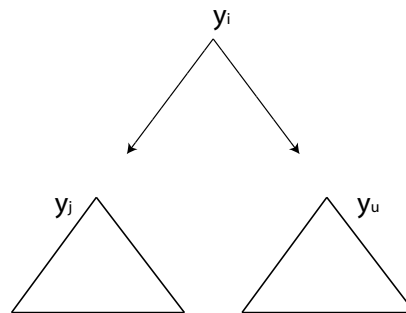


Abbildung 19:

Wir ersetzen durch $\{y_i \Leftrightarrow (y_j \wedge y_k)\}$ und entsprechend für \vee .

Schritt 3

Jede Unterformel von F_2 , die eine Äquivalenz enthält, wird auf folgende Weise umgeformt:

$$[a \Leftrightarrow (b \vee c)] \longrightarrow (a \vee \neg b) \wedge (a \vee \neg c) \wedge (\neg a \vee b \vee c)$$

$$[a \Leftrightarrow (b \wedge c)] \longrightarrow (\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)$$

Die sich ergebende Formel $g(F)$ ist in 3CNF.

Die Vergrößerung der Formel in den einzelnen Schritten ist linear und g lässt sich in Polynomzeit berechnen. wegen $g^{-1}(3SAT) = SAT$ ist also $SAT \leq_m^P 3SAT$.

Das Knapsack-Problem

Knapsack := $\{(a_1, a_2, \dots, a_k, b) \mid k \geq 1, a_i \in \mathbb{N}, b \in \mathbb{N}, \exists I \subseteq \{1, 2, \dots, n\} \text{ mit } \sum_{i \in I} a_i = b\}$

Offenbar ist Knapsack \in NP.

Satz 35

Knapsack ist NP-vollständig.

Beweis

Durch Nachweis von $3SAT \leq_P$ Knapsack.

Es sei $F = (z_{11} \vee z_{12} \vee z_{13}) \wedge \dots \wedge (z_{m1} \vee z_{m2} \vee z_{m3})$ mit $z_{ij} \in \{x_1, \dots, x_m\} \cup \{\neg x_1, \dots, \neg x_n\}$ ein 3 SAT-Kandidat.

Die Zahlen des zu konstruierenden Knapsack-Kandidaten haben m+n Ziffern über einem mindestens fünf-elementigen Ziffernvorrat. Die Konstruktion wird sicherstellen, dass Überträge keine Rolle spielen. \square

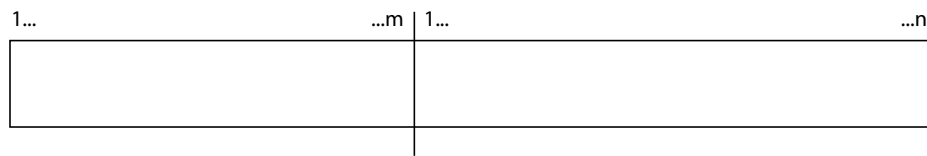


Abbildung 20:

Funktion: Jede Klausel erfüllt | Jede Variable falsch oder wahr

Die Zielzahl b hat die Gestalt: $b = \underbrace{44 \dots 4}_m \underbrace{11 \dots 1}_n$

Zu jeder Klausel und zu jeder Variable wird es zwei a_i -Zahlen geben; zu jeder Variablen $x_i, 1 \leq i \leq n$ gibt es eine Zahl $v_i = d_1^i \dots d_m^i 00 \dots 010 \dots 0$ (1 an i-ter Stelle) mit $d_j^i =$ Anzahl der positiven Vorkommen von x_i in Klausel j, also $d_j^i \in \{0, 1, 2, 3\}$ entsprechend sei $v'_i = d_1^i \dots d_m^i 00 \dots 010 \dots 0$ mit der $d_j^i =$ Anzahl der negativen Vorkommen von x_i in Klausel j.

Die weiteren Zahlen werden in den rechten n Stellen nur Nullen enthalten. Um die Zielzahl b zu erhalten muss also für jedes $1 \leq i \leq n$ entweder v_i oder v'_i in die Menge I gewählt werden .

Die Wahl eines v_i entspricht einer positiven, die von v'_i einer negativen Belegung der Variable x_i .

Derartig gewählte v_1, v_2, \dots, v_n bzw. v'_1, v'_2, \dots, v'_n summieren sich zu einer Zahl $d_1 \dots d_m 11 \dots 1$ derart auf, dass $d_j, 1 \leq j \leq m$ die Anzahl der durch diese Belegung in Klausel j erfüllten Literale ausgibt.

Jedes d_j hat einen Wert in $\{0, 1, 2, 3\}$. Die Belegung ist erfüllend (g.d.), wenn jedes $d_j \geq 1$ ist. Durch Hinzufügen der Zahlen:

und

lassen sich genau für erfüllende Belegung die Zielzahl b aufsummierend erreichen.

$$\begin{array}{cc}
 \overbrace{10000\dots 0}^m & \overbrace{000\dots 0}^n \\
 01000\dots 0 & 000\dots 0 \\
 \vdots & \vdots \\
 00000\dots 1 & 000\dots 0 \\
 \\
 20000\dots 0 & 000\dots 0 \\
 02000\dots 0 & 000\dots 0 \\
 \vdots & \vdots \\
 00000\dots 2 & 000\dots 0
 \end{array}$$

Das Partitions-Problem

Partition := $\{ \langle a_1, \dots, a_k \rangle \mid k \geq 2, a_j \in \mathbb{N}, \exists I \subseteq \{1, 2, \dots, k\}_{i \in I} \text{ mit } \sum a_i = \sum_{i \in I} a_i \}$

Offenbar ist Partition \in NP.

Satz 36

Partition ist NP-vollständig.

Beweis

Durch Nachweis von Partition \leq_P Knapsack.

$$\langle a_1, \dots, a_k, b \rangle \rightarrow \langle a_1, \dots, a_k, \sum_{i=1}^k a_i - b + 1, b + 1 \rangle \quad \square$$

Das Bin Packing-Problem

Bin Packing := $\{ \langle b, k, a_1, \dots, a_n \rangle \mid n, k, a_i, b \in \mathbb{N}, \exists f : \{1, \dots, n\} \rightarrow \{1, \dots, k\} \wedge \forall_{j=1}^k : \sum_{f(i)=j} a_i \leq b \}$

Offenbar ist Bin Packing \in NP.

Satz 37

Bin Packing ist NP-vollständig.

Beweis

Durch Nachweis von Partition \leq_P Bin Packing

$$\langle a_1, \dots, a_k \rangle \rightarrow \langle \sum_{i=1}^k \frac{a_i}{2}, 2, a_1, \dots, a_k \rangle \quad \square$$

Index

- Äquivalenz von Zuständen, 16
- Abschlusseigenschaften, 18, 24, 36
- Ackermannfunktion, 45
- Automaten, 2
- Bin Packing, 66
- Bin-Packing, 62
- Chomsky-Hierarchie, 4
- Chomsky-Normalform, 20
- Churchsche These, 37
- Clique, 62
- Cocke, 25
- CYK-Algorithmus, 25
- deterministisch kontextfreie Sprachen, 30
- Endliche Automaten, 6
- endliche Automaten, 15
- Entscheidbarkeit, 19, 36, 37
- Entscheidbarkeit bei kontextfreien Sprachen, 32
- Farbbarkeit, 63
- Formale Sprache, 3
- Formale Sprachen, 2
- gerichteter Hamiltonkreis, 62
- GOTO, 39
- GOTO-Berechenbarkeit, 42
- Grammatik, 3
- Greibach-Normalform, 21
- Halteproblem, 48, 52
- Kasami, 25
- Kellerautomat, 26
- Kleene, 11, 45
- Knapsack, 64
- Knotenüberdeckung, 63
- Kodierung, 50
- Komplexität, 37
- Komplexitätsklassen, 57
- Komplexitätstheorie, 57
- kontextfreie Sprachen, 19
- kontextsensitiv, 33
- Korrespondenzproblem, 54
- LOOP, 39
- LOOP-Berechenbarkeit, 39
- Mehrbandturingmaschinen, 38
- Minimalautomat, 17
- Myhill-Nerode, 15
- nichtdeterministisch, 7
- Normalform, 20
- NP-Vollständigkeit, 58
- NPDA, 27
- Parikh, 24
- partielle Rekursion, 44
- Partition, 62, 66
- PCP, 54, 56
- PDA, 27
- Post'sches Korrespondenzproblem, 54
- primitive Rekursion, 44
- Pumping Lemma, 14
- Rabin, 9
- RAM, 39
- Reduzierbarkeit, 48
- Registermaschine, 39
- regulär, 10
- reguläre Ausdrücke, 10
- Reguläre Sprachen, 6
- reguläre Sprachen, 18
- reguläre Sprache, 14
- Rice, 53
- Rucksack, 62
- SAT, 60, 62
- Scott, 9
- Selbstanwendung, 50
- Traveling Salesman, 63
- Turingberechenbarkeit, 38
- Typ-0-Sprachen, 33
- Unentscheidbarkeit, 48
- ungerichteter Hamiltonkreis, 63
- uvwxy-Theorem, 21
- WHILE, 39
- WHILE-Berechenbarkeit, 41
- Younger, 25