Grundlagen der Theoretischen Informatik

Wintersemester 2024/25

Prof. Dr. Heribert Vollmer

Institut für Theoretische Informatik
Leibniz Universität Hannover

Link für Kurzfragen:

URL: https://pingo.coactum.de/events/739935

Bitte scannen!

URL: https://pingo.coactum.de/events/739935

Organisatorisches

- Vorlesung Mo 10.15h hier in E001, Aufzeichnung in Stud.IP (aus WS20/21) werden eine Woche vor Vorlesung bereitgestellt
- Materialien in Stud.IP: Skript, Folien
- Übungen in Kleingruppen mit max. 25 TN, Beginn: heute!
- Hausübungen werden dreimal im Semester verteilt;
  vermutlich Mitte Nov., Mitte Dez., Ende Jan.;
  Bearbeitungszeit 2 Wochen;
  elektr. Abgabe (PDF) in Gruppen (1-4 TN)
- Studienleistung: 60% der Punkte der Hausübungen
  bitte E-Mail an Vivian Holzapfel
- Prüfungsleistung: Klausur, 120 min,
  geplanter Termin: 24.02.24, 14h;
  für Lehramtsstudiengänge: mündl. Prüfung, Termin n. V.

Übungskonzept

- Tutorien in Gruppen (max. 25 TN)
- Vorbereitung: Vorlesung (ev.Aufzeichnung), Skript
- Keine Wiederholung des Vorlesungsinhaltes in der Übung!
- Aufgaben werden in der Übung gerechnet
- Kleingruppen erarbeiten Lösungen
- Lösungen werden in Gesamtgruppe besprochen

Inhalt

Sprachen und Grammatiken

Alphabete, Zeichen und Symbole

Ein Alphabet ist eine endliche, nichtleere Menge. Die Elemente eines Alphabets heißen auch Zeichen oder Symbole.

Wie üblich: Ist M eine Menge, so bezeichnet $|M|$ die Anzahl der Elemente von M.

Wörter und Sprachen

Sei Σ ein Alphabet.
Ein Wort über Σ ist eine Folge von Symbolen aus Σ.
Ein Wort entsteht also durch Hintereinanderschreiben (Konkatenation) von Symbolen aus Σ.
Mit ε wird das leere Wort bezeichnet.

Wörter und Sprachen

Die Menge aller Wörter über dem Alphabet Σ bezeichnen wir mit Σ^*. Eine Sprache über Σ ist eine Menge von Wörtern über Σ, also eine Teilmenge von Σ^*.

Konkatenation

- Operation auf Wörtern: Konkatenation bzw. Hintereinanderschreiben
- Schreibweise: $u \circ v$ oder kurz uv für Konkatenation der Wörter u und v
- Für ein Wort w und $n \in \mathbb{N}$ ist w^n die Konkatenation $w^n = \underbrace{w \circ w \circ \cdots \circ w}_{n-\mathrm{mal}}$
- Wir definieren: $w^0 = \varepsilon$.

Länge

- Die Länge eines Wortes w ist die Anzahl der Symbole in w. Schreibweise: $|w|$
- $|\varepsilon| = 0$.
- Es ist $|w^n| = n|w|$.

Schreibweise: $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$

Syntax der Aussagenlogik: Beispiel für EBNF

$\phi ::= p \mid 0 \mid 1 \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\phi \leftrightarrow \phi)$,

wobei p eine aussagenlogische Variable ist, also $p \in \{p_1, p_2, p_3, \ldots\}$.

Definition

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

- V ist eine endliche Menge, die so genannte Menge der Variablen
- Σ ist ein Alphabet, das so genannte Terminalalphabet, mit $V \cap \Sigma = \emptyset$
- P ist die endliche Menge der Produktionen, $P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$
- $S \in V$ ist die so genannte Startvariable

Definition

Sei $G = (V, \Sigma, P, S)$ eine Grammatik und seien $u, v \in (V \cup \Sigma)^*$.
Wir definieren eine Relation $\Rightarrow_G$ wie folgt:

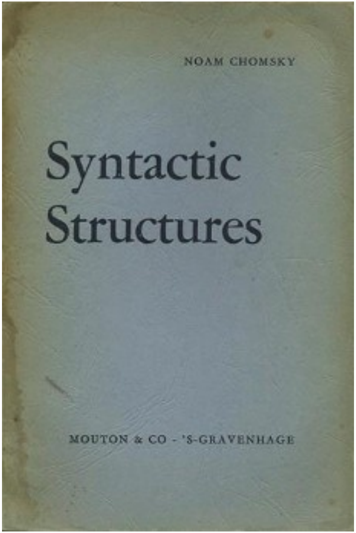
- $u \Rightarrow_G v$, falls u, v zerlegt werden können in Teilwörter $u = xyz$ und $v = xy'z$ mit $x, z \in (V \cup \Sigma)^*$ und $y \to y'$ ist Regel in P.
  „u geht unter (Anwendung einer Regel in) G unmittelbar über in v"
- $u \Rightarrow_G^* v$, falls $u = v$ oder es Wörter $w_1, \ldots, w_k \in (V \cup \Sigma)^*$ gibt mit $u = w_1$, $w_i \Rightarrow_G w_{i+1}$ für $i = 1, 2. \ldots, k-1$ und $v = w_k$.
  Wir lassen den Index G weg, falls dieser eindeutig ist.

Die von G erzeugte Sprache ist $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$.
Eine Ableitung von $w \in L(G)$ in k Schritten ist eine Folge $(w_0, w_1, \ldots, w_k)$ mit $w_0 = S$, $w_k = w$ und $w_i \Rightarrow_G w_{i+1}$ für $i = 0, 1, \ldots, k-1$.

Die Chomsky-Hierarchie

Noam Chomsky

∗ 7. Dez. 1928, Philadelphia



1957: Syntactic Structures

Definition

- Jede Grammatik ist vom Typ 0 (d. h. keine Einschränkungen).
- Eine Grammatik ist vom Typ 1 (oder: kontextsensitiv), falls für alle ihre Regeln $u \to v$ gilt: $|u| \leq |v|$.
- Eine Typ-1-Grammatik ist vom Typ 2 (oder: kontextfrei), falls für alle ihre Regeln $u \to v$ gilt, dass u eine einzelne Variable ist (d. h. $u \in V$).
- Eine Typ-2-Grammatik ist vom Typ 3 (oder: regulär), falls für alle ihre Regeln $u \to v$ gilt, dass v ein einzelnes Terminalzeichen ist ($v \in \Sigma$) oder v aus einem Terminalzeichen gefolgt von einer Variablen besteht.

Zurück zur Syntax der Aussagenlogik

EBNF:

$\phi ::= p \mid 0 \mid 1 \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\phi \leftrightarrow \phi)$,

wobei p eine aussagenlogische Variable ist,
also $p \in \{p_1, p_2, p_3, \ldots\}$.

Typ-2-Grammatik:

$S \rightarrow V \mid C \mid \neg S \mid (S \wedge S) \mid (S \vee S) \mid (S \rightarrow S) \mid (S \leftrightarrow S)$

$V \rightarrow p_1 \mid p_2 \mid p_3 \mid \ldots$

$C \rightarrow 0 \mid 1$

Problem: unendliches Alphabet!

Zurück zur Syntax der Aussagenlogik

Lösung:

Für p_i schreiben wir: pI^i.

$G = (\Sigma_{AL}, \{S, V, C\}, P, S)$, wobei

$$\Sigma_{AL} = \{p, I, 0, 1, \wedge, \vee, \neg, \rightarrow, \leftrightarrow, (,)\}$$

$$P = \left\{ \begin{array}{l} S \rightarrow V \mid C \mid \neg S \mid (S \wedge S) \mid (S \vee S) \mid (S \rightarrow S) \mid (S \leftrightarrow S) \\ V \rightarrow p \mid VI \\ C \rightarrow 0 \mid 1 \end{array} \right\}$$

Die syntaktisch korrekten Wörter (also die aussagenlogischen Formeln) kann man nun z. B. wie folgt erzeugen:

$$\begin{aligned} S &\Rightarrow \neg S \Rightarrow \neg(S \wedge S) \Rightarrow \neg(VI \wedge VI) \Rightarrow \neg(VI \wedge VII) \\ &\Rightarrow \neg(pI \wedge pII) \simeq \neg(p_1 \wedge p_2) \end{aligned}$$

Spezialfall des leeren Wortes

Bei einer Grammatik $G = (V, \Sigma, P, S)$ vom Typ 1, 2 oder 3 ist unabhängig von den oben genannten Restriktionen die Regel $S \rightarrow \varepsilon$ zugelassen.
Ist aber $S \rightarrow \varepsilon \in P$, so darf es keine Regel in P geben, in der S auf der rechten Seite vorkommt.

Eine Sprache $L \subseteq \Sigma^*$ heißt vom Typ 0 (Typ 1, Typ 2, Typ 3), falls es eine Typ-0-Grammatik (Typ-1-Grammatik, Typ-2-Grammatik, Typ-3-Grammatik) G gibt mit $L = L(G)$.

Satz

Das Wortproblem für Typ-1-Sprachen ist „entscheidbar“ , d. h. es gibt einen Algorithmus, der bei Eingabe einer kontextsensitiven Grammatik $G = (V, \Sigma, P, S)$ und eines Wortes $w \in \Sigma^*$ nach endlicher Zeit mit der Ausgabe „$w \in L(G)$“ oder „$w \notin L(G)$“ anhält.

Reguläre Sprachen

Definition

Ein (deterministischer) endlicher Automat (kurz: DEA) ist ein 5-Tupel

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

wobei für die einzelnen Komponenten gilt:

- Z ist eine endliche Menge, die so genannte Zustandsmenge
- Σ ist ein Alphabet, das so genannte Eingabealphabet, $Z \cap \Sigma = \emptyset$
- δ: $Z \times \Sigma \to Z$ ist die so genannte Überführungsfunktion
- $z_0 \in Z$ ist der so genannte Startzustand
- $E \subseteq Z$ ist die Menge der so genannten Endzustände

Definition

Sei $M = (Z, \Sigma, \delta, z_0, E)$ ein DEA. Die erweiterte Überführungsfunktion $\hat{\delta}$: $Z \times \Sigma^* \to Z$ ist (induktiv) definiert wie folgt:

$\hat{\delta}(z, \varepsilon) = z$ für alle $z \in Z$

$\hat{\delta}(z, ax) = \hat{\delta}(\delta(z, a), x)$ für alle $z \in Z$, $a \in \Sigma$ und $x \in \Sigma^*$

Die von M akzeptierte Sprache ist

$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(z_0, x) \in E\}.$$

Definition

Ein nichtdeterministischer endlicher Automat (kurz: NEA) ist ein 5-Tupel

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

wobei für die einzelnen Komponenten gilt:

- Z, Σ, z_0 und E sind wie bei deterministischen endlichen Automaten definiert
- Für die Überführungsfunktion gilt: δ: $Z \times \Sigma \to \mathcal{P}(Z)$. $\mathcal{P}(Z)$ ist die Potenzmenge von Z. Für $z \in Z$ und $a \in \Sigma$ ist also $\delta(z, a)$ eine Menge von möglichen Folgezuständen

Definition

Wir definieren $\hat{\delta}\colon \mathcal{P}(Z) \times \Sigma^* \to \mathcal{P}(Z)$ wie folgt:
$\hat{\delta}(Z', \varepsilon) = Z'$ für alle $Z' \subseteq Z$
$\hat{\delta}(Z', ax) = \bigcup_{z \in Z'} \hat{\delta}(\delta(z, a), x)$ für alle $Z' \subseteq Z$, $a \in \Sigma$ und $x \in \Sigma^*$.

Die von M akzeptierte Sprache ist

$$L(M) = \{x \in \Sigma^* \mid \hat{\delta}(\{z_0\}, x) \cap E \neq \emptyset\}.$$

Satz

Zu jedem NEA M existiert ein DEA M' mit $\mathrm{L}(M) = \mathrm{L}(M')$.

Satz

Sei $L \subseteq \Sigma^*$ eine Sprache. Es gibt einen DEA M mit $L = L(M)$ gdw. es eine reguläre Grammatik G mit $L = L(G)$ gibt.

Satz (Pumping-Lemma, uvw-Theorem)

Sei L eine reguläre Sprache. Dann gibt es eine Zahl n, sodass sich alle Wörter $x \in L$ mit $|x| \geq n$ zerlegen lassen in $x = uvw$, sodass folgende Eigenschaften gelten:

1. $|v| \geq 1$
2. $|uv| \leq n$
3. Für alle $i \geq 0$ gilt: $uv^i w \in L$.

Logische Struktur der Aussage des Pumping-Lemmas:

$$(L \text{ regulär}) \Rightarrow \quad \underbrace{(\exists n)(\forall x \in L, |x| \geq n)(\exists u, v, w), [x = uvw \text{ und } (1)\text{–}(3) \text{ gelten}]}_{\text{Aussage } (\star)}$$

Nach dem Pumping-Lemma gilt: „L regulär $\Rightarrow$ $(\star)$“.

Die Umkehrung (d. h. „$(\star) \Rightarrow$ L regulär“) gilt im Allgemeinen nicht!

Aber: $(\star)$ gilt nicht $\Rightarrow$ L nicht regulär. In dieser Form wird das Pumping-Lemma meistens verwendet.

Kontextfreie Sprachen

Definition

Ein (nichtdeterministischer) Kellerautomat (NKA, Pushdown Automaton (PDA)) ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \#, E),$$

wobei für die einzelnen Komponenten gilt:

- Z ist die endliche Menge der Zustände
- Σ ist das Eingabealphabet
- Γ ist das Kelleralphabet
- δ: $Z \times \Sigma \times \Gamma \to \mathcal{P}(Z \times \Gamma^*)$ ist die Überführungsfunktion. Es gilt: $\delta(z, a, A)$ ist endlich für alle $z \in Z$, $a \in \Sigma$ und $A \in \Gamma$
- $z_0 \in Z$ ist der Startzustand
- $\# \in \Gamma$ ist das unterste Kellersymbol
- $E \subseteq Z$ ist die Menge der Endzustände

Erläuterung der Arbeitsweise

Startkonfiguration:

M befindet sich am Anfang im Zustand z_0. Der Eingabekopf steht auf dem ersten Zeichen der Eingabe. Der Keller enthält lediglich das Symbol $\#$.

Zustandsübergang:

$\delta(z, a, A) \ni (z', B_1, \ldots, B_k)$ bedeutet:
Ist M im Zustand z, liest das Eingabezeichen a und ist A das oberste Kellersymbol, so kann M in den Zustand z' übergehen und das Kellersymbol A durch die Symbole $B_1, \ldots, B_k$ (B_1 wird oberstes Kellersymbol) ersetzen. Der Eingabekopf wandert eine Position nach rechts.
$(z, z' \in Z,\ a \in \Sigma,\ A, B_1, \ldots, B_k \in \Gamma.)$

Erläuterung der Arbeitsweise

Ende der Rechnung:

- Eingabe ganz gelesen
- oder keine Einträge in δ passen zur aktuellen Situation, d. h. M stürzt ab, beispielsweise dadurch, dass der Keller geleert wurde.

Akzeptierte Sprache:

Ein Eingabewort wird akzeptiert, falls ein Zustand aus E angenommen wird, nachdem die Eingabe ganz gelesen wurde. Genauer: Falls es eine Folge von nichtdeterministischen Wahlmöglichkeiten gibt, sodass M einen Endzustand annimmt, nachdem die Eingabe ganz gelesen wurde.

$$L(M) = \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$$

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

$L = L(M)$ für den NKA

$$M = (\{z_0, z_1, z_2\}, \{a, b\}, \{\#, A, \underline{A}\}, \delta, z_0, \{z_2\}) ,$$

wobei δ wie folgt definiert ist:

$$z_0 a \# \to z_0 \underline{A} \qquad (1)$$
$$z_0 a \underline{A} \to z_0 A \underline{A} \qquad (2)$$
$$z_0 a A \to z_0 A A \qquad (3)$$

$$z_0 b A \to z_1 \varepsilon \qquad (4)$$
$$z_0 b \underline{A} \to z_2 \varepsilon \qquad (5)$$

$$z_1 b A \to z_1 \varepsilon \qquad (6)$$
$$z_1 b \underline{A} \to z_2 \varepsilon \qquad (7)$$

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

$w = \text{aaabbb}$

Zustand	Rest der Eingabe	Kellerinhalt	Befehl
z_0	aaabbb	#	(1)
z_0	aabbb	$\underline{A}$	(2)
z_0	abbb	$A\underline{A}$	(3)
z_0	bbb	$AA\underline{A}$	(4)
z_1	bb	$A\underline{A}$	(6)
z_1	b	$\underline{A}$	(7)
z_2	ε	ε	

Damit gilt also aaabbb $\in L(M)$.

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

$w = \text{aaabb}$

Zustand	Rest der Eingabe	Kellerinhalt	Befehl
z_0	aaabb	#	(1),(2),(3)
z_0	bb	$AA\underline{A}$	(4)
z_1	b	$A\underline{A}$	(6)
z_1	ε	$\underline{A}$	

An dieser Stelle ist die Eingabe ganz gelesen und kein Endzustand erreicht worden, also gilt: aaabb $\notin L(M)$.

Beispiel 1c: $L = \{a^n b^n \mid n \geq 1\}$

$w = \text{abb}$

Zustand	Rest der Eingabe	Kellerinhalt	Befehl
z_0	abb	#	(1)
z_0	bb	$\underline{A}$	(5)
z_2	b	ε	

An dieser Stelle ist kein weiterer Befehl möglich und die Eingabe ist noch nicht vollständig gelesen worden, also gilt: $\text{abb} \notin L(M)$.

Beispiel 2: $L = \{w\$w^R \mid w \in \{a, b\}^+\}$

$L = L(M)$ für den NKA

$$M = (\{z_0, z_1, z_2\}, \{a, b, \$\}, \{\#, A, B, \underline{A}, \underline{B}\}, \delta, z_0, \{z_2\}),$$

wobei δ wie folgt definiert ist:

$z_0 a\# \to z_0 \underline{A}$	$z_0 a\underline{A} \to z_0 A\underline{A}$	$z_0 aA \to z_0 AA$	$z_0 a\underline{B} \to z_0 A\underline{B}$	$z_0 aB \to z_0 AB$
$z_0 b\# \to z_0 \underline{B}$	$z_0 b\underline{A} \to z_0 B\underline{A}$	$z_0 bA \to z_0 BA$	$z_0 b\underline{B} \to z_0 B\underline{B}$	$z_0 bB \to z_0 BB$
	$z_0 \$\underline{A} \to z_1 \underline{A}$	$z_0 \$A \to z_1 A$	$z_0 \$\underline{B} \to z_1 B$	$z_0 \$B \to z_1 B$
	$z_1 a\underline{A} \to z_2 \varepsilon$	$z_1 aA \to z_1 \varepsilon$		
			$z_1 b\underline{B} \to z_2 \varepsilon$	$z_1 bB \to z_1 \varepsilon$

Beispiel 2a: $L = \{w\$w^R \mid w \in \{a, b\}^+\}$

$w = ab\$ba$

Zustand	Rest der Eingabe	Kellerinhalt
z_0	ab\$ba	#
z_0	b\$ba	$\underline{A}$
z_0	\$ba	$B\underline{A}$
z_1	ba	$B\underline{A}$
z_1	a	$\underline{A}$
z_2	ε	ε

Also ist $ab\$ba \in L(M)$.

Beispiel 2b: $L = \{w\$w^R \mid w \in \{a, b\}^+\}$

$w = ab\$bb$

Zustand	Rest der Eingabe	Kellerinhalt
z_0	ab\$bb	#
z_0	b\$bb	$\underline{A}$
z_0	\$bb	$B\underline{A}$
z_1	bb	$B\underline{A}$
z_1	b	$\underline{A}$

keine weitere Bewegung möglich

Also ist ab\$bb $\notin L(M)$.

Satz

Eine Sprache L ist kontextfrei gdw. es einen NKA M gibt mit $\mathrm{L} = \mathrm{L}(M)$.

Satz (Pumping-Lemma ($uvwxy$-Theorem))

Sei L eine kontextfreie Sprache. Dann gibt es eine Zahl n, sodass sich alle Wörter $z \in L$ mit $|z| \geq n$ zerlegen lassen in $z = uvwxy$, sodass folgende Eigenschaften erfüllt sind:

1. $|vx| \geq 1$
2. $|vwx| \leq n$
3. Für alle $i \geq 0$ gilt: $uv^iwx^iy \in L$

Logische Struktur der Aussage des Pumping-Lemmas:

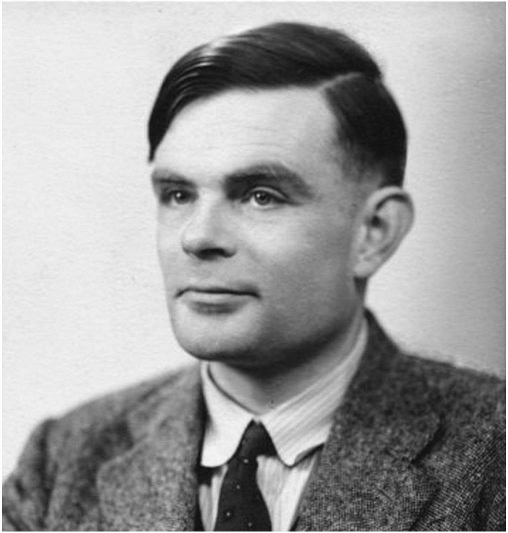
$$(L \text{ kontextfrei}) \Rightarrow \quad \underbrace{\begin{array}{c}(\exists n \in \mathbb{N})(\forall z \in L, |z| \geq n)(\exists u, v, w, x, y), \\ [z = uvwxy \wedge (1)\text{–}(3) \text{ gelten}]\end{array}}_{(\star)}$$

Anwendung: Kontraposition des Satzes, also:

$$(\star) \text{ gilt nicht} \Rightarrow L \text{ ist nicht kontextfrei.}$$

Typ-1- und Typ-0-Sprachen

Alan Turing



Geboren: 23. Juni 1912, Maida Vale

Gestorben: 7. Juni 1954, Wilmslow, Vereinigtes Königreich

Definition

Eine Turingmaschine (TM) ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E),$$

wobei für die einzelnen Komponenten gilt:

- Z ist die Menge der Zustände
- Σ ist das Eingabealphabet
- $\Gamma \supset \Sigma$ ist das Arbeitsalphabet
- $z_0 \in Z$ ist der Startzustand
- $\Box \in \Gamma \setminus \Sigma$ ist das Leerzeichen bzw. Blank
- $E \subseteq Z$ ist die Menge der Endzustände
- δ ist die Übergangsfunktion

Definition (Fortsetzung)

Bei deterministischen Turingmaschinen (DTM, TM) gilt:

$$\delta\colon Z \times \Gamma \to Z \times \Gamma \times \{L, N, R\}$$

Bei nichtdeterministischen Turingmaschinen (NTM) gilt:

$$\delta\colon Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, N, R\})$$

Erläuterung der Arbeitsweise

Startkonfiguration:

M befindet sich am Anfang im Zustand z_0. Der Eingabekopf steht auf dem ersten Zeichen der Eingabe. Alle Bandzellen außerhalb der Eingabe enthalten das Leersymbol.

Zustandsübergang: (deterministischer Fall)

$\delta(z, a) = (z', b, X)$ bedeutet:
Ist M im Zustand z und liest das Eingabezeichen a, so geht M in den Zustand z' über, ersetzt das Eingabezeichen durch b und bewegt den Kopf gemäß X: R $\triangleq$ rechts, L $\triangleq$ links, N $\triangleq$ neutral (keine Kopfbewegung).
$(z, z' \in Z,\ a, b \in \Gamma.)$

Nichtdeterministische Maschine: mehrere mögliche analoge Übergänge.

Erläuterung der Arbeitsweise

Ende der Rechnung:

M hält, sobald ein Zustand aus E angenommen wird.

Akzeptierte Sprache:

Ein Eingabewort x wird akzeptiert, falls in der Rechnung von M auf x irgendwann ein Zustand aus E angenommen wird.

$$L(M) = \{w \in \Sigma^* \mid M \text{ akzeptiert } w\}$$

Definition

Eine Konfiguration einer TM $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$ ist ein Wort $k = uzv$, wobei $u, v \in \Gamma^*$ und $z \in Z$.

Startkonfiguration von M bei Eingabe w: $z_0 w$.

Definition

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0, \Box, E)$ eine TM. Wir definieren eine zweistellige Relation $\vdash$ auf der Menge der Konfigurationen wie folgt für $z \in Z \setminus E$:

$$a_1 \dots a_m z b_1 \dots b_n \vdash$$

$$\begin{cases} a_1 \dots a_m z' c b_2 \dots b_n, & \text{falls } \delta(z, b_1) = (z', c, N),\ m \geq 0,\ n \geq 1 \\ a_1 \dots a_m c z' b_2 \dots b_n, & \text{falls } \delta(z, b_1) = (z', c, R),\ m \geq 0,\ n \geq 2 \\ a_1 \dots z' a_m c b_2 \dots b_n, & \text{falls } \delta(z, b_1) = (z', c, L),\ m \geq 1,\ n \geq 1 \end{cases}$$

Sonderfälle

$n = 1$, Maschine läuft nach rechts:

$$a_1 \dots a_m z b_1 \vdash a_1 \dots a_m c z' \Box, \quad \text{falls } \delta(z, b_1) = (z', c, R),\ m \geq 0$$

$m = 0$, Maschine läuft nach links:

$$z b_1 \dots b_n \vdash z' \Box c b_2 \dots b_n, \quad \text{falls } \delta(z, b_1) = (z', c, L),\ n \geq 1$$

Für $z \in E$ gibt es keine Konfiguration k mit

$$a_1 \dots a_m z b_1 \dots b_n \vdash k.$$

Definition

Die von einer Turingmaschine $M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ akzeptierte Sprache ist

$$L(M) = \{w \in \Sigma^* \mid z_0 w \vdash^* uzv \text{ für ein } z \in E \text{ und } u, v \in \Gamma^*\}.$$

Dabei ist $k_a \vdash^* k_e$, falls $k_a = k_e$ oder es $k_1, \ldots, k_n$ gibt mit

$$k_a \vdash k_1 \vdash \cdots \vdash k_n \vdash k_e.$$

Also: Ein Wort wird akzeptiert, falls irgendwann ein Endzustand angenommen wird.

Definition

Ein linear-beschränkter Automat (LBA) ist eine NTM

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$$

mit folgenden Eigenschaften:

- $\Gamma \setminus \Sigma$ enthält zwei spezielle Symbole $\triangleright$ und $\triangleleft$, die so genannte linke bzw. rechte Bandendemarkierung
- Falls M $\triangleright$ liest, ist keine Kopfbewegung nach links erlaubt
- Falls M $\triangleleft$ liest, ist keine Kopfbewegung nach rechts erlaubt
- Die Bandsymbole $\triangleright$ und $\triangleleft$ dürfen nicht durch andere Zeichen überschrieben werden

Die von M akzeptierte Sprache ist

$$L(M) = \{w \in \Sigma^* \mid z_0 \triangleright w \triangleleft \vdash^* uzv \text{ für ein } z \in E \text{ und } u, v \in \Gamma^*\}.$$

Satz

1. Eine Sprache L ist kontextsensitiv (Typ 1) gdw. es einen LBA gibt mit $L(M) = L$
2. Eine Sprache L ist vom Typ 0 gdw. es eine TM M gibt mit $L(M) = L$ gdw. es eine NTM M gibt mit $L(M) = L$

Bemerkung

Es ist unbekannt, ob deterministische LBAen nicht schon die Klasse der Typ-1-Sprachen akzeptieren.

LBA-Problem: Gibt es für jede Typ-1-Sprache einen deterministischen LBA, der sie akzeptiert?

Der intuitive Berechenbarkeitsbegriff

Berechenbarkeit

Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ heißt berechenbar, falls es einen Algorithmus gibt, der f berechnet, d. h. gestartet mit Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}^k$ hält der Algorithmus nach endlich vielen Schritten mit Ausgabe $f(n_1, \ldots, n_k)$.

Wir fordern nicht, dass f total sein muss, d. h. für gewisse $(n_1, \ldots, n_k) \in \mathbb{N}^k$ darf $f(n_1, \ldots, n_k)$ undefiniert sein. In diesem Fall soll der Algorithmus nicht stoppen (Endlosschleife).

Ziel: Präzisierung des Berechenbarkeitsbegriffs, d.h. des Begriffs Algorithmus.

Nur so ist es möglich, zu beweisen, dass eine Funktion nicht berechenbar ist.

Beispiel 1

$$f_1(n) = \begin{cases} 1, & \text{falls } n \text{ ein Anfangsabschnitt der} \\ & \text{Nachkommastellen von } \pi \text{ ist} \\ 0, & \text{sonst} \end{cases}$$

Beispiel 2

$$f_2(n) = \begin{cases} 1, & \text{falls } n \text{ irgendwo in den} \\ & \text{Nachkommastellen von } \pi \text{ vorkommt} \\ 0, & \text{sonst} \end{cases}$$

Beispiel 3

$$f_3(n) = \begin{cases} 1, & \text{falls 7 in den Nachkommastellen von } \pi \text{ irgendwo} \\ & \text{mindestens } n\text{-mal hintereinander vorkommt} \\ 0, & \text{sonst} \end{cases}$$

Beispiel 4

$$f_4(n) = \begin{cases} 1, & \text{falls die Antwort auf das LBA-Problem „ja“ ist} \\ 0, & \text{sonst} \end{cases}$$

Turing-Berechenbarkeit

Definition

Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ heißt Turing-berechenbar, falls es eine DTM M gibt, sodass für alle $n_1, \ldots, n_k, m \in \mathbb{N}$ gilt:
$f(n_1, \ldots, n_k) = m \Rightarrow$

M mit Eingabe $\mathrm{bin}(n_1)\#\ldots\#\mathrm{bin}(n_k)$
hält mit $\Box\cdots\Box\mathrm{bin}(m)\Box\cdots\Box$
auf dem Arbeitsband.

$f(n_1, \ldots, n_k)$ undefiniert $\Rightarrow$

M mit Eingabe $\mathrm{bin}(n_1)\#\mathrm{bin}(n_2)\#\ldots\#\mathrm{bin}(n_k)$
stoppt nicht.

$\mathrm{bin}(n)$ für $n \in \mathbb{N}$ bezeichnet die Binärdarstellung von n ohne führende Nullen.

Bemerkung

Das Eingabealphabet einer TM, die eine Funktion über $\mathbb{N}$ im obigen Sinne berechnet, ist stets $\{0, 1, \#\}$.

Definition

Eine Funktion $f\colon \Sigma^* \to \Delta^*$ heißt Turing-berechenbar, falls es DTM M gibt, sodass für alle $x \in \Sigma^*$ und $y \in \Delta^*$ gilt:

$f(x) = y \Rightarrow$

> M mit Eingabe x
> hält mit $\square\cdots\square y\square\cdots\square$ auf dem Arbeitsband.

$f(x)$ undefiniert $\Rightarrow$

> M mit Eingabe x stoppt nicht.

Mehrband-Maschinen

Definition

Eine k-Band-DTM ist ein 7-Tupel

$$M = (Z, \Sigma, \Gamma, \delta, z_0, \square, E),$$

wobei für die einzelnen Komponenten gilt:

- Z, Σ, Γ, z_0, $\square$ und E sind wie bei einer 1-Band-DTM definiert.
- $\delta\colon \underbrace{Z}_{\text{(i)}} \times \underbrace{\Gamma^k}_{\text{(ii)}} \to \underbrace{Z}_{\text{(iii)}} \times \underbrace{\Gamma^k}_{\text{(iv)}} \times \underbrace{\{L, R, N\}^k}_{\text{(v)}}$ mit
  - (i) aktueller Zustand
  - (ii) gelesene Zeichen auf den k Bändern
  - (iii) neuer Zustand
  - (iv) geschriebene Zeichen auf den k Bändern
  - (v) Kopfbewegungen auf den k Bändern

Arbeitsweise

Die Eingabe steht zunächst auf Band 1. Die Bänder 2 bis k sind zunächst leer.
Die Maschine führt einzelne Schritte durch, analog zu gewöhnlichen DTMn.

Akzeptierte Sprache: Das Eingabewort x wird akzeptiert gdw. M erreicht irgendwann einen Endzustand.

Berechnete Funktion: $f(n_1, \dots, n_k) = m$ gdw. M mit Eingabe $\mathrm{bin}(n_1)\#\dots\#\mathrm{bin}(n_k)$ erreicht irgendwann einen Endzustand mit $\mathrm{bin}(m)$ auf Band 1.
(Berechnung von Funktionen $f\colon \Sigma^* \to \Delta^*$ analog.)

Beispiel

Folgende 2-Band-Turingmaschine akzeptiert $\{w\#w \mid w \in \{0,1\}^*\}$:

$$M = (\{z_0, z_1, z_2, z_e\}, \{0,1,\#\}, \{0,1,\#,\square\}, \delta, z_0, \square, \{z_e\}),$$

wobei für die Überführungsfunktion gilt:

$$\left.\begin{array}{lcl} z_0 0\square & \to & z_0 00RR \\ z_0 1\square & \to & z_0 11RR \end{array}\right\} \text{ 0, 1 auf Band 1 werden auf Band 2 kopiert}$$

$$\left.\begin{array}{lcl} z_0 \#\square & \to & z_1 \#\square NL \end{array}\right\} \text{ \# auf Band 1} \Rightarrow \text{Zustand } z_1$$

$$\left.\begin{array}{lcl} z_0 \square\square & \to & z_0 \square\square NN \end{array}\right\} \text{ Endlosschleife, falls kein \# gefunden wird}$$

$$\left.\begin{array}{lcl} z_1 \#0 & \to & z_1 \#0NL \\ z_1 \#1 & \to & z_1 \#1NL \end{array}\right\} \begin{array}{l}\text{Kopf auf Band 2 nach links} \\ \text{Kopf auf Band 1 bleibt auf \#}\end{array}$$

Beispiel (Fortsetzung)

$$\left.\begin{array}{lcl} z_1\#\Box & \rightarrow & z_2\#\Box RR \end{array}\right\}\quad \text{$\Box$ auf Band 2 $\Rightarrow$ Zustand z_2}$$

$$\left.\begin{array}{lcl} z_2 00 & \rightarrow & z_2 00RR \\ z_2 11 & \rightarrow & z_2 11RR \end{array}\right\}\quad \text{auf beiden Bändern nach rechts gehen, solange gleiche Zeichen gefunden werden}$$

$$\left.\begin{array}{lcl} z_2 01 & \rightarrow & z_2 01NN \\ z_2 10 & \rightarrow & z_2 10NN \end{array}\right\}\quad \text{verschiedene Zeichen $\Rightarrow$ Endlosschleife}$$

$$\left.\begin{array}{lcl} z_2\Box\Box & \rightarrow & z_e\Box\Box NN \end{array}\right\}\quad \text{Alles gleich, daher fertig}$$

Beispiel (Fortsetzung)

$$\left.\begin{array}{lcl}
z_2 0 \Box & \to & z_2 0 \Box NN \\
z_2 1 \Box & \to & z_2 1 \Box NN \\
z_2 \Box 0 & \to & z_2 \Box 0 NN \\
z_2 \Box 1 & \to & z_2 \Box 1 NN
\end{array}\right\} \quad \text{unterschiedliche Länge} \Rightarrow \text{Endlosschleife}$$

$$\left.\begin{array}{lcl}
z_2 \# 0 & \to & z_2 \# 0 NN \\
z_2 \# 1 & \to & z_2 \# 1 NN \\
z_2 \# \Box & \to & z_2 \# \Box NN
\end{array}\right\} \quad \text{Endlosschleife, falls zweites } \# \text{ gefunden wird}$$

Satz

Sei $k > 1$. Zu jeder k-Band-DTM M gibt es eine (1-Band-)DTM M', sodass $L(M) = L(M')$ bzw. dass M und M' dieselbe Funktion berechnen.

Beweisidee:

Sei $M = (Q, \Sigma, \Gamma, \delta, z_0, \Box, E)$ eine k-Band-Maschine.
Wir speichern auf dem Band von M' hintereinander die Inhalte der k Bänder von M, getrennt durch ein spezielles Trennsymbol.
Wir markieren die Positionen der k Köpfe von M.
Simulation eines Schrittes von M: Aktualisierung der gelesenen Zeichen sowie der Kopfpositionen an k Stellen auf dem Band von M'.
Falls notwendig: Bereich für Bänder von M auf dem Band von M' vergrößern.

1-Band nach k-Band

Sei M eine 1-Band-TM. Dann bezeichnet $M(i,k)$ $(1 \leq i \leq k)$, die k-Band-TM, die auf Band i genau die Aktion ausführt, die M auf seinem Band ausführt, und die Bänder $1, \ldots, i-1, i+1, \ldots, k$ unverändert lässt. Ist also z. B. in M

$\delta(z, a) = (z', b, X)$ mit $X \in \{L, N, R\}$, so ergibt sich für $M(2,4)$:

$$\delta(z, c_1, a, c_3, c_4) = (z', c_1, b, c_3, c_4, N, X, N, N)$$

für alle c_1, c_3 und c_4 aus dem Arbeitsalphabet von M (= Arbeitsalpahbet von $M(2,4)$).

Schreibweise: $M(i)$ statt $M(i,k)$, falls k aus dem Kontext klar.

Spezielle Maschinen

„Band := Band + 1“

„Band i := Band i + 1“

„Band i := Band $i - 1$“ (hier: $0 - 1 = 0$)

„Band i := 0“

„Band i := Band j“

Hintereinanderschaltung von Turingmaschinen

Seien $M_i = (Z_i, \Sigma, \Gamma_i, \delta_i, z_{0,i}, \square, E_i)$ mit $i = 1, 2$ zwei DTMn mit o. B. d. A. $Z_1 \cap Z_2 = \emptyset$.
Wir definieren daraus die neue Turingmaschine

$$M = (Z_1 \cup Z_2, \Sigma, \Gamma_1 \cup \Gamma_2, \delta, z_{0,1}, \square, E_2),$$

wobei:

$$\delta(z, a) = \begin{cases} \delta_1(z, a), & \text{falls } z \in Z_1 \setminus E_1 \text{ und } a \in \Gamma_1 \\ \delta_2(z, a), & \text{falls } z \in Z_2 \text{ und } a \in \Gamma_2 \\ (z_{0,2}, a, N), & \text{falls } z \in E_1 \text{ und } a \in \Gamma_1 \end{cases}$$

Bezeichnungen für M: „$M_1;M_2$“ oder
Start $\rightarrow M_1 \rightarrow M_2 \rightarrow$ Stopp. Dies lässt sich analog definieren für mehr als zwei Maschinen.

Bedingte Verzweigungen

$$\begin{tikzcd}
\text{Start} \arrow[d] & & \\
M \arrow[r, "z_{e_1}"] \arrow[d, "z_{e_2}"'] & M_1 \arrow[r] & \text{Stopp} \\
M_2 \arrow[d] & & \\
\text{Stopp} & &
\end{tikzcd}$$

bezeichnet die Turingmaschine, die zuerst M simuliert und vom Endzustand z_{e_1} von M nach M_1 und vom Endzustand z_{e_2} von M nach M_2 übergeht.
Bezeichnung: „IF M THEN M_1 ELSE M_2“, falls $z_{e_1} =$ ja und $z_{e_2} =$ nein.

Test auf Null

Definiere $M = (\{z_0, z_1, \mathrm{ja}, \mathrm{nein}\}, \Sigma, \Gamma, \delta, z_0, \square, \{\mathrm{ja}, \mathrm{nein}\})$ mit

- $\Sigma \supseteq \{0, 1\}$
- $\Gamma \supseteq \{0, 1, \square\}$
- für die Überführungsfunktion δ gilt:

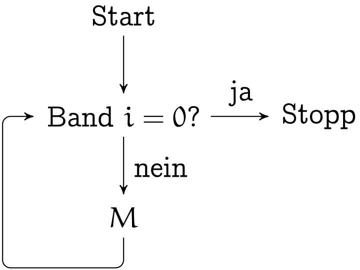
$$
\begin{aligned}
\delta(z_0, a) &= (\mathrm{nein}, a, N) \text{ für } a \in \Gamma \setminus \{0\} \\
\delta(z_0, 0) &= (z_1, 0, R) \\
\delta(z_1, \square) &= (\mathrm{ja}, \square, L) \\
\delta(z_1, a) &= (\mathrm{nein}, a, L) \text{ für } a \in \Gamma \setminus \{0\}
\end{aligned}
$$

Bezeichnung für M: „Band = 0?“.

Schreibweise: „Band i = 0? “ statt „Band = 0? (i)“.

Schleifen

Sei nun M eine beliebige Turingmaschine. „WHILE Band $i \neq 0$ DO M“ bezeichnet dann die Turingmaschine



Die Programmiersprache LOOP

Syntaktische Komponenten von LOOP

- Variablen: $x_0, x_1, x_2, \ldots$
  Zur besseren Lesbarkeit werden wir auch Variablennamen wie z. B. $u, v, x, y, z, \ldots$ benutzen.
- Konstanten: $0, 1, 2, \ldots$
- Operationszeichen: $+$ und $-$
- Trennsymbole: ; und :=
- Schlüsselwörter: LOOP, DO und END

Syntax von LOOP

- Sind x_i und x_j Variablen und c eine Konstante, so sind

$$x_i := x_j + c \quad \text{und} \quad x_i := x_j - c$$

  LOOP-Programme.

- Sind P_1 und P_2 LOOP-Programme, so ist

$$P_1; P_2$$

  ein LOOP-Programm.

- Ist P ein LOOP-Programm und x_i eine Variable, so ist

$$\text{LOOP } x_i \text{ DO } P \text{ END}$$

  ein LOOP-Programm.

Semantik von LOOP

Sei P ein LOOP-Programm. P berechnet eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ wie folgt:
Zu Beginn der Rechnung befinden sich Eingabewerte $n_1, \ldots, n_k \in \mathbb{N}$ in den Variablen $x_1, \ldots, x_k$. Alle anderen Variablen haben den Startwert 0. P wird wie folgt ausgeführt:

- Durch das Programm „$x_i := x_j + c$" erhält x_i den Wert von $x_j + c$.
- Durch das Programm „$x_i := x_j - c$" erhält x_i den Wert von $x_j - c$, falls dieser nicht negativ ist, ansonsten den Wert 0.
- Bei Ausführung von „$P_1; P_2$" wird zunächst P_1 und dann P_2 ausgeführt.
- Ausführung des Programms „LOOP x_i DO P' END":
  P' wird so oft ausgeführt, wie der Wert der Variablen x_i zu Beginn angibt, d. h. Zuweisungen an x_i in P' haben keinen Einfluss auf die Anzahl der Wiederholungen.

Ergebnis der Ausführung von P

$f(n_1, \ldots, n_k) =$ Wert von x_0 am Ende der Ausführung.
Eine Funktion $f\colon \mathbb{N}^k \to \mathbb{N}$ heißt LOOP-berechenbar, falls es ein LOOP-Programm gibt, das f wie soeben festgelegt berechnet.

Beachte: Jedes LOOP-Programm hält nach endlich vielen Schritten an. Daraus folgt, dass jede LOOP-berechenbare Funktion total ist.

Einige spezielle LOOP-Programme

„$x_i := x_j$“

steht für

„$x_i := x_j + 0$“.

„$x_i := c$“ (für eine Konstante c)

steht für

„$x_i := x_j + c$“
(x_j ist eine noch nicht benutzte Variable, die also den Wert 0 hat).

„IF $x_i = 0$ THEN P END“ (für ein LOOP-Programm P)

steht für

„$x_j := 1$;
LOOP x_i DO $x_j := 0$ END;
LOOP x_j DO P END.“
(x_j ist eine Variable, die in P nicht vorkommt)

„$x_i := x_j + x_k$“

steht für

„$x_i := x_j$;
LOOP x_k DO $x_i := x_i + 1$ END.“

„$x_i := x_j * x_k$“

steht für

„$x_i := 0$;
LOOP x_k DO $x_i := x_i + x_j$ END.“

Analog:

„$x_i := x_j$ DIV x_k“

„$x_i := x_j$ MOD x_k“

Die Programmiersprache WHILE

Syntax von WHILE

Erweiterung von LOOP:
neues Schlüsselwort: WHILE
Syntax: Ist P ein WHILE-Programm und x_i eine Variable, so ist

$$\text{WHILE } x_i \neq 0 \text{ DO } P \text{ END}$$

ein WHILE-Programm.

Semantik von WHILE

Die Ausführung von „WHILE $x_i \neq 0$ DO P END“ geschieht so, dass Programm P so lange wiederholt ausgeführt wird, wie der Wert von x_i ungleich Null ist.

P berechnet $f\colon \mathbb{N}^k \to \mathbb{N}$ wie folgt:
Eingabewerte $n_1, \ldots, n_k$ in Variablen $x_1, \ldots, x_k$, die anderen Variablen haben Startwert 0.

$f(n_1, \ldots, n_k)$ ist der Wert von x_0 nach der Ausführung von P, falls diese stoppt, ansonsten ist $f(n_1, \ldots, n_k)$ undefiniert.

Eine Funktion f heißt WHILE-berechenbar, falls es ein WHILE-Programm gibt, das f wie eben festgelegt berechnet.

Beispiel

Das LOOP-Programm

$$\text{LOOP } x \text{ DO } P \text{ END}$$

kann simuliert werden durch

$$y := x;$$
$$\text{WHILE } y \neq 0 \text{ DO } y := y - 1; P \text{ END.}$$

(Dabei ist y eine noch nicht verwendete Variable.)

Korollar

Jedes WHILE-Programm ist äquivalent zu (d. h. berechnet die gleiche Funktion) einem WHILE-Programm, in dem keine LOOP-Schleifen vorkommen.

Erfahrung:

WHILE-Berechenbarkeit = Java-Berechenbarkeit.

Satz

Jede WHILE-berechenbare Funktion ist Turing-berechenbar.

Satz

Jede Turing-berechenbare Funktion ist WHILE-berechenbar.

Die Church'sche These

$$
\begin{aligned}
\text{WHILE-Berechenbarkeit} \quad &= \quad \text{Java-Berechenbarkeit}\\
&= \quad \text{C++-Berechenbarkeit}\\
&= \quad \text{Berechenbarkeit in beliebigen}\\
&\qquad\qquad \text{Programmiersprachen}\\
&= \quad \text{Berechenbarkeit durch Registermaschinen}\\
&= \quad \text{Berechenbarkeit mit Quanten-Computern}\\
&= \quad \text{Markov-Berechenbarkeit}\\
&= \quad \lambda\text{-Berechenbarkeit}\\
&= \quad \mu\text{-Rekursivität}\\
&= \quad \text{Berechenbarkeit in jedem bislang}\\
&\qquad\qquad \text{untersuchten formalen System}
\end{aligned}
$$

$$
\text{WHILE-Berechenbarkeit} \quad = \quad \text{Turing-Berechenbarkeit}
$$

These von Church

Eine Funktion ist berechenbar im intuitiven Sinne, gdw. sie Turing-berechenbar ist.

(Nicht beweisbar, da „berechenbar im intuitiven Sinne“ nicht formal gefasst.)

Manchmal auch: „Church-Turing-These“

Allgemeine Sprechweise:

berechenbar $\equiv$ Turing-berechenbar

Weitere gebräuchliche Bezeichnungen:

rekursiv, partiell rekursiv, total rekursiv

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

- Es gibt totale WHILE-berechenbare Funktionen, die nicht LOOP-berechenbar sind.
  Beispiel: Ackermann-Funktion

Entscheidbarkeit und Aufzählbarkeit

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt entscheidbar, wenn die Funktion $c_A \colon \Sigma^* \to \{0, 1\}$ mit

$$c_A(w) := \begin{cases} 1, & \text{falls } w \in A \\ 0, & \text{sonst} \end{cases}$$

berechenbar ist. c_A heißt charakteristische Funktion von A.

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt semi-entscheidbar, wenn die Funktion
$\chi_A \colon \Sigma^* \to \{0, 1\}$ mit

$$\chi_A(w) := \begin{cases} 1, & \text{falls } w \in A \\ \text{undefiniert}, & \text{sonst} \end{cases}$$

berechenbar ist.

Satz

Eine Sprache ist genau dann semi-entscheidbar, wenn sie vom Typ 0 ist.

Definition

Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen.
A heißt auf B reduzierbar, in Zeichen: $A \leq B$, falls es eine totale, berechenbare Funktion $f \colon \Sigma^* \to \Gamma^*$ gibt, sodass für alle $w \in \Sigma^*$ gilt:

$$w \in A \Leftrightarrow f(w) \in B$$

Lemma

Ist $A \leq B$ und B entscheidbar, so ist A entscheidbar.

Ist $A \leq B$ und B semi-entscheidbar, so ist A semi-entschuldbar.

Beobachtung

Sei $A \subseteq \Sigma^*$. Es gilt:

- A ist entscheidbar $\Longrightarrow$ A ist semi-entscheidbar.
- A ist entscheidbar $\Longleftrightarrow$ $\overline{A}$ ist entscheidbar.
- A ist entscheidbar $\Longrightarrow$ A und $\overline{A}$ sind semi-entscheidbar.

Satz

Sei $A \subseteq \Sigma^*$. Es gilt:

A ist entscheidbar gdw. A und $\overline{A}$ sind semi-entscheidbar.

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv-aufzählbar, falls $A = \emptyset$ oder falls es eine totale berechenbare Funktion $f\colon \mathbb{N} \to \Sigma^*$ gibt, sodass

$$A = \{f(0), f(1), f(2), \ldots\}.$$

Wir sagen: f zählt A auf.

Satz

Eine Sprache ist rekursiv-aufzählbar gdw. sie semi-entscheidbar ist.

Korollar

Eine Sprache A ist entscheidbar gdw. A und $\overline{A}$ rekursiv-aufzählbar sind.

Unentscheidbare Probleme

Erkennen von Endlosschleifen:

Das Halteproblem ist die Sprache

$$\mathrm{H} = \{\, \langle M, x \rangle \mid M \text{ hält bei Eingabe } x \,\}.$$

Gödelisierung

Gödelisierung = Kodierung von Turing-Maschinen durch Binärwörter

Sei $w \in \{0, 1\}^*$. Dann ist

$$M_w := \begin{cases} M, & \text{falls } w \text{ Gödelisierung von } M \\ \widehat{M}, & \text{sonst (d. h. } w \text{ ist keine gültige Gödelisierung),} \end{cases}$$

wobei $\widehat{M}$ eine festgehaltene Turingmaschine ist.

Definition

Das spezielle Halteproblem ist die Sprache

$$K = \{ w \in \{0,1\}^* \mid M_w \text{ hält bei Eingabe } w \}.$$

Das (allgemeine) Halteproblem ist die Sprache

$$H = \{ w\#x \mid M_w \text{ hält bei Eingabe } x \}.$$

Beobachtung

K und H sind rekursiv-aufzählbar.

Satz

K ist nicht entscheidbar.

Korollar

$\overline{K}$ ist nicht rekursiv-aufzählbar.

Satz

H ist nicht entscheidbar.

Satz

Eine Sprache $A \subseteq \Sigma^*$ ist rekursiv-aufzählbar gdw. es eine berechenbare Funktion $f \colon \mathbb{N} \to \Sigma^*$ gibt, sodass

$$A = \{f(0), f(1), f(2), \ldots\}.$$

Satz

Eine Sprache $A \subseteq \Sigma^*$ ist rekursiv-aufzählbar gdw. es eine entscheidbare Sprache B gibt, sodass

$$A = \{x \in \Sigma^* \mid \exists y : \langle x, y \rangle \in B\}.$$

Zusammenfassung

Sei A eine Sprache. Aus den bisherigen Resultaten ergibt sich, dass die folgenden Aussagen äquivalent sind:

1. A ist vom Typ 0.
2. $A = L(M)$ für eine Turingmaschine M.
3. A ist semi-entscheidbar.
4. A ist rekursiv-aufzählbar.
5. A ist Wertebereich einer totalen berechenbaren Funktion oder $A = \emptyset$.
6. A ist Wertebereich einer (eventuell partiellen) berechenbaren Funktion.
7. A ist Definitionsbereich einer berechenbaren Funktion.
8. Es gibt eine entscheidbare Sprache B sodass $A = \{x \in \Sigma^* \mid \exists y : \langle x, y \rangle \in B\}$.

Korollar

Die Klasse der Typ-1-Sprachen ist eine echte Teilmenge der Klasse der Typ-0-Sprachen.

Satz von Rice

Sei $\mathcal{R}$ die Klasse aller berechenbaren Funktionen. Sei $\mathcal{S} \subseteq \mathcal{R}$ mit $\mathcal{S} \neq \emptyset$ und $\mathcal{S} \neq \mathcal{R}$. Dann ist die Sprache

$$\mathrm{C}(\mathcal{S}) = \{w \mid \text{die von } \mathrm{M}_w \text{ berechnete Funktion ist aus } \mathcal{S}\}$$

nicht entscheidbar.

Definition

Das Halteproblem auf leerem Band ist die Sprache

$$H_0 = \{w \mid M_w \text{ angesetzt auf leerem Band hält}\}.$$

Satz

H_0 ist nicht entscheidbar.

Satz

Sei $\mathcal{R}$ die Klasse aller berechenbaren Funktionen. Sei $\mathcal{S} \subseteq \mathcal{R}$ mit $\mathcal{S} \neq \emptyset$ und $\mathcal{S} \neq \mathcal{R}$. Die Sprache $\mathrm{C}(\mathcal{S})$ sei definiert als

$$\mathrm{C}(\mathcal{S}) = \{w \mid \text{die von } M_w \text{ berechnete Funktion ist aus } \mathcal{S}\}.$$

Dann gilt:

$$\mathrm{K} \leq \mathrm{C}(\mathcal{S}) \text{ oder } \overline{\mathrm{K}} \leq \mathrm{C}(\mathcal{S})$$

Korollar (Satz von Rice)

Sei $\mathcal{R}$ die Klasse aller berechenbaren Funktionen. Sei $\mathcal{S} \subseteq \mathcal{R}$ mit $\mathcal{S} \neq \emptyset$ und $\mathcal{S} \neq \mathcal{R}$. Dann ist die Sprache

$$\mathrm{C}(\mathcal{S}) = \{w \mid \text{die von } \mathrm{M}_w \text{ berechnete Funktion ist aus } \mathcal{S}\}$$

nicht entscheidbar.

Korollar

Die folgenden Sprachen sind nicht entscheidbar:

- $\{w \mid M_w \text{ berechnet eine totale Funktion}\}$
  „Das gegebene Programm stürzt nicht ab."
- $\{w \mid M_w \text{ berechnet eine monotone Funktion}\}$
- $\{w \mid M_w \text{ berechnet eine konstante Funktion}\}$
- $\{w \mid M_w \text{ berechnet die Funktion } f(x) = x + 1\}$
  „Das gegebene Programm erfüllt eine gegebene Spezifikation"
  (hier im Beispiel: „Das gegebene Programm berechnet die Nachfolgerfunktion").