

Grundlagen der Theoretischen Informatik

Wintersemester 2015/2016

Prof. Dr. Heribert Vollmer

Institut für Theoretische Informatik
Leibniz Universität Hannover

15. November 2015

| | |
|---|--|
| Sprachen und Grammatiken | Der intuitive Berechenbarkeitsbegriff |
| Die Chomsky-Hierarchie | Berechenbarkeit durch Maschinen |
| Reguläre (Typ-3-) Sprachen | Turing-Berechenbarkeit |
| Endliche Automaten | Mehrband-Maschinen |
| Nichtdeterministische endliche Automaten | Zusammensetzung von Turingmaschinen |
| Endliche Automaten und Typ-3-Grammatiken | Berechenbarkeit in Programmiersprachen |
| Das Pumping Lemma für reguläre Sprachen | Die Programmiersprache LOOP |
| Kontextfreie (Typ-2-) Sprachen | Die Programmiersprache WHILE |
| Kellerautomaten | Die Church'sche These |
| Das Pumping-Lemma für kontextfreie Sprachen | Entscheidbarkeit und Aufzählbarkeit |
| Typ-1- und Typ-0-Sprachen | Unentscheidbare Probleme |
| | Das Halteproblem |
| | Der Satz von Rice |

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.

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

5

6

Konkatenation

- ▶ Für ein Wort w und $n \in \mathbb{N}$ ist w^n die Konkatenation
$$w^n = \underbrace{w \circ w \circ \dots \circ w}_{n\text{-mal}}$$
- ▶ Wir definieren: $w^0 = \varepsilon$.
Es ist $|w^n| = n|w|$.
Schreibweise: $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$

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**

7

8

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 $y \rightarrow y'$ 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 $w_1, \dots, w_k \in (V \cup \Sigma)^*$ gibt mit $u = w_1$, $w_i \Rightarrow_G w_{i+1}$ für $i = 1, 2, \dots, k-1$ 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, \dots, w_k) mit $w_0 = S$, $w_k = w$ und $w_i \Rightarrow_G w_{i+1}$ für $i = 0, 1, \dots, k-1$.

9

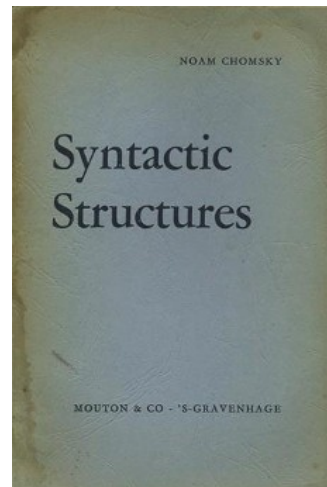
Die Chomsky-Hierarchie

10

Noam Chomsky



* 7. Dez. 1928, Philadelphia
1957: **Syntactic Structures**



11

Definition

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

12

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 $L = L(G)$.

13

14

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

15

16

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, $Z \cap \Sigma = \emptyset$
- ▶ $\delta: Z \times \Sigma \rightarrow Z$ ist die so genannte Überföhrungsfunktion
- ▶ $z_0 \in Z$ ist der so genannte Startzustand
- ▶ $E \subseteq Z$ ist die Menge der so genannten Endzustände

17

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 Überföhrungsfunktion gilt: $\delta: Z \times \Sigma \rightarrow \mathcal{P}(Z)$. $\mathcal{P}(Z)$ ist die Potenzmenge von Z . Für $z \in Z$ und $a \in \Sigma$ ist also $\delta(z, a)$ eine Menge von möglichen Folgezuständen

19

Definition

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

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

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

Die von M akzeptierte Sprache ist

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

18

Definition

Wir definieren $\hat{\delta}: \mathcal{P}(Z) \times \Sigma^* \rightarrow \mathcal{P}(Z)$ wie folgt:

$$\hat{\delta}(Z', \varepsilon) = Z' \text{ für alle } Z' \subseteq Z$$

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

Die von M akzeptierte Sprache ist

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

20

Satz

Zu jedem NEA M existiert ein DEA M' mit $L(M) = L(M')$.

21

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.

22

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: $uv^i w \in L$.

23

Logische Struktur der Aussage des Pumping-Lemmas:

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

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.

24

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
- ▶ $\delta: Z \times \Sigma \times \Gamma \rightarrow \mathcal{P}(Z \times \Gamma^*)$ ist die Überföhrungsfunktion. 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

25

26

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:

$\delta(z, a, A) \ni (z', B_1, \dots, B_k)$ 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, \dots, B_k (B_1 wird oberstes Kellersymbol) ersetzen. Der Eingabekopf wandert eine Position nach rechts.

($z, z' \in Z$, $a \in \Sigma$, $A, B_1, \dots, B_k \in \Gamma$.)

27

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.

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

28

Satz

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

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: $uv^iwx^iy \in L$

29

30

Logische Struktur der Aussage des Pumping-Lemmas:

$$(L \text{ kontextfrei}) \Rightarrow (\exists n \in \mathbb{N})(\forall z \in L, |z| \geq n)(\exists u, v, w, x, y),$$
$$\underbrace{[z = uvwxy \wedge (1)-(3) \text{ gelten}]}_{(*)}$$

Anwendung: Kontraposition des Satzes, also:

$(*)$ gilt nicht $\Rightarrow L$ ist nicht kontextfrei.

Typ-1- und Typ-0-Sprachen

31

32

Definition

Eine Turingmaschine (TM) ist ein 7-Tupel

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

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
- ▶ $\square \in \Gamma \setminus \Sigma$ ist das Leerzeichen bzw. Blank
- ▶ $E \subseteq Z$ ist die Menge der Endzustände
- ▶ δ ist die Übergangsfunktion

Definition

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

Definition (Fortsetzung)

Bei deterministischen Turingmaschinen (DTM, TM) gilt:

$$\delta: Z \times \Gamma \rightarrow Z \times \Gamma \times \{L, N, R\}$$

Bei nichtdeterministischen Turingmaschinen (NTM) gilt:

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

Definition

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

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

$m = 0$, Maschine läuft nach **links**:

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

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

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

37

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

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

39

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, \dots, k_n gibt mit

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

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

38

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 $L(M) = L$

40

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

41

42

Berechenbarkeit

Eine Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **berechenbar**, falls es ein Rechenverfahren bzw. einen Algorithmus gibt, das f berechnet, d. h. gestartet mit Eingabe $(n_1, \dots, n_k) \in \mathbb{N}^k$ hält der Algorithmus nach endlich vielen Schritten mit Ausgabe $f(n_1, \dots, n_k)$.

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

Ziel: Präzisierung des Berechenbarkeitsbegriffs.

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

43

44

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

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

Beispiel 4

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

45

46

Turing-Berechenbarkeit

Definition

Eine Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ heißt **Turing-berechenbar**, falls es eine DTM M gibt, sodass für alle $n_1, \dots, n_k, m \in \mathbb{N}$ gilt:

$$f(n_1, \dots, n_k) = m \Rightarrow$$

M mit Eingabe $\text{bin}(n_1)\# \dots \# \text{bin}(n_k)$
 hält mit $\square \dots \square \text{bin}(m) \square \dots \square$
 auf dem Arbeitsband.

$$f(n_1, \dots, n_k) \text{ undefiniert} \Rightarrow$$

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

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

47

48

Bemerkung

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

Definition

Eine Funktion $f: \Sigma^* \rightarrow \Delta^*$ 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.

49

50

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:

► $Z, \Sigma, \Gamma, \square$ und E sind wie bei einer 1-Band-DTM definiert.

► $\delta: \underbrace{Z}_{(i)} \times \underbrace{\Gamma^k}_{(ii)} \rightarrow \underbrace{Z}_{(iii)} \times \underbrace{\Gamma^k}_{(iv)} \times \underbrace{\{L, R, N\}^k}_{(v)}$ 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

51

52

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 $\text{bin}(n_1)\# \dots \# \text{bin}(n_k)$ erreicht irgendwann einen Endzustand mit $\text{bin}(m)$ auf Band 1.

(Berechnung von Funktionen $f: \Sigma^* \rightarrow \Delta^*$ analog.)

Zusammensetzung von Turingmaschinen

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 $M = (Q, \Sigma, \Gamma, \delta, z_0, \square, E)$. Wir unterteilen das Band von M' in $2k$ „Spuren“, in denen wir die Inhalte von k Bändern von M sowie die Position der k Köpfe von M speichern.

53

54

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 $1, \dots, i-1, i+1, \dots, k$ 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 (= Arbeitsalphabet von $M(2, 4)$).

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

55

56

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:

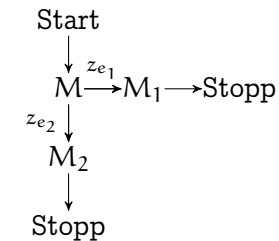
$$\delta(z, a) = \begin{cases} \delta_1(z, a), & \text{falls } z \in Z_1 \setminus E_1 \text{ und } a \in \Gamma_1 \\ \delta_2(z, a), & \text{falls } z \in Z_2 \text{ und } a \in \Gamma_2 \\ (z_{0,2}, a, N), & \text{falls } z \in E_1 \text{ und } a \in \Gamma_1 \end{cases}$$

Bezeichnungen für M: „ $M_1; M_2$ “ oder

$\text{Start} \rightarrow M_1 \rightarrow M_2 \rightarrow \text{Stopp}$. Dies lässt sich analog definieren für mehr als zwei Maschinen.

57

Bedingte Verzweigungen



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: „ $\text{IF } M \text{ THEN } M_1 \text{ ELSE } M_2$ “, falls $z_{e_1} = \text{ja}$ und $z_{e_2} = \text{nein}$.

58

Test auf Null

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

- ▶ $\Sigma \supseteq \{0, 1\}$
- ▶ $\Gamma \supseteq \{0, 1, \square\}$
- ▶ für die Überföhrungsfunktion δ gilt:
 - $\delta(z_0, a) = (\text{nein}, a, N)$ für $a \in \Gamma \setminus \{0\}$
 - $\delta(z_0, 0) = (z_1, 0, R)$
 - $\delta(z_1, \square) = (\text{ja}, \square, L)$
 - $\delta(z_1, a) = (\text{nein}, a, L)$ für $a \in \Gamma \setminus \{0\}$

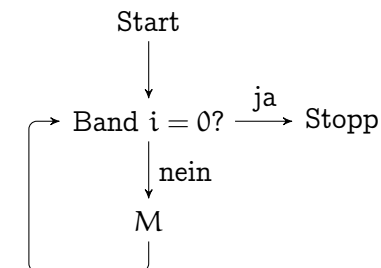
Bezeichnung für M: „ $\text{Band} = 0?$ “.

Schreibweise: „ $\text{Band } i = 0?$ “ statt „ $\text{Band} = 0? (i)$ “.

59

Schleifen

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



60

Die Programmiersprache LOOP

Syntaktische Komponenten von LOOP

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

61

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.

63

Semantik von LOOP

Sei P ein LOOP-Programm. P berechnet eine Funktion

$f: \mathbb{N}^k \rightarrow \mathbb{N}$ wie folgt:

Zu Beginn der Rechnung befinden sich Eingabewerte

$n_1, \dots, n_k \in \mathbb{N}$ in den Variablen x_1, \dots, 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 „ $\text{LOOP } x_i \text{ DO } P' \text{ END}$ “:
 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.

62

64

Ergebnis der Ausführung von P

$f(n_1, \dots, n_k)$ = Wert von x_0 am Ende der Ausführung.

Eine Funktion $f: \mathbb{N}^k \rightarrow \mathbb{N}$ 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.

65

„ $x_i := c$ “ (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).

67

Einige spezielle LOOP-Programme

„ $x_i := x_j$ “

steht für

„ $x_i := x_j + 0$ “.

66

„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)

68

„ $x_i := x_j + x_k$ “

steht für

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

69

„ $x_i := x_j * x_k$ “

steht für

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

70

Analog:

„ $x_i := x_j \text{ DIV } x_k$ “

„ $x_i := x_j \text{ MOD } x_k$ “

71

Die Programmiersprache WHILE

72

Syntax von WHILE

Erweiterung von LOOP:

neues **Schlüsselwort**: WHILE

Syntax: Ist P ein WHILE-Programm und x_i eine Variable, so ist

WHILE $x_i \neq 0$ DO P END

ein WHILE-Programm.

73

Beispiel

Das LOOP-Programm

LOOP x DO P END

kann simuliert werden durch

$y := x;$

WHILE $y \neq 0$ DO $y := y - 1; P$ END.

(Dabei ist y eine noch nicht verwendete Variable.)

75

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 $f: \mathbb{N}^k \rightarrow \mathbb{N}$ wie folgt:

Eingabewerte n_1, \dots, n_k in Variablen x_1, \dots, x_k , die anderen Variablen haben Startwert 0.

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

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

74

Korollar

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

76

Erfahrung:

WHILE-Berechenbarkeit = Java-Berechenbarkeit.

Satz

Jede WHILE-berechenbare Funktion ist Turing-berechenbar.

Satz

Jede Turing-berechenbare Funktion ist WHILE-berechenbar.

77

WHILE-Berechenbarkeit = Java-Berechenbarkeit
= C++-Berechenbarkeit
= Berechenbarkeit in beliebigen
 Programmiersprachen
= Berechenbarkeit mit Quanten-Computern
= Markov-Berechenbarkeit
= λ -Berechenbarkeit
= Berechenbarkeit in jedem bislang
 untersuchten formalen System
WHILE-Berechenbarkeit = **Turing-Berechenbarkeit**

79

Die Church'sche These

78

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.)

80

Entscheidbarkeit und Aufzählbarkeit

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt **entscheidbar**, wenn die Funktion $c_A: \Sigma^* \rightarrow \{0, 1\}$ 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 .

81

82

Definition

Eine Sprache $A \subseteq \Sigma^*$ heißt **semi-entscheidbar**, wenn die Funktion

$\chi_A: \Sigma^* \rightarrow \{0, 1\}$ mit

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

berechenbar ist.

Beobachtung

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

- ▶ A ist entscheidbar $\implies A$ ist semi-entscheidbar.
- ▶ A ist entscheidbar $\iff \bar{A}$ ist entscheidbar.
- ▶ A ist entscheidbar $\implies A$ und \bar{A} sind semi-entscheidbar.

83

84

Satz

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

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

85

Satz

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

87

Definition

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

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

Wir sagen: f zählt A auf.

86

Korollar

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

88

Satz

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

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

89

Unentscheidbare Probleme

91

Satz

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

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

90

Erkennen von Endlosschleifen:

Das **Halteproblem** ist die Sprache

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

92

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.

Beobachtung

K ist rekursiv-aufzählbar.

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

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

Satz

K ist nicht entscheidbar.

Korollar

\bar{K} ist nicht rekursiv-aufzählbar.

97

Lemma

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

99

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 $f: \Sigma^* \rightarrow \Gamma^*$ gibt, sodass für alle $w \in \Sigma^*$ gilt:

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

98

Satz

H ist nicht entscheidbar.

100

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 rekursiv-aufzählbar
4. A ist Wertebereich einer totalen berechenbaren Funktion oder $A = \emptyset$
5. A ist semi-entscheidbar
6. A ist Definitionsbereich einer berechenbaren Funktion
7. A ist Wertebereich einer (eventuell partiellen) berechenbaren Funktion

101

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

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

nicht entscheidbar.

103

Korollar

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

102

Definition

Das **Halteproblem auf leerem Band** ist die Sprache

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

104

Satz

H_0 ist nicht entscheidbar.

105

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

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

nicht entscheidbar.

107

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 $C(\mathcal{S})$ sei definiert als

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

Dann gilt:

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

106

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 Spezifikation.“

108