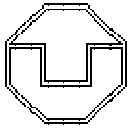
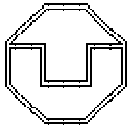
Inhaltsangabe



Typ-0-Sprache	$\longleftrightarrow$	Turing-akzeptierbare Sprache
Ableitung	$\longleftrightarrow$	Konfigurationsfolge

Beim Übergang von der Turingmaschine zur Grammatik dreht sich allerdings die Richtung um:

- eine akzeptierende Konfigurationsfolge beginnt mit dem zu akzeptierenden Wort
- eine Ableitung endet mit dem erzeugten Wort



Satz 12.1

Eine Sprache L gehört zu $\mathcal{L}_0$ gdw sie Turing-akzeptierbar ist.

Beweis:

„$\Rightarrow$“: Es sei $L = L(G)$ für eine Typ-0-Grammatik $G = (N, \Sigma, P, S)$ mit $|P| = k$ Regeln.

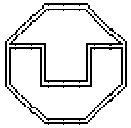
Wir geben eine 2-Band-NTM an, die $L(G)$ akzeptiert:

1. Band: speichert Eingabe w

2. Band: ausgehend von S werden gemäß den Regeln von G alle ableitbaren Wörter gebildet

Es wird verglichen, ob auf Band 2 irgendwann w (d.h. der Inhalt von Band 1) entsteht.

Wenn ja, so geht man in akzeptierende Stoppkonfiguration, sonst sucht die Turingmaschine weiter.



Die Maschine geht dabei wie folgt vor:

1. Schreibe S auf Band 2, gehe nach links auf das $\not b$ vor S.

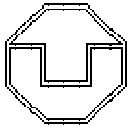
2. Gehe auf Band 2 nach rechts und wähle (nichtdeterministisch) eine Stelle aus, an der die linke Seite der anzuwendenden Produktion beginnen soll.

3. Wechsle nun (nichtdeterministisch) in einen der Zustände $\text{Regel}_1, \ldots, \text{Regel}_k$

   (entspricht der Entscheidung, daß man diese Regel anwenden will).

4. Es sei Regel_i der gewählte Zustand und $\alpha \longrightarrow \beta$ die entsprechende Produktion.

   Überprüfe, ob α tatsächlich die Beschriftung des Bandstücks der Länge $|\alpha|$ ab der gewählten Bandstelle ist.



5. Falls der Test erfolgreich war, so ersetze α durch β.

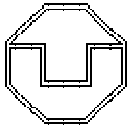
   Vorher müssen bei $|\alpha| < |\beta|$ die Symbole $\neq \not{b}$ rechts von α um $|\beta| - |\alpha|$ Positionen nach rechts

   (bzw. bei $|\alpha| > |\beta|$ um $|\alpha| - |\beta|$ Positionen nach links) geschoben werden.

6. Gehe nach links bis zum ersten $\not{b}$ und vergleiche, ob die Inschrift auf dem Band 1 mit der auf Band 2 übereinstimmt.

7. Wenn ja, so gehe in akzeptierenden Stoppzustand.

   Sonst fahre fort bei 2.



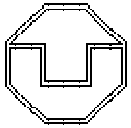
Beispiel:

Betrachte die Grammatik mit den Produktionen

$$\begin{aligned} S &\longrightarrow aSb \\ aSb &\longrightarrow ab \end{aligned}$$

Das Wort $w = aabb$ gehört zu der davon erzeugten Sprache:

$$S \longrightarrow aSb \longrightarrow aaSbb \longrightarrow aabb$$



„$\Leftarrow$“: Es sei $L = L(\mathcal{A})$ für eine NTM $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$.

Wir konstruieren eine Grammatik G, die jedes Wort $w \in L(\mathcal{A})$ wie folgt erzeugt:

1. Phase: Erst wird w mit „genügend vielen“ $\not b$-Symbolen links und rechts davon erzeugt

   Dies passiert für jedes w, auch für $w \notin L(\mathcal{A})$).

   „Genügend viele“: so viele, wie $\mathcal{A}$ beim Akzeptieren von w vom Arbeitsband benötigt.

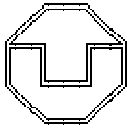
2. Phase: Auf dem so erzeugten Arbeitsband simuliert G die Berechnung von $\mathcal{A}$ bei Eingabe w.

3. Phase: War die Berechnung akzeptierend, so erzeuge nochmals w.

Damit man in der zweiten Phase das in der dritten Phase benötigte w nicht vergißt, verwendet man als Nichtterminalsymbole Tupel aus

$$(\Sigma \cup \{\not b\}) \times \Gamma,$$

wobei man sich in der ersten Komponente w merkt und in der zweiten Komponente die Berechnung simuliert.



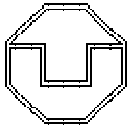
Formale Definition der Grammatik:

$$N = \{S, A, B, E\} \cup Q \cup (\Sigma \cup \{\not b\}) \times \Gamma,$$

wobei

- S, A, B zum Aufbau des Rechenbandes am Anfang,
- E zum Löschen am Schluß,
- Q zur Darstellung des aktuellen Zustands von $\mathcal{A}$,
- $\Sigma \cup \{\not b\}$ zum Speichern von w und
- Γ zur $\mathcal{A}$-Berechnung

dienen.

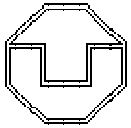


Regeln der ersten Phase: Erzeuge w und ein genügend großes Arbeitsband

$$\begin{aligned} S &\longrightarrow Bq_0A \\ A &\longrightarrow [a,a]A \text{ für alle } a \in \Sigma, \\ A &\longrightarrow B, \\ B &\longrightarrow [\not b, \not b]B, \\ B &\longrightarrow \varepsilon. \end{aligned}$$

Man erhält also somit für alle $a_1 \dots a_n \in \Sigma^*, k, l, \geq 0$:

$$S \vdash_G^* [\not b, \not b]^k q_0 [a_1, a_1] \dots [a_n, a_n] [\not b, \not b]^l.$$



Dresden

Regeln der 2. Phase: simuliert TM-Berechnung in der „zweiten Spur“:

- $p[a,b] \longrightarrow [a,b']q$

  falls $(p,b,b',r,q) \in \Delta, a \in \Sigma \cup \{\not b\}$

- $[a,c]p[a',b] \longrightarrow q[a,c][a',b']$

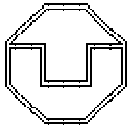
  falls $(p,b,b',l,q) \in \Delta, a,a' \in \Sigma \cup \{\not b\}, c \in \Gamma$

- $p[a,b] \longrightarrow q[a,b']$

  falls $(p,b,b',n,q) \in \Delta, a \in \Sigma \cup \{\not b\}$

Beachte:

Da wir in der ersten Phase genügend viele Blanks links und rechts von $a_1 \ldots a_n$ erzeugen können, muß in der zweiten Phase das „Nachschieben“ von Blanks am Rand nicht mehr behandelt werden.



3. Phase: Aufräumen und erzeugen von $a_1 \ldots a_n$, wenn die TM akzeptiert hat

- $q[a,b] \longrightarrow EaE$ für $a \in \Sigma, b \in \Gamma$
  $q[\not b, b] \longrightarrow E$ für $b \in \Gamma$

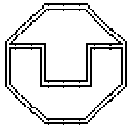
  falls $q \in F$ und es keine Transition der Form $(q, b, \ldots, \ldots) \in \Delta$ gibt (d.h. akzeptierende Stoppkonfiguration erreicht)

- $E[a,b] \longrightarrow aE$

  für $a \in \Sigma, b \in \Gamma$
  (Aufräumen nach rechts)

- $[a,b]E \longrightarrow Ea$

  für $a \in \Sigma, b \in \Gamma$
  (Aufräumen nach links)



- $E[\not b, b] \longrightarrow E$ für $b \in \Gamma$

  (Entfernen des zusätzlich erzeugten Arbeitsbandes nach rechts)

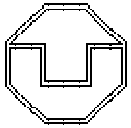
- $[\not b, b]E \longrightarrow E$ für $b \in \Gamma$

  (Entfernen des zusätzlich erzeugten Arbeitsbandes nach links)

- $E \longrightarrow \varepsilon$

Man sieht nun leicht, daß für alle $w \in \Sigma^*$ gilt:

$$w \in L(G) \text{ gdw } \mathcal{A} \text{ akzeptiert } w.$$



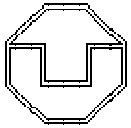
Beispiel:

Betrachte die Turingmaschine mit den Übergängen:

q_0	a	a	r	q_0
q_0	b	b	r	q_1
q_1	b	b	r	q_1
q_1	$\not b$	$\not b$	n	q_{akz}

Das Wort $w = abb$ wird von dieser TM akzeptiert:

$$\not b q_0 abb \not b \vdash \not b a q_0 bb \not b \vdash \not b ab q_1 b \not b \vdash \not b abb q_1 \not b \vdash \not b abb q_{akz} \not b$$



Dresden

Satz 12.2 (Abschlußeigenschaften)

$\mathcal{L}_0$ ist abgeschlossen unter $\cup, \cdot, ^*$ und $\cap$.

Beweis:

$$\begin{aligned} S &\longrightarrow S_1 \mid S_2 \\ S &\longrightarrow S_1 S_2 \\ S &\longrightarrow S S_1 \mid \varepsilon \end{aligned}$$

(1)

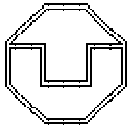
Für die regulären Operationen $\cup, \cdot, ^*$ zeigt man dies im Prinzip wie für $\mathcal{L}_2$ durch Konstruktion einer entsprechenden Grammatik.

Damit sich die Produktionen der verschiedenen Grammatiken nicht gegenseitig beeinflussen, genügt es allerdings nicht mehr, nur die Nichtterminalsymbole der Grammatiken disjunkt zu machen.

Zusätzlich muß man die Grammatiken in die folgende Form bringen:

Die Produktionen sind von der Form

$$\begin{aligned} u &\longrightarrow v \quad \text{mit } u \in N_i^+ \text{ und } v \in N_i^* \\ X_a &\longrightarrow a \quad \text{mit } X_a \in N_i \text{ und } a \in \Sigma \end{aligned}$$



Für den Durchschnitt verwendet man Turingmaschinen.

Die NTM für $L_1 \cap L_2$ verwendet zwei Bänder und simuliert zunächst auf dem ersten die Berechnung der NTM für L_1 und dann auf dem zweiten die der NTM für L_2.

Wenn beide zu akzeptierenden Stoppkonfigurationen der jeweiligen Turingmaschinen führen, so geht die NTM für $L_1 \cap L_2$ in eine akzeptierende Stoppkonfiguration.

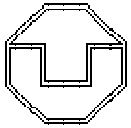
Beachte:

Es kann sein, daß die NTM für L_1 auf einer Eingabe w nicht terminiert, die NTM für $L_1 \cap L_2$ also gar nicht dazu kommt, die Berechnung der NTM für L_2 zu simulieren.

Aber dann ist ja w auch nicht in $L_1 \cap L_2$.

Beachte:

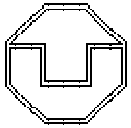
Man kann zeigen, daß die Klasse der Turing-akzeptierbare Sprachen nicht unter Komplement abgeschlossen ist.



Man kann außerdem zeigen, daß für Turing-akzeptierbare Sprachen (und damit für $\mathcal{L}_0$) alle bisher betrachteten Entscheidungsprobleme unentscheidbar sind.

Satz 12.3 (Entscheidungsprobleme)

Für $\mathcal{L}_0$ sind das Leerheitsproblem, das Wortproblem und das Äquivalenzproblem unentscheidbar.



Automaten/Maschinenmodell für $\mathcal{L}_1$ (kontextsensitive Sprachen)?

Spezielle Turingmaschinen, die nur den Bandabschnitt verwenden dürfen, auf dem anfangs die Eingabe steht.

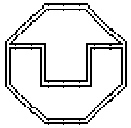
Intuition:

Die Beweisidee von Satz 12.1 kann geeignet angepaßt werden:

da ks Grammatiken nicht-kürzend sind, benötigt man zum Erzeugen von w nicht mehr Platz als $|w|$.

Bandbeschränkung:

Um ein Überschreiten der durch $|w|$ gegebenen Bandgrenzen zu verhindern, verwendet man Randmarker $\not\in$, \$.



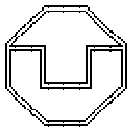
Definition 12.4 (linear beschränkter Automat)

Ein linear beschränkter Automat (LBA) ist eine NTM $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \Delta, F)$, so daß gilt:

- $\$, \not\in \Gamma \setminus \Sigma$
- Übergänge $(q, \not\in, \ldots)$ sind nur in der Form $(q, \not\in, \not\in, r, q')$ erlaubt (linker Rand darf nicht überschritten werden).
- Übergänge $(q, \$, \ldots)$ sind nur in der Form $(q, \$, \$, l, q')$ erlaubt (rechter Rand darf nicht überschritten werden).
- Randmarker $\not\in$ und $\$$ dürfen nicht geschrieben werden.

Ein gegebener LBA $\mathcal{A}$ akzeptiert die Sprache

$$L(\mathcal{A}) := \{w \in \Sigma^* \mid \text{es gibt eine akzeptierende Stoppkonfiguration } k \text{ mit } \not\in q_0 w\$ \vdash^* k\}.$$



Dresden

Korollar 12.5

Eine Sprache L gehört zu $\mathcal{L}_1$ gdw sie von einem LBA akzeptiert wird.

Beweis:

„$\Rightarrow$“: Verwende die Konstruktion aus dem Beweis von Satz 12.1.

Ausnahme
$S \longrightarrow \varepsilon$

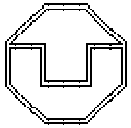
Da alle Produktionen von kontextsensitiven Grammatiken nicht-kürzend sind muß man auf dem zweiten Band nur ableitbare Wörter bis zur Länge $|w|$ erzeugen

(längere können nie mehr zu w abgeleitet werden).

Daher kommt man mit $|w|$ vielen Feldern aus.

Beachte:

Zwei Bänder liefern nicht doppelt so langes Band, sondern größeres Alphabet.



„$\Leftarrow$“:

Durch Modifikation der Beweisidee von Satz 12.1 gelingt es, zu einem LBA eine Grammatik zu konstruieren, die nur nicht-kürzenden Regeln hat:

$$\alpha \longrightarrow \beta \quad \text{mit } |\alpha| \leq |\beta|.$$

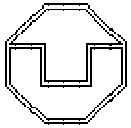
Wie im Beweis von Satz 9.4 illustriert, kann man diese in eine kontextsensitive Grammatik transformieren.

Idee:

Da man mit $|w|$ Arbeitsfeldern auskommt, muß man keine $[\not b,\ \not b]$ links und rechts von w erzeugen.

Dadurch fallen dann auch die folgenden kürzenden Regeln weg:

$$\begin{aligned} E[\not b, b] &\longrightarrow E, \\ [\not b, b]E &\longrightarrow E. \end{aligned}$$



Erste Phase:

$$
\begin{array}{lcl}
S & \longrightarrow & Bq_0A \\
A & \longrightarrow & [a,a]A \ \text{ für alle } a \in \Sigma, \\
A & \longrightarrow & B, \\
B & \longrightarrow & [\not b, \not b]B, \\
B & \longrightarrow & \varepsilon.
\end{array}
$$

Erste Phase bei LBA:

$$
\begin{array}{lcl}
S & \longrightarrow & [\not\in, \not\in]q_0A \\
A & \longrightarrow & [a,a]A \ \text{ für alle } a \in \Sigma, \\
A & \longrightarrow & [\$,\$]
\end{array}
$$

2. Phase:

$$
\begin{array}{lcll}
p[a,b] & \longrightarrow & [a,b']q & \text{falls ...} \\
[a,c]p[a',b] & \longrightarrow & q[a,c][a',b'] & \text{falls ...} \\
p[a,b] & \longrightarrow & q[a,b'] & \text{falls ...}
\end{array}
$$

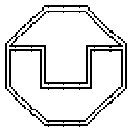
Alle Regeln
nicht-kürzend

3. Phase:

$$
\begin{array}{lcll}
q[a,b] & \longrightarrow & EaE & \text{falls ...} \\
q[\not b,b] & \longrightarrow & E & \text{falls ...} \\
E[a,b] & \longrightarrow & aE & \\
[a,b]E & \longrightarrow & Ea & \\
E[\not b,b] & \longrightarrow & E & \\
[\not b,b]E & \longrightarrow & E & \\
E & \longrightarrow & \varepsilon &
\end{array}
$$

3. Phase bei LBA:

$$
\begin{array}{lcll}
q[a,b] & \longrightarrow & EaE & \text{falls ...} \\
E[a,b] & \longrightarrow & aE & \\
[a,b]E & \longrightarrow & Ea & \\
[\not\in,\not\in]E & \longrightarrow & E & \} \\
E[\$,\$] & \longrightarrow & E & \} \ \textit{kürzend} \\
E & \longrightarrow & \varepsilon &
\end{array}
$$



Dresden

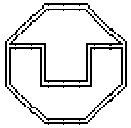
Probleme gibt es also mit den folgenden Transitionen:

$$\begin{array}{lcl} q[a,b] & \longrightarrow & EaE \\ [\not\in,\not\in]E & \longrightarrow & E \\ E[\$,\$] & \longrightarrow & E \\ E & \longrightarrow & \varepsilon \end{array}$$

Lösungsidee:

Führe die Symbole $\not\in$, $\$$ und den Zustand q nicht als zusätzliche Symbole ein, sondern kodiere sie in die anderen Symbole hinein:

Z.B. statt $[a,b]q[a',b'][a'',b'']$ verwende $[a,b][q,a',b'][a'',b'']$.



Satz 12.6

$\mathcal{L}_1$ ist abgeschlossen unter $\cup, \cdot, ^*, \cap$ und Komplement.

Beweis:

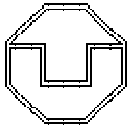
Für $\cup, \cdot, ^*$ und $\cap$ geht dies wie bei $\mathcal{L}_0$.

Komplement: schwierig

Das Problem war lange offen und wurde dann in den 1980ern unabhängig von zwei Forschern gelöst (Immerman und Szelepcsenyi).

Beachte:

Für LBAs ist bisher nicht bekannt, ob deterministische LBAs genauso stark wie nichtdeterministische LBAs sind.



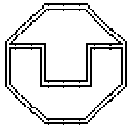
Satz 12.7

Für $\mathcal{L}_1$ ist das Wortproblem entscheidbar.

Beweis:

Da kontextsensitive Produktionen (bis auf Spezialfall $S \longrightarrow \varepsilon$) nichtkürzend sind, muß man zur Entscheidung „$w \in L(G)$?“ nur alle aus S ableitbaren Wörter aus $(N \cup \Sigma)^*$ der Länge $\leq |w|$ erzeugen.

Dies sind endlich viele.



Ausblick: Berechenbarkeit und Entscheidbarkeit

Aus der Sicht der Theorie der formalen Sprachen liefern Turingmaschinen Automatenmodelle für die Typ-0- und die Typ-1-Sprachen.

Allgemeiner geht es aber bei Turingmaschinen darum, die intuitiven Begriffe „berechenbare Funktion“ und „entscheidbares Problem“ zu formalisieren.

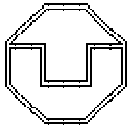
Nachweis der Berechenbarkeit:

Um für eine (eventuell partielle) Funktion

$$\text{bzw.} \quad \begin{array}{l} f : \mathbb{N}^k \to \mathbb{N} \\ f : (\Sigma^*)^k \to \Sigma^* \end{array}$$

intuitiv klarzumachen, daß sie berechenbar ist, genügt es, ein Berechnungsverfahren (einen Algorithmus) für die Funktion anzugeben

(z.B. in Form eines Pascal-, Java- oder C-Programms oder einer abstrakten Beschreibung der Vorgehensweise bei der Berechnung).



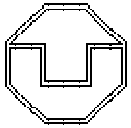
Nachweis der Entscheidbarkeit:

Entsprechend haben wir die Entscheidbarkeit von Problemen wie

- Wortproblem,
- Leerheitsproblem,
- Äquivalenzproblem

dadurch begründet, daß wir beschrieben haben, wie man die Probleme mit Hilfe eines Rechenverfahrens (d.h. effektiv) entscheiden kann.

Aus dieser Beschreibung hätte man jeweils leicht ein Pascal-, etc. Programm zur Entscheidung des Problems konstruieren können.



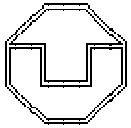
Nachweis der Nicht-Entscheidbarkeit/Nicht-Berechenbarkeit:

Intuitives Vorgehen nicht möglich, da man formal nachweisen muß, daß es kein Berechnungsverfahren für die Funktion oder das Entscheidungsproblem geben kann.

Beweis benötigt formale Definition eines Berechnungsmodells, das

- einfach ist, damit formale Beweise erleichtert werden (z.B. nicht Programmiersprache ADA),
- berechnungsuniversell ist, d.h. alle intuitiv berechenbaren Funktionen damit berechnet werden können (z.B. nicht nur endliche Automaten).

Turingmaschinen liefern so ein Modell!



Definition (Turing-berechenbar)

Die (partielle) Funktion

$$f : (\Sigma^*)^n \to \Sigma^*$$

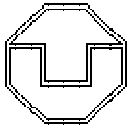
heißt Turing-berechenbar, falls es eine DTM $\mathcal{A}$ gibt, für die gilt:

- der Definitionsbereich $\mathrm{dom}(f)$ von f besteht aus genau den Tupeln $(x_1, \ldots, x_n) \in (\Sigma^*)^n$ mit

  $$\not{b}\, q_0 x_1 \not{b} \ldots \not{b}\, x_n \not{b} \vdash_{\mathcal{A}}^* k$$

  für eine Stoppkonfiguration k;

- im Fall $(x_1, \ldots, x_n) \in \mathrm{dom}(f)$ ist die erreichte Stoppkonfiguration k von der Form $k = uqyv$ mit
  - $y = f(x_1, \ldots, x_n)$ und
  - $v \in (\Gamma \setminus \Sigma) \cdot \Gamma^* \cup \{\varepsilon\}$.

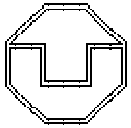


Beachte:

1. Undefiniertheit des Funktionswertes von f entspricht der Tatsache, daß die Maschine bei dieser Eingabe nicht terminiert (da keine Stoppkonfiguration erreicht wird).

2. Bei berechenbaren Funktionen betrachten wir nur deterministische Maschinen, da sonst der Funktionswert nicht eindeutig sein müßte.

3. Bei $|\Sigma| = 1$ kann man Funktionen von $(\Sigma^*)^n \to \Sigma^*$ als Funktionen von $\mathbb{N}^n \to \mathbb{N}$ auffassen (a^k entspricht k).



Beispiel

Die Funktion

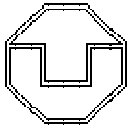
$$f : \mathbb{N} \to \mathbb{N} : \quad n \mapsto 2n$$

ist Turing-berechenbar.

Wie kann eine Turingmaschine die Anzahl der as auf dem Band verdoppeln?

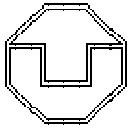
Idee:

- Ersetze das erste a durch b,
- laufe nach rechts bis zum ersten blank und ersetze dieses durch c
- laufe zurück bis zum zweiten a (unmittelbar rechts vom b) und ersetze dieses durch b,
- laufe nach rechts bis zum ersten blank etc.
- Sind alle as aufgebraucht, so ersetze noch die bs und cs wieder durch as.



Dies wird durch die folgende Übergangstafel realisiert:

q_0	$\not b$	$\not b$	n	stop	$2 \cdot 0 = 0$
q_0	a	b	r	q_1	ersetze a durch b $\quad (\star)$
q_1	a	a	r	q_1	laufe nach rechts über as
q_1	c	c	r	q_1	und bereits geschriebene cs
q_1	$\not b$	c	n	q_2	schreibe weiteres c
q_2	c	c	l	q_2	laufe zurück über cs und
q_2	a	a	l	q_2	as
q_2	b	b	r	q_0	bei erstem b eins nach rechts und weiter wie $(\star)$ oder
q_0	c	c	r	q_3	alle as bereits ersetzt
q_3	c	c	r	q_3	laufe nach rechts bis Ende der cs
q_3	$\not b$	$\not b$	l	q_4	letztes c erreicht
q_4	c	a	l	q_4	ersetze cs und bs
q_4	b	a	l	q_4	durch as
q_4	$\not b$	$\not b$	r	stop	bleibe am Anfang der erzeugten $2n$ as stehen



Dresden

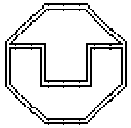
Turingmaschinen sind nur ein solches Berechnungsmodell. Es wurden in der Informatik noch eine Vielzahl anderer Modelle eingeführt, z.B.:

- μ-rekursive Funktionen
- WHILE-Programme
- GOTO-Programme
- URMs (unbeschränkte Registermaschinen),
- Programmiersprachen wie Pascal

Es hat sich herausgestellt, daß all diese Modelle äquivalent sind, d.h. die gleiche Klasse von Funktionen berechnen.

Außerdem ist es bisher nicht gelungen, ein formales Berechnungsmodell zu finden, so daß

- die dadurch berechneten Funktionen noch intuitiv berechenbar sind,
- dadurch Funktionen berechnet werden, die nicht in den oben genannten Modellen ebenfalls berechenbar sind.



Churchsche These:

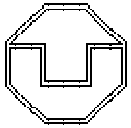
> Die (intuitiv) berechenbaren Funktionen sind genau die mit Turingmaschinen (und damit mit GOTO-, WHILE-, Pascal-Programmen, URMs, . . .) berechenbaren Funktionen.

Man spricht hier von einer These und nicht von einem Satz, da es nicht möglich ist, diese Aussage formal zu beweisen:

der intuitive Berechenbarkeitsbegriff ist ja nicht formal definierbar,
man kann darüber also keine formalen Beweise führen.

Es gibt aber sehr viele Indizien für die Richtigkeit der These:

Vielzahl äquivalenter Berechnungsmodelle.



Gemäß der These von Church werden die Begriffe Turing-berechenbar und (intuitiv) berechenbar als synonym verwenden.

Definition (Berechenbarkeit)

Wir betrachten partielle oder totale Funktionen

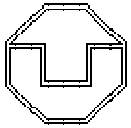
$$f : (\Sigma^*)^n \to \Sigma^*.$$

Die Funktion f heißt berechenbar (partiell rekursiv), falls sie Turing-berechenbar ist.

Ist f total (d.h. $\mathrm{dom}(f) = (\Sigma^*)^n$), so heißt f rekursiv.

Beachte:

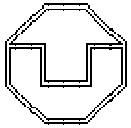
Bei partiellen Funktionen entspricht Undefiniertheit des Funktionswertes der Tatsache, daß das Berechnungsverfahren (die DTM) bei dieser Eingabe nicht terminiert.



Wegen der erwähnten Äquivalenz von Berechnungsmodellen treffen alle Definitionen und Resultate, die für Turingmaschinen formuliert sind, auch auf die anderen Modelle zu.

Die Existenz eines Berechnungsverfahrens (einer DTM) wird meist intuitiv begründen.

Diese intuitiven Argumente können aber im Prinzip leicht (wenn auch im Detail zeitaufwendig) in tatsächliche TM-Konstruktionen übersetzt werden.



Definition (entscheidbar)

Wir betrachten n-stellige Relationen $R \subseteq (\Sigma^*)^n$.

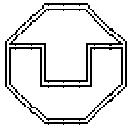
R heißt entscheidbar (rekursiv), falls ihre charakteristische Funktion

$$\chi_R : (\Sigma^*)^n \rightarrow \Sigma^* \quad \text{mit}$$

$$(x_1, \ldots, x_n) \mapsto \begin{cases} a & \text{falls } (x_1, \ldots, x_n) \in R \\ \varepsilon & \text{sonst} \end{cases}$$

berechenbar ist.

Dabei ist a ein Element von Σ (beliebig aber fest gewählt).



Bemerkung

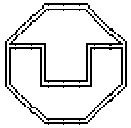
Probleme wie z.B. das Wortproblem für kontextfreie Sprachen können als Relationen aufgefaßt werden:

Es sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik und $w \in \Sigma^*$.

Die Grammatik G kann als Wort $\mathrm{code}(G) \in \Gamma^*$ über einem erweiterten Alphabet Γ aufgefaßt werden.

Das Wortproblem für G entspricht der Relation

$$R = \{(\mathrm{code}(G), w) \mid w \in L(G)\} \subseteq \Gamma^* \times \Gamma^*.$$



Nicht-berechenbare Funktionen / nicht-entscheidbare Probleme

Die Existenz nicht-entscheidbarer Probleme und nicht-berechenbarer Funktionen ergibt sich aus einem einfachen Abzählargument:

- es gibt nur abzählbar viele Turingmaschinen (bis auf Alphabet- und Zustandsumbenennung)
- es gibt überabzählbar viele Funktionen $f : (\Sigma^*)^k \to \Sigma^*$ und überabzählbar viele Relationen $R \subseteq (\Sigma^*)^k$.

Man kann aber auch zeigen, daß viele „natürlichen“ Probleme unentscheidbar sind, z.B.:

- Das Äquivalenzproblem für kontextfreie Grammatiken.
- Das Wortproblem und das Leerheitsproblem für Typ-0-Sprachen.
- Das Halteproblem für Turingmaschinen: hält die Maschine für eine gegebene Eingabe nach endlich vielen Schritten an?