Theoretische Informatik für Wirtschaftsinformatik und Lehramt

Turing-Maschinen und Berechenbarkeit

Priv.-Doz. Dr. Stefan Milius
stefan.milius@fau.de

Theoretische Informatik
Friedrich-Alexander Universität Erlangen-Nürnberg

SS 2016

Gliederung

Worum geht es in diesem Abschnitt? (I)

Grundlegende Fragen der theoretischen Informatik

- Was ist durch Modellierung, Formalisierung und Algorithmisierung im Computer *prinzipiell* möglich?
- Wo liegen die Grenzen des Computereinsatzes?

  Warum sind mache Aufgaben auch mit den leistungsstärksten Computern überhaupt nicht (Berechenbarkeit) oder nicht effizient (Komplexität) lösbar?

Bisher: einfache Formalismen zur Beschreibung von regulären und kontextfreien Sprachen:

- (nicht-)deterministische endliche/Keller- Automaten,
- reguläre und kontextfreie Grammatiken

Worum geht es in diesem Abschnitt? (II)

- Zur Beantwortung der grundlegenden Fragen nötig: möglichst leistungsstarkes Algorithmenmodell.
- Turing-Maschinen sind so ein Modell:

  man kann nachweisen, dass Turing-Maschinen genauso leistungsstark wie typische Computersysteme sind
- Turing-Maschinen sind Automaten, die:
  - über (potentiell) unendlich langes, in einzelne Felder unterteiltes, *Speicherband* verfügen,
  - mittels eines über ein *„festes Programm“ gesteuerten Schreib-Lese-Kopfes* . . .
  - Zeichen auf der aktuellen Kopf-Position lesen/schreiben und den Schreib-Lese-Kopf feldweise weiterbewegen können.

Worum geht es in diesem Abschnitt? (III)

- weiteres Modell:

Chomsky-Grammatiken (vom Typ 0) als *regelbasierte* Systeme.

Ermöglichen Erzeugung (**generative Beschreibung**) von formalen Sprachen.

Typ-0 Grammatiken sind zu Turing-Maschinen äquivalent

Lernziele

1. Begriff der intuitiven Berechenbarkeit erklären und argumentieren, dass nicht jede Funktion berechenbar ist.
2. das Konzept der Turing-Maschinen und Turing-berechenbare Funktionen erklären und für eine gegebene Problemstellung eine Turing-Maschine konstruieren können
3. Chomsky-Grammatiken erklären können,

   Zusammenhang zwischen Chomsky-Grammatiken und Turing-Maschinen kennen

Intuitive Berechenbarkeit (I)

Welche intuitive Bedeutung hat Berechenbarkeit?

- Eine (evtl. partielle) Funktion $f : \mathbb{N}^k \to \mathbb{N}$ soll als berechenbar angesehen werden, falls es einen Algorithmus gibt (z.B. ein funktionales Haskell-Programm), das f berechnet, d.h. einen Algorithmus für das (Berechnungs-)Problem:

  Eingabe: $(n_1, \ldots, n_k) \in \mathbb{N}^k$

  (z.B. $n_1, \ldots, n_k$ als Startwerte der Variablen $x_1, \ldots, x_k$)

  Aufgabe: Ausgabe von $f(n_1, \ldots, n_k)$

  (nach endlich vielen Schritten).

- An Stellen, an denen f undefiniert ist, soll der Algorithmus **nicht** stoppen (Endlosschleife).

Intuitive Berechenbarkeit (II)

Beispiel 1

Der Algorithmus

```
input(n);
while(true){};
```

„berechnet“ die überall undefinierte Funktion $\Omega : \mathbb{N} \to \mathbb{N}$

Intuitive Berechenbarkeit (III)

Beispiel 2

Die Funktion

$$f(n) = \begin{cases} 1 & \text{falls } n \text{ irgendwo in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

ist möglicherweise nicht berechenbar.

Ist die Ziffernfolge von π echt zufällig, so ist allerdings die Wahrscheinlichkeit, dass jede Ziffernfolge irgendwann mal vorkommt, gleich 1.

Dann wäre

$$f(n) = 1 \quad \text{für alle } n \in \mathbb{N}$$

und f damit sicher berechenbar.

Intuitive Berechenbarkeit (IV)

Beispiel 3

Sei $r \in (0,1)$ irrational.

Betrachte die Funktionenschar $f_r : \mathbb{N} \to \mathbb{N}$:

$$f_r(n) = \begin{cases} 1 & \text{falls } n \text{ Anfangsstück des Nachkommateils von r ist} \\ 0 & \text{sonst} \end{cases}$$

Die Funktionen f_r sind paarweise verschieden, also ist die Menge der f_r *überabzählbar*.

Die Menge aller durch ein Computerprogramm berechenbaren Funktionen $f : \mathbb{N} \to \mathbb{N}$ ist aber *abzählbar*
(da jedes Programm durch endlichen Text niedergeschrieben werden kann).

Daraus folgt die Existenz nicht berechenbarer Funktionen f_r.

Bearbeitungsmodelle

- Fragestellungen:

  algorithmische (systematische, automatische, ...)
  „Bearbeitung“ von Wörtern (und Sprachen), genauer:
  - Berechnung von (im Allgemeinen partiellen) Funktionen $f : \Sigma^* \to \Sigma^*$ (für ein Alphabet Σ)
  - Feststellung des Enthaltenseins eines Wortes w in einer Sprache L: $w \stackrel{?}{\in} L$.
  - Beschreibung und/oder Erzeugung von Sprachen
  - Aufwand solcher algorithmischer Verfahren

- Erforderlich: Formale Modelle für intuitive Berechenbarkeit, z. B.
  - Turing-Maschinen
  - Chomsky-Grammatiken
  - LOOP und WHILE-Programme (später)

Turing-Maschinen (TM)

- 1936 vom englischen Mathematiker Alan Turing als ein grundlegendes „mechanisch“, „maschinell“ orientiertes Modell zur „automatischen“ Bearbeitung von Wörtern eingeführt
- **Ziel:** Modell des mathematisch arbeitenden Menschen
- **Besonderheit:** TM kann mit nur drei Operationen (Lesen, Schreiben und Schreib-Lese-Kopf Bewegen) alle Probleme lösen, die auch von einem Computer gelöst werden können
- sämtliche mathematischen Grundfunktionen wie Addition und Multiplikation lassen sich mit diesen drei Operationen implementieren
- komplexe Operationen der üblichen Computerprogramme darauf aufbauend implementierbar

Alan Turing

Begründer der Informatik

2. Weltkrieg: Krypto-Analyse der deutschen Enigma-Maschine

Turing-Test (für künstliche Intelligenz)

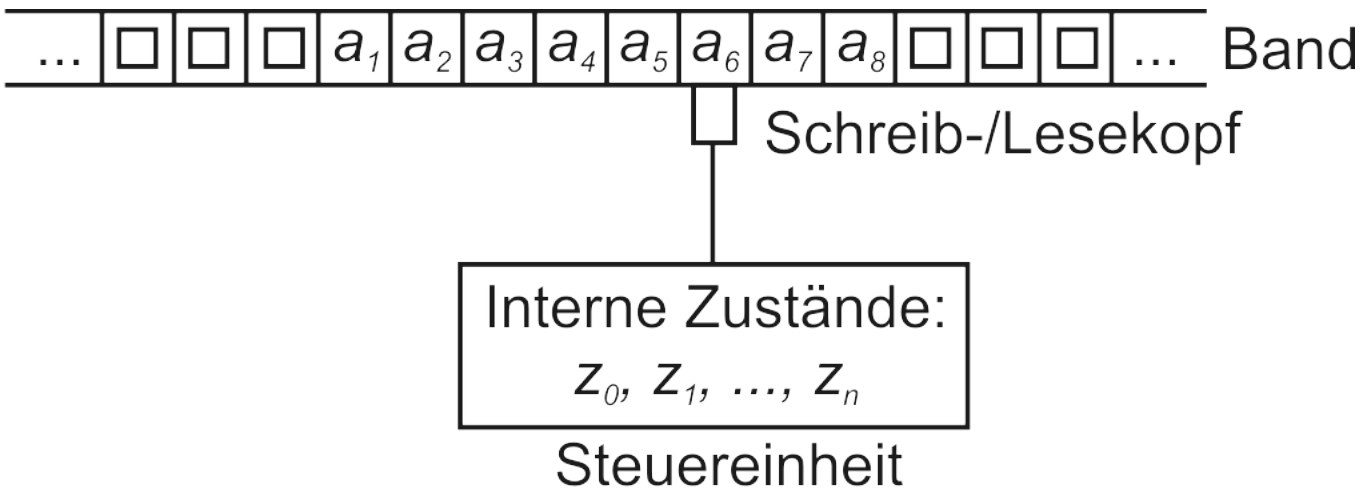
Turing-Award = „Nobelpreis für Informatik“

Alan M. Turing (1912–1954)
Quelle: Wikipedia

Andrew Hodges. *Alan Turing – The Enigma*. Walker & Company, 2000.
Robert Harris. *Enigma*. Heyne, 1996.

Aufbau einer Turing-Maschine (I)

Eine Turing-Maschine ist informell eine Maschine, die aus einer Steuereinheit und einem (beidseitig potentiell unendlichen) Speicherband besteht:



Aufbau einer Turing-Maschine (II)

- Band ist in Felder unterteilt, die jeweils genau ein Zeichen aufnehmen können
- Steuereinheit kann *endlich viele Zustände* annehmen und Schreib-/Lesekopf über die Bandfelder bewegen
  (pro Rechenschritt jeweils ein Feld nach links oder rechts)
- Steuereinheit kann den Inhalt des jeweils aktuellen (Arbeits-) Feldes lesen und/oder überschreiben
- Kennzeichnung von „leeren“ Feldern geschieht mit ausgezeichnetem Leerzeichen (hier durch $\square$ bezeichnet)
- Steuereinheit enthält zur Steuerung der Maschine ein „fest verdrahtetes“ Programm,
  die *Überführungsfunktion*

Formale Definition einer Turing-Maschine

Definition

Eine **(deterministische) Turing-Maschine (TM, DTM)** $M = (Z, \Sigma, \Gamma, \delta, z_0)$ ist gegeben durch:

- eine endliche Menge Z von **Zuständen**
- ein **Eingabealphabet** Σ
- ein **Bandalphabet** $\Gamma \supseteq \Sigma$ mit einem **Leerzeichen** $\square \in \Gamma \setminus \Sigma$
- eine partielle **Überführungsfunktion** $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$
- einen **Startzustand** $z_0 \in Z$

Eine **Konfiguration** K von M ist ein Wort $K \in \Gamma^* Z \Gamma^*$.

Erweiterung von Σ zum Bandalphabet Γ ermöglicht Verwendung weiterer Hilfszeichen, z. B. $\square$

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

Frage: Was bedeutet $\delta(z, a) = (z', b, X)$?

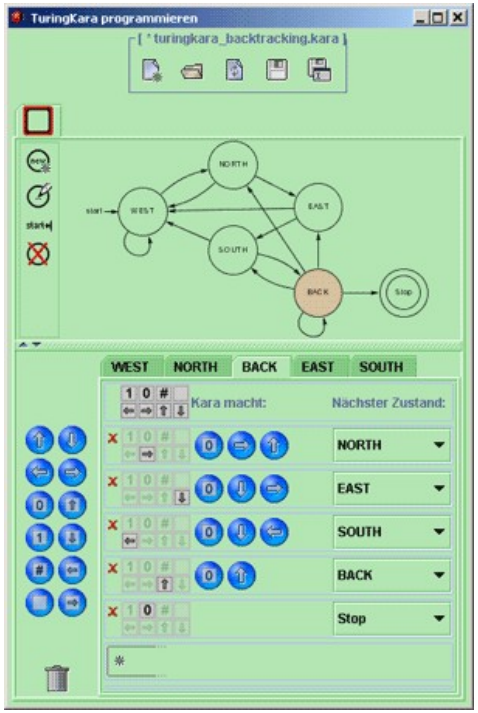
Interpretation:

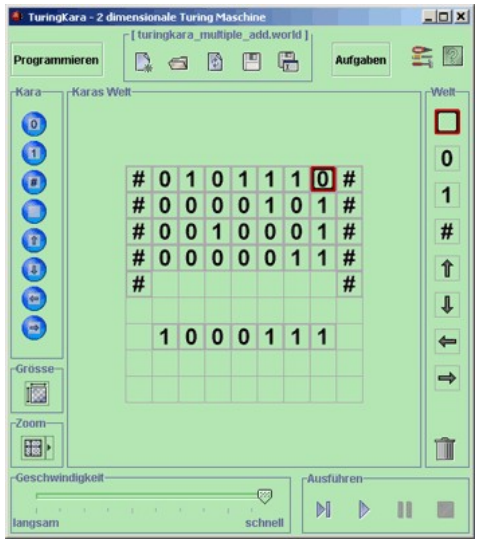
Liest M im Zustand z das Zeichen a auf dem Arbeitsfeld so,

- geht M in den Zustand z' über,
- ersetzt a durch b und
- bewegt den Schreib-/Lesekopf um ein Feld
  nach links (falls $X = L$)
  oder nach rechts ($X = R$)
  oder belässt den Kopf an gleicher Position ($X = N$).

Lernsoftware zu TM (Auswahl)

TuringKara





`http://www.swisseduc.ch/informatik/karatojava/turingkara/index.html`

MathePrisma `http://www.matheprisma.uni-wuppertal.de/Module/Turing/`

Tursi `http://ais.informatik.uni-freiburg.de/tursi/`

Reflexive und transitive Hülle einer Relation (I)

- Für die weiteren Betrachtungen von Turingmaschinen wiederhole *reflexive und transitive Hülle* einer Relation.

- Sei $R \subseteq M \times M$ eine binäre Relation.

  $R^* = \bigcup_{n \geq 0} R^n$ heißt reflexive und transitive Hülle von R.

Beobachtung

R^* ist die kleinste reflexive und transitive Relation die R enthält

Es gilt $x\ R^*\ y$ genau dann wenn $\exists n \geq 0, x_0, x_1, \ldots x_n$:

$$x = x_0\ R\ x_1\ R\ x_2\ R \ldots R\ x_n = y$$

(für $n = 0$, $x = x_0 = y$)

Reflexive und transitive Hülle einer Relation (II)

Beispiel

Sei $R = \{(n, n+1) \mid n \in \mathbb{N}\} \subseteq \mathbb{N} \times \mathbb{N}$, also

$$0\,R\,1, \quad 1\,R\,2, \quad 2\,R\,3, \quad \ldots$$

und R^* ist die Relation $\leq$ auf $\mathbb{N}$.

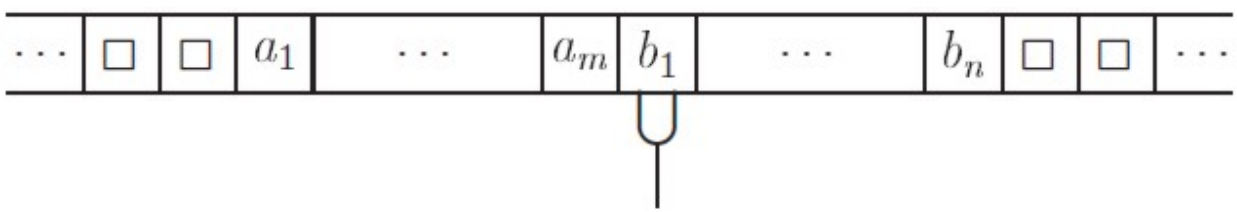
Konfigurationen(I)

Es sei $M = (Z, \Sigma, \Gamma, \delta, z_0)$ eine TM.
Eine Konfiguration

$$K = \alpha z \beta \qquad \text{mit } \alpha = a_1 \cdots a_m \in \Gamma^*,\ \beta = b_1 \cdots b_n \in \Gamma^*,\ z \in Z$$

beschreibt eine „Momentaufnahme“ der Maschine M:

- M befindet sich im Zustand z
- auf den Feldern links vom Schreib-/Lesekopf stehen die Zeichen $a_1, \ldots, a_m$
- auf den Feldern ab dem Kopf nach rechts stehen die Zeichen $b_1, \ldots, b_n$
- links von a_1 und rechts von b_n stehen nur noch Leerzeichen $\square$



Konfigurationen(II)

Man beachte:

- falls $\beta = \varepsilon$, so steht $\square$ auf dem Arbeitsfeld
- alle Konfigurationen der Form

$$\square^n \alpha z \beta \square^m$$

werden miteinander identifiziert (sind ununterscheidbar)

(z.B. sind *azbb*, $\square\square$*azbb*$\square$ und $\square$*azbb* dieselbe Konfiguration)

Wenn eine TM „rechnet“ durchläuft sie eine Folge von Konfigurationen, die durch die Überführungssfunktion δ „programmiert“ ist:

Übergangsrelation (I)

Definition

Sei $M = (Z, \Sigma, \Gamma, \delta, z_0)$ eine TM. Die binäre **Übergangsrelation**

$\vdash_M$ (lies: „ist direkt überführbar in“)

auf der Menge der Konfigurationen von M ist wie folgt definiert:

Ist $\delta(z, b_1) = (z', c, X)$, so gilt

$$a_1 \cdots a_m z b_1 \cdots b_n \vdash_M \begin{cases} a_1 \cdots a_m z' c b_2 \cdots b_n, & \text{falls } X = N \\ a_1 \cdots a_m c z' b_2 \cdots b_n, & \text{falls } X = R \\ a_1 \cdots a_{m-1} z' a'_m c b_2 \cdots b_n, & \text{falls } X = L. \end{cases}$$

Beachte: $b_1 \cdots b_n$ ist hier niemals das leere Wort.

Falls $a_1 \cdots a_m = \varepsilon$, so ist im dritten Fall $a'_m = \square$ sonst $a'_m = a_m$.

Ist $K \vdash_M K'$, so heißt K' **Nachfolgekonfiguration** von K.

Eine Konfiguration heißt **final** (oder **terminierend**), wenn sie keine Nachfolgekonfiguration hat.

Übergangsrelation (II)

Definition (Fortsetzung)

$\vdash_M^*$ (lies: „ist überführbar in“)
ist die reflexive, transitive Hülle von $\vdash_M$.

Das heißt, es gilt $K \vdash_M^* K'$, wenn es Konfigurationen $K_0, \ldots, K_\ell$ ($\ell \in \mathbb{N}$) gibt mit

$$K = K_0 \vdash_M K_1 \vdash_M K_2 \vdash_M \cdots \vdash_M K_\ell = K'.$$

Die Folge der $K_0, \ldots, K_\ell$ heißt **Rechnung** der **Länge** ℓ von M.

Die Gesamtanzahl der verschiedenen Felder des Bandes, auf denen der Schreib-/Lesekopf in $K_0, \ldots, K_\ell$ zu stehen kommt, heißt **Bandverbrauch** der Rechnung.

Wenn der Kontext klar ist, schreiben wir meist

$\vdash$ bzw. $\vdash^*$ statt $\vdash_M$ und $\vdash_M^*$.

Überführungsfunktion und Rechnung einer TM

Beispiel

Sei M_{beisp} eine TM mit

$$Z = \{z_0, z_1, z_2, z_3\}, \quad \Sigma = \{a, b\}, \quad \Gamma = \{a, b, \square\},$$

und der Überführungsfunktion

$$\begin{aligned}
\delta: \quad (z_0, a) &\mapsto (z_0, b, R),\\
(z_0, b) &\mapsto (z_0, a, R),\\
(z_0, \square) &\mapsto (z_1, \square, L),\\
(z_1, a) &\mapsto (z_1, a, L),\\
(z_1, b) &\mapsto (z_1, b, L),\\
(z_1, \square) &\mapsto (z_2, \square, R),\\
(z_2, a) &\mapsto (z_3, a, N).
\end{aligned}$$

Mögliche Rechnung von M_{beisp}:

$$\begin{aligned}
z_0aba &\vdash bz_0ba\\
&\vdash baz_0a\\
&\vdash babz_0\\
&\vdash baz_1b\\
&\vdash bz_1ab\\
&\vdash z_1bab\\
&\vdash z_1\square bab\\
&\vdash z_2bab
\end{aligned}$$

z_2bab ist eine finale Konfiguration.

Zustandstabelle und Zustandsübergangsgraph einer TM

Beispiel

Überführungsfunktion:

$$
\begin{aligned}
\delta: \quad (z_0, a) &\mapsto (z_0, b, R),\\
(z_0, b) &\mapsto (z_0, a, R),\\
(z_0, \square) &\mapsto (z_1, \square, L),\\
(z_1, a) &\mapsto (z_1, a, L),\\
(z_1, b) &\mapsto (z_1, b, L),\\
(z_1, \square) &\mapsto (z_2, \square, R),\\
(z_2, a) &\mapsto (z_3, a, N).
\end{aligned}
$$

Zustandstabelle:

	a	b	$\square$
z_0	(z_0, b, R)	(z_0, a, R)	$(z_1, \square, \mathrm{L})$
z_1	(z_1, a, L)	(z_1, b, L)	$(z_2, \square, \mathrm{R})$
z_2	(z_3, a, N)	–	–
z_3	–	–	–

Zustandstabelle und Zustandsübergangsgraph einer TM

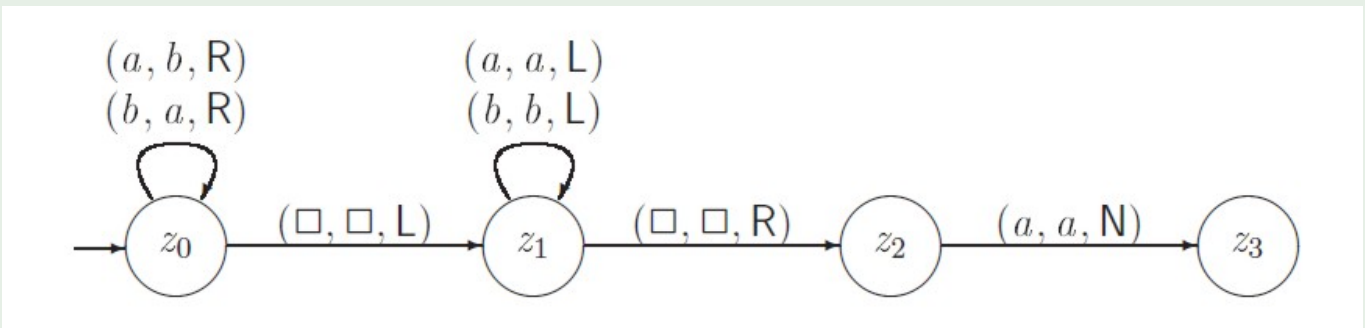
Beispiel

Zustandstabelle:

	a	b	$\square$
z_0	(z_0, b, R)	(z_0, a, R)	$(z_1, \square, \mathsf{L})$
z_1	(z_1, a, L)	(z_1, b, L)	$(z_2, \square, \mathsf{R})$
z_2	(z_3, a, N)	–	–
z_3	–	–	–

Zustandsübergangsgraph/-diagramm:

Für z, z' mit $\delta(z, a) = (z', b, X)$ enthält dieser Graph eine mit (a, b, X) markierte gerichtete Kante von z nach z':



Terminierung von TM (I)

Definition

Sei M eine TM mit Eingabealphabet Σ und Anfangszustand z_0. M **terminiert für (die Eingabe)** $w \in \Sigma^*$, wenn es eine finale Konfiguration K von M gibt mit

$$z_0 w \vdash^* K \qquad \text{(bzw.} \quad z_0 \square \vdash^* K \quad \text{für } w = \varepsilon\text{)}.$$

Verwendung von TM zur Bearbeitung eines Eingabewortes w:

- TM bearbeitet w gemäß „Programm“:
  - beginnt im Anfangszustand z_0
  - außer w steht nichts auf dem Band
  - Schreib-/Lesekopf steht auf dem ersten Zeichen von w (oder $\square$, falls $w = \varepsilon$)
- Bearbeitung terminiert, wenn nach endlich vielen Schritten eine Konfiguration erreicht wird, in der die Rechnung nicht fortsetzbar ist

Terminierung von TM (II)

Beispiel

Die TM M_{beisp} im obigen Beispiel terminiert für jede Eingabe $w \in \{a, b\}^*$, denn sie

- liest das Eingabewort von links nach rechts und ersetzt dabei $a \mapsto b$ und $b \mapsto a$
- bewegt dann den Kopf nach links und
- hält zuletzt auf dem ersten Zeichen (im Zustand z_2 oder z_3)

Berechnungsbegriffe für TM (I)

Fragen: Was berechnen TM?
Welches Problem löst eine gegebene TM?

Definition

Es sei $M = (Z, \Sigma, \Gamma, \delta, z_0)$ eine TM und $f : \Sigma^* \to \Sigma^*$ eine Funktion.

M **berechnet** f, falls gilt:

1. Ist $f(w)$ definiert, so ist

   $z_0 w \vdash^* zv$ für ein $z \in Z$, $\quad v = f(w)$, $\quad$ und zv ist final.
2. Ist $f(w)$ undefiniert, so terminiert M für w nicht.

f heißt **(Turing-) berechenbar,** wenn es eine TM gibt, die f berechnet.

Berechnungsbegriffe für TM (II)

Beispiel

M_{beisp} von oben berechnet die (totale) Funktion

$$f : \{a, b\}^* \to \{a, b\}^* \quad \text{mit} \quad f(s_1 s_2 \cdots s_n) = s_1' s_2' \cdots s_n'$$

wobei

$$s_i' = \begin{cases} b, & \text{falls } s_i = a \\ a, & \text{falls } s_i = b \end{cases} \qquad (i = 1, \ldots, n).$$

„Invertierer“

Rechnen mit TM (I)

Beispiel

Gegeben: $\Sigma = \{a, b\}$

Gesucht: TM M_R, die die Spiegelungs-Funktion auf Σ^* berechnet, für die also für $w \in \Sigma^*$ (und einen Zustand z) gilt:

$z_0 w \vdash^* z w^R$ ($z w^R$ final) also z. B. $z_0 abb \vdash^* zbba$

Lösungsidee:

M_R hat Bandalphabet $\{a, b, \$, \square\}$ und arbeitet wie folgt:

- M_R sucht zunächst das letzte Zeichen von w.
- Ist kein solches Zeichen vorhanden (d. h. $w = \varepsilon$), so ist nichts mehr zu tun, andernfalls geht M_R dann nach links zum vorletzten Zeichen.
- Ist dieses nicht vorhanden (d. h. $|w| = 1$), so ist nichts mehr zu tun, andernfalls wird das vorletzte Zeichen durch das Hilfszeichen \$ ersetzt und, nachdem M_R wieder nach rechts bis zum (ersten) Leerzeichen gegangen ist, dort geschrieben.

Rechnen mit TM (III)

Beispiel (Fortsetzung) – Arbeitsweise von M_R

- Um welches Zeichen (a oder b) es sich dabei handelt, merkt M_R sich in zwei verschiedenen Zuständen z_a und z_b.
- Anschließend geht M_R wieder nach links und sucht (links von \$) das vorvorletzte Zeichen von w. Dieses wird durch \$ ersetzt und nach rechts auf das erste leere Feld „transportiert“:
  $z_0abb \vdash^* abbz_0 \vdash abz_1b \vdash az_2bb \vdash a\$z_bb \vdash a\$bz_b \vdash a\$bz_3b$
- Dieser Vorgang wird so lange wiederholt, bis das erste Zeichen von w nach rechts gebracht worden ist:
  $az_3\$bb \vdash z_4a\$bb \vdash z_2a\$bb \vdash \$z_a\$bb \vdash^* \$\$bbz_a \vdash \$\$bbz_3a$
- Anschließend werden noch alle \$ gelöscht (d. h. durch $\square$ ersetzt) und der Schreib-/Lesekopf an die richtige Stelle gebracht:
  z_5bba

Rechnen mit TM (II)

Beispiel (Fortsetzung)

M_R hat Zustände mit folgender intuitiver Bedeutung:

- z_0 Startzustand, letztes Zeichen von w suchen
- z_1 zum vorletzten Zeichen gehen
- z_2 nächstes Zeichen gefunden (falls vorhanden, sonst: Übergang zu z_5)
- z_a a nach rechts transportieren
- z_b b nach rechts transportieren
- z_3 Transport eines Zeichens beendet;
  nach links das am weitesten rechts stehende \$ suchen
- z_4 nach links das als nächste zu transportierende Zeichen suchen
- z_5 nichts mehr zu transportieren, alle (evtl. vorhandenen) \$ löschen

Rechnen mit TM (IV)

Beispiel (Fortsetzung)

Zustandstabelle

	a	b	\$	$\square$
z_0	(z_0, a, R)	(z_0, b, R)	–	$(z_1, \square, \mathsf{L})$
z_1	(z_2, a, L)	(z_2, b, L)	–	–
z_2	$(z_a, \$, \mathsf{R})$	$(z_b, \$, \mathsf{R})$	–	$(z_5, \square, \mathsf{R})$
z_a	(z_a, a, R)	(z_a, b, R)	$(z_a, \$, \mathsf{R})$	(z_3, a, N)
z_b	(z_b, a, R)	(z_b, b, R)	$(z_b, \$, \mathsf{R})$	(z_3, b, N)
z_3	(z_3, a, L)	(z_3, b, L)	$(z_4, \$, \mathsf{L})$	–
z_4	(z_2, a, N)	(z_2, b, N)	$(z_4, \$, \mathsf{L})$	$(z_5, \square, \mathsf{R})$
z_5	–	–	$(z_5, \square, \mathsf{R})$	–

Die Einträge für $(z_1, \square)$, (z_5, a) und (z_5, b) verursachen finale Konfigurationen. Die übrigen leeren Zellen betreffen Konfigurationen, die nicht vorkommen können.

Rechnen mit TM (V)

Beispiel (Fortsetzung)

Beispielrechnungen bis zu finalen Konfigurationen

1. $w = \varepsilon$:
   $z_0\Box \vdash z_1\Box$
2. $w = a$:
   $z_0 a \vdash a z_0 \vdash z_1 a \vdash z_2\Box a \vdash z_5 a$
3. $w = abb$:

$z_0 abb \vdash a z_0 bb \vdash ab z_0 b \vdash abb z_0$	ans Ende laufen
$\vdash ab z_1 b \vdash a z_2 bb$	vorletztes Zeichen suchen
$\vdash a\$ z_b b \vdash a\$ b z_b \vdash a\$ b z_3 b$	nach rechts transportieren
$\vdash a\$ z_3 bb \vdash a z_3 \$ bb \vdash z_4 a\$ bb \vdash z_2 a\$ bb$	nächstes Zeichen suchen
$\vdash \$ z_a \$ bb \vdash \$\$ z_a bb \vdash \$\$ b z_a b \vdash \$\$ bb z_a \vdash \$\$ bb z_3 a$	nach rechts
$\vdash \$\$ b z_3 ba \vdash \$\$ z_3 bba \vdash \$ z_3 \$ bba \vdash z_4 \$\$ bba \vdash z_4\Box\$\$ bba$	
$\vdash z_5 \$\$ bba \vdash z_5 \$ bba \vdash z_5 bba$	\$ löschen

Rechnen mit TM (VI)

Wie misst man den Aufwand einer Berechnung?

Messgrößen von Algorithmen: Rechenzeit und/oder Speicherbedarf

Messgrößen bei Turing-Maschinen:

- Rechenzeit: Länge der Rechnung (= Anzahl der Konfigurationen)
- Speicherbedarf: Bandverbrauch der Rechnung

(später mehr $\rightsquigarrow$ Komplexitätstheorie)

Modellierung arithmetischer Funktionen mit TM (I)

Arithmetische Funktionen sind (mehrstellige) Funktionen auf natürlichen Zahlen

$$f : \mathbb{N}^k \to \mathbb{N}$$

(Beschränkung ist nicht wesentlich, da andere Daten durch natürliche Zahlen kodiert werden können.)

Beispiele

- $f_1(x, y) = x + y$
- $f_2(x, y) =$„größter gemeinsamer Teiler von x und y“
- $f_3(x, y, z) = \begin{cases} 1 & \text{falls } z = x * y \\ 0 & \text{sonst} \end{cases}$
- $f_4(x) = 2^x$

Modellierung arithmetischer Funktionen mit TM (II)

- Funktionen wie f_3, d. h. von der speziellen Art

$$f : \mathbb{N}^k \to \{0, 1\}$$

  heißen auch **arithmetische Prädikate**.
- arithmetische Funktionen sind (Turing-)berechenbar, wenn sie – nach geeigneter Kodierung – durch eine TM berechnet werden
- Repräsentation natürlicher Zahlen z. B. mittels Strichdarstellung:
  - Zahl $n \in \mathbb{N}$ wird durch Wort $|^n = \underbrace{|||\cdots|}_{n \text{ Zeichen}}$ repräsentiert
  - $\circ$ als Trenner der Komponenten eines k-Tupels
  - **Vorteil:** Einfachheit der Berechnungen
  - **Nachteil:** hoher Bandverbrauch

Modellierung arithmetischer Funktionen mit TM (III)

Definition

Sei $f : \mathbb{N}^k \to \mathbb{N}$ eine arithmetische Funktion.

Eine TM $M = (Z, \{|, \circ\}, \Gamma, \delta, z_0)$ **berechnet** f, falls gilt:

1. Ist $f(n_1, \ldots, n_k)$ definiert, so ist

$$z_0|^{n_1} \circ |^{n_2} \circ \cdots \circ |^{n_k} \vdash_M^* z|^m$$

   für ein $z \in Z$ und $m = f(n_1, \ldots, n_k)$ und $z|^m$ ist final.
2. Ist $f(n_1, \ldots, n_k)$ undefiniert, so terminiert M für $|^{n_1} \circ |^{n_2} \circ \cdots \circ |^{n_k}$ nicht.

f heißt **(Turing-)berechenbar**, wenn es eine TM gibt, die f berechnet.

Modellierung arithmetischer Funktionen mit TM (IV)

Beispiel

Die Funktion $f_1(x, y) = x + y$ wird berechnet durch die TM mit der Zustandstabelle

	$\mid$	$\circ$	$\square$
z_0	$(z_0, \mid, \mathsf{R})$	$(z_1, \mid, \mathsf{L})$	–
z_1	$(z_1, \mid, \mathsf{L})$	–	$(z_2, \square, \mathsf{R})$
z_2	$(z_3, \square, \mathsf{R})$	–	–
z_3	–	–	–

Algorithmische Idee:

- nach rechts laufen und $\circ$ suchen
- $\circ$ durch $\mid$ ersetzen
- nach links laufen und ersten $\mid$ durch $\square$ überschreiben

Modellierung arithmetischer Funktionen mit TM (V)

Beispiel (Fortsetzung)

$$z_0 \overbrace{|\ldots|}^{n} \circ \overbrace{|\ldots|}^{m} \vdash^* \overbrace{|\ldots|}^{n} z_0 \circ \overbrace{|\ldots|}^{m}$$

$$\vdash \overbrace{|\ldots|}^{n-1} z_1 || \overbrace{|\ldots|}^{m}$$

$$\vdash^* z_1 \square \overbrace{|\ldots|}^{n+m+1}$$

$$\vdash z_2 \overbrace{|\ldots|}^{n+m+1}$$

$$\vdash \square z_3 \overbrace{|\ldots|}^{n+m}$$

Berechnungsbegriffe für TM (III)

Der Akzeptanzbegriff dient der Beschreibung einer Sprache über den „Test“, ob ein Wort in der Sprache enthalten ist

Definition

Eine TM M enthalte (zusätzlich) eine ausgezeichnete Teilmenge $E \subseteq Z$ von **akzeptierenden** Zuständen (**Endzuständen**):

$$M = (Z, \Sigma, \Gamma, \delta, z_0, E) \qquad (M \text{ heißt dann auch } \textbf{Akzeptor}).$$

M **akzeptiert** ein Wort $w \in \Sigma^*$, falls es eine finale Konfiguration $\alpha z_+ \beta$ mit $z_+ \in E$ $(\alpha, \beta \in \Gamma^*)$ gibt mit

$$z_0 w \vdash^* \alpha z_+ \beta.$$

Die betreffende Rechnung heißt **akzeptierend**.

Die Sprache

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

heißt die von M **akzeptierte Sprache**.

Eine Sprache $L \subseteq \Sigma^*$ heißt **semi-entscheidbar (rekursiv aufzählbar)**, wenn es eine TM M gibt mit $L = L(M)$.

Berechnungsbegriffe für TM (IV)

Beachte:

- Im Falle der Akzeptanz wird *kein* Ausgabewort der TM betrachtet.
- Nichtakzeptanz eines Eingabewortes w durch M erfolgt wenn
  - entweder M auf Eingabe w in einer Konfiguration $\alpha z \beta$ mit $z \notin E$ terminiert
  - oder M auf Eingabe von w *nicht* terminiert

Berechnungsbegriffe für TM (V)

Definition

Sei $L \subseteq \Sigma^*$ eine Sprache.

Die TM M **entscheidet** L, wenn
$L = L(M)$ gilt und M für alle $w \in \Sigma^*$ terminiert.

L heißt **entscheidbar (rekursiv)**, wenn es eine TM gibt, die L entscheidet.
Andernfalls heißt L **unentscheidbar**.

- **Anschaulich:** Ein Entscheidungsverfahren für ein Ja-/Nein-Problem ist ein Algorithmus, der für alle Eingaben terminiert und die korrekte Antwort „ja“ bzw. „nein“ liefert.
- **Hier:** Eine Sprache $L \subseteq \Sigma^*$ heißt entscheidbar, wenn es eine TM gibt, die für jedes $w \in \Sigma^*$ mit der korrekten Antwort auf die Frage „Gilt $w \in L$?“ terminiert.

Entscheidbarkeit am Beispiel

Beispiel

Sei $L_b = \{w \in \Sigma^* \mid w = bw' \text{ mit } w' \in \Sigma^*\}$.

(alle Wörter, die mit „b“ beginnen)

DTM für L_b den Zuständen z_0 und z_+ (akzeptierend) und der Zustandstafel:

	a	b	$\square$
z_0	–	(z_+, b, N)	–
z_+	–	–	–

Diese DTM akzeptiert und entscheidet L_b.

Charakteristische Funktion einer Sprache

Für ein Alphabet Σ sei $a_+ \in \Sigma$ ein ausgezeichnetes **(Positiv-)** Zeichen.

Ist $L \subseteq \Sigma^*$ eine Sprache, so ist die **charakteristische Funktion**

$$\chi_L : \Sigma^* \to \Sigma^*$$

von L definiert durch

$$\chi_L(w) = \begin{cases} a_+, & \text{falls } w \in L \\ \varepsilon & \text{sonst.} \end{cases}$$

Außerdem sei $\chi'_L : \Sigma^* \to \Sigma^*$ definiert durch

$$\chi'_L(w) = \begin{cases} a_+, & \text{falls } w \in L \\ \text{undefiniert} & \text{sonst.} \end{cases}$$

Zusammenhänge (I)

Satz 6.1

Sei L eine Sprache.

a) L ist genau dann entscheidbar, wenn χ_L berechenbar ist.

b) L ist genau dann semi-entscheidbar, wenn χ'_L berechenbar ist.

Satz 6.2

Eine Sprache L ist genau dann entscheidbar, wenn L und $\overline{L}$ semi-entscheidbar sind.

Daraus folgt natürlich:

Jede entscheidbare Sprache ist auch semi-entscheidbar.

Zusammenhänge (II)

Satz 6.3

Sei Σ ein Alphabet.

Es gibt Funktionen $f : \Sigma^* \to \Sigma^*$, die nicht berechenbar und Sprachen $L \subseteq \Sigma^*$, die nicht semi-entscheidbar (und damit auch nicht entscheidbar) sind.

Denn: es gibt nur *abzählbar* unendliche viele TM, aber *überabzählbar* viele $f : \Sigma^* \to \Sigma^*$.

Satz 6.4

Jeder reguläre Sprache ist entscheidbar.

Denn: jeder DFA ist eine spezielle TM.

Mehrband-Turing-Maschinen (Mehrband-TM)

Varianten von Turing-Maschinen:

- TM mit nur einseitig unendlichem Band
- TM mit mehr als einem Band **(Mehrband-TM)**

Eine **2-Band-TM** besitzt zwei Bänder.
Die Überführungsfunktion ist vom Typ

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

wobei $\delta(z, a_1, a_2) = (z', b_1, b_2, X_1, X_2)$ informell bedeutet:

falls die Maschine im Zustand z auf dem ersten Band a_1 und auf dem zweiten Band a_2 liest,

dann werden diese Zeichen durch b_1 bzw. b_2 ersetzt, die (unabhängigen) Schreib-/Leseköpfe gemäß X_1 bzw. X_2 bewegt, und die Maschine geht in den Zustand z' über.

Alle diese TM-Modelle sind bezüglich ihrer Verarbeitungs-„Mächtigkeit“ gleich.

Nichtdeterministische Turing-Maschinen (NTM)

- **Bisher:** TM sind deterministisch:
  gemäß der Überführungsfunktion δ gibt es zu jeder Konfiguration höchstens eine Nachfolgekonfiguration
- **Wichtige TM-Variante:** nichtdeterministische Turing-Maschinen (NTM) bei denen eine Konfiguration mehr als eine Nachfolgekonfiguration haben kann
- bei NTM bestimmt $\delta(z, a)$ eine Menge von Tripeln (z', b, X)

Definition

Eine **nichtdeterministische Turing-Maschine (NTM)** ist definiert wie eine DTM mit dem Unterschied, dass δ eine totale Abbildung

$$\delta : Z \times \Gamma \to \wp(Z \times \Gamma \times \{L, R, N\})$$

ist. (Erinnerung: $\wp(A)$ ist die Potenzmenge der Menge A.)

Terminierung von NTM

- weitere TM-Begriffe für NTM definiert wie für DTM, z. B.

  $a_1 \cdots a_m z b_1 \cdots b_n \vdash a_1 \cdots a_m z' c b_2 \cdots b_n, \quad$ falls $(z', c, N) \in \delta(z, b_1)$

- auch Akzeptanz-/Entscheidungsbegriffe für NTM (mit akzeptierenden Zuständen) definiert
- zunächst muss aber der Terminierungsbegriff angepasst werden

Definition

Sei M eine NTM mit Eingabealphabet Σ und Anfangszustand z_0. M **terminiert für (die Eingabe)** $w \in \Sigma^*$, wenn es keine unendliche Folge $K_0, K_1, K_2, \ldots$ von Konfigurationen von M gibt mit $\quad K_0 = z_0 w \quad$ und $\quad K_0 \vdash K_1 \vdash K_2 \vdash \cdots$

Akzeptanz und Entscheidungsbegriff bei NTM

Definition

Es sei $M = (Z, \Sigma, \Gamma, \delta, z_0, E)$ eine NTM (inkl. Endzust. in E).

M **akzeptiert** ein Wort $w \in \Sigma^*$, falls es eine finale Konfiguration $\alpha z_+ \beta$ mit $z_+ \in E$ $(\alpha, \beta \in \Gamma^*)$ gibt mit

$$z_0 w \vdash^* \alpha z_+ \beta.$$

Die Menge

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

heißt die **von M akzeptierte Sprache**.

Sei $L \subseteq \Sigma^*$ eine Sprache.

M **entscheidet** L, wenn M für alle $w \in \Sigma^*$ terminiert und $L = L(M)$ ist.

Arbeitsweise einer NTM

- Wenn M im Zustand z das Zeichen a liest,

  so geht M in eine der möglichen Nachfolgekonfigurationen über (durch eines der Elemente von $\delta(z, a)$ bestimmt),

  oder M hält an, wenn keine Nachfolgekonfiguration existiert (d. h. $\delta(z, a) = \emptyset$ leer ist).

- **Annahme:** Zur Auflösung des Nichtdeterminismus existiert ein „Orakel“, das den richtigen Weg kennt und die passende nächste Konfiguration auswählt.

- **Hinweis:** Nichtdeterminismus ist ein gedankliches Konstrukt; reale Rechner arbeiten deterministisch.

Nichtdeterministische Turing-Maschine (I)

Beispiel

Die NTM M_{beisp} mit akzeptierendem Zustand z_+ und der Zustandstafel

	a	b	$\square$
z_0	(z_0, a, R) (z_1, a, R)	(z_0, b, R)	–
z_1	–	(z_2, b, R)	–
z_2	(z_+, a, N)	–	–
z_+	–	–	–

akzeptiert und entscheidet die Sprache

$$L = \left\{ w \in \{a, b\}^* \mid aba \text{ ist Teilwort von } w \right\}$$

Nichtdeterministische Turing-Maschine (II)

Beispiel (Fortsetzung)

- Akzeptierende Rechnung für die Eingabe $w = baabab$:

$$\begin{aligned} z_0 baabab &\vdash bz_0 aabab \\ &\vdash baz_0 abab \\ &\vdash baaz_1 bab \\ &\vdash baabz_2 ab \\ &\vdash baabz_+ ab \end{aligned}$$

- Nicht akzeptierende Rechnung für die Eingabe $w = baabab$:

$$z_0 baabab \vdash bz_0 aabab \vdash baz_1 abab$$

Grundsätzliche Gleichmächtigkeit von NTM und DTM (I)

- jede DTM ist auch eine NTM
- NTM definieren dieselben Sprachen wie DTM

Satz 6.4

Zu jeder NTM M gibt es eine DTM M' mit $L(M') = L(M)$.

Trotz dieser grundsätzlichen Gleichmächtigkeit können nichtdeterministische Berechnungen aber „einfacher“ sein als entsprechende deterministische.

Grundsätzliche Gleichmächtigkeit von NTM und DTM (II)

Beispiel: Arbeitsweise DTM zu NTM M_{beisp}

- würde Eingabezeichenreihe von links nach rechts durchlaufen
- solange Teilwort *aba* nicht gefunden, versuche ab jedem gefundenen *a* Teilwort *aba* zu „verifizieren“

Beispiel: Arbeitsweise NTM M_{beisp}

- M_{beisp} „überläuft“ (in akzeptierender Rechnung) alle auftretenden Zeichen *a*, die nicht Beginn des Teilworts *aba* sind, „errät“ also die Stelle, an der das Teilwort *aba* beginnt und verifiziert *aba* erst bei geeignetem *a*

Im Allgemeinen:
NTM kann mit weniger Rechenschritten auskommen als DTM.
Im Beispiel: kein wesentlicher Unterschied.

Kontextfreie und entscheidbare Sprachen

Satz 6.5

Jede kontextfreie Sprache ist entscheidbar.

Denn:

- Jeder PDA ist eine spezielle TM. ($\Rightarrow$ Semi-Entscheidbarkeit)
- der CYK-Algorithmus kann durch eine TM berechnet werden (nach geeigneter Codierung der Eingaben)

Bemerkung

Es gibt entscheidbare Sprachen, die nicht kontextfrei sind.

Z.B. die Sprache $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.

Chomsky-Grammatiken

Frage: Gibt es einen zu TM äquivalenten generativen Beschreibungsformalismus für (semi-)entscheidbare Sprachen?

(ähnlich der Äquivalenz kontextfreie Grammatiken $\longleftrightarrow$ PDA)

Chomsky-Grammatiken liefern so einen Formalismus.

Zur Erinnerung: Definition einer Grammatik

Definition

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

- V endliche Menge der Nicht-Terminale (oder Variablen)
- Σ Alphabet der Terminale
- P endliche Menge der Regeln (oder Produktionen):

$$P \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$$

- $S \in V$ Startvariable.

Eingeschränkte Regelformen bei

- regulären Grammatiken: $A \to \varepsilon$ $A \to a$ $A \to aB$
- kontextfreien Grammatiken: $A \to w$

Zur Erinnerung: Beispiel einer Grammatik

Beispiel

Für $G = (V, \Sigma, P, S)$ mit $V = \{S, B, C\}$, $\Sigma = \{a, b, c\}$ und P:

$S \to aSBC$	$aB \to ab$	$cC \to cc$
$S \to aBC$	$bB \to bb$	
$CB \to BC$	$bC \to bc$	

Es gilt: $S \Rightarrow^* aaabbbccc$, denn

$$\begin{aligned} \underline{S} &\Rightarrow a\underline{S}BC \Rightarrow aa\underline{S}BCBC \Rightarrow aaaB\underline{CB}CBC \\ &\Rightarrow aaaBBC\underline{CB}C \Rightarrow aaaBB\underline{CB}CC \Rightarrow aa\underline{aB}BBCCC \\ &\Rightarrow aaa\underline{bB}BCCC \Rightarrow aaab\underline{bB}CCC \Rightarrow aaabb\underline{bC}CC \\ &\Rightarrow aaabbb\underline{cC}C \Rightarrow aaabbbc\underline{cC} \Rightarrow aaabbbccc \end{aligned}$$

G erzeugt die Sprache $L(G) = \{a^n b^n c^n \mid n \geq 1\}$.

Chomsky-Grammatiken und TM

Turing Maschinen und Chomsky Grammatiken sind gleich mächtige Algorithmenmodelle.

Satz 6.6

a) Zu jeder TM M gibt es eine Chomsky-Grammatik G mit $L(G) = L(M)$.

b) Zu jeder Chomsky-Grammatik G gibt es eine TM M mit $L(M) = L(G)$.

Korollar 6.7

Die von Chomsky-Grammatiken erzeugten Sprachen sind genau die semi-entscheidbaren Sprachen.

Zusammenfassung