
Formale Sprachen und Automaten

Turingmaschinen, Typ-0- und Typ-1-Grammatiken

Turingmaschinen

- Das Konzept der **Turingmaschine** wurde von dem Englischen Mathematiker Alan M. Turing (1912 - 1954) ersonnen.



- Der Aufsatz Turing (1936) hat die Berechnbarkeitstheorie begründet und damit den Grundstein für die theoretische Informatik gelegt.

Turingmaschinen 2

- Die Turingmaschine (**TM**) ist im Prinzip das Modell eines modernen Computers und kann alles, was ein solcher Computer kann.
- Wie PDAs und EAs haben TMn ein Eingabeband, das von einem Kopf abgetastet wird.
- Wie PDAs und EAs ändern TMn ihren Zustand in Abhängigkeit von der gelesenen Eingabe.
- Was TMn von PDAs und EAs unterscheidet:
 1. Der Kopf kann sowohl nach **links** als auch nach **rechts** verschoben werden.
 2. Der Kopf kann sowohl **lesen** als auch **schreiben**.
- Deterministische und nicht-deterministische Turingmaschinen sind äquivalent.

TM: Intuitives Beispiel

- $L_1 = \{w\#w \mid w \in \{0, 1\}^*\}$
- Was eine TM M_1 macht, wenn sie L_1 akzeptiert:
 1. Scannt das Eingabeband von links nach rechts, um zu überprüfen, dass auch nur ein Symbol $\#$ auftaucht.
 2. Fährt zick-zack über das Band zwischen **korrespondierenden** Positionen auf jeder Seite von $\#$ hin und her, um zu überprüfen, dass diese Positionen mit demselben Symbol gefüllt sind.
 3. Wenn die Symbole nicht übereinstimmen, dann lehnt M_1 die Eingabe ab.
 4. Löscht die schon überprüften Symbole.
 5. Wiederholt das Verfahren solange, bis alle Symbole links von $\#$ gelöscht sind.
 6. Überprüft, ob rechts von $\#$ noch Symbole übrig sind. Wenn ja, lehnt M_1 ab, wenn nein, akzeptiert M_1 .

TM: Intuitives Beispiel 2

- $L_2 = \{a^{2^n} | n \geq 0\}$
- Was TM M_2 macht, wenn sie L_2 akzeptiert:
 1. Geht von links nach rechts und löscht jedes zweite a .
 2. Wenn nur ein a auf dem Band ist, akzeptiert M_2 .
 3. Wenn die Zahl der a s ungerade ist und größer als 1, dann weist M_2 die Eingabe zurück.
 4. Kehrt zum linken Rand der Eingabe zurück.
 5. Beginnt wieder mit Punkt 1.

TM: Intuitives Beispiel 3

- $L_2 = \{a^n b^n c^n | n \geq 0\}$
- Was TM M_3 macht, wenn sie L_3 akzeptiert:
 1. Geht von links nach rechts; wenn kein Symbol auf dem Band ist, dann akzeptiert M_3 .
 2. Kehrt zum linken Rand des Bandes zurück.
 3. Wenn es a s gibt, ersetzt M_3 das linkeste durch ein x . Wenn nicht, dann akzeptiert M_3 , falls es keine b s und c s gibt, sonst weist M_3 zurück.
 4. Wenn es b s gibt, dann ersetzt M_3 das linkeste durch ein x , sonst weist M_3 zurück.
 5. Wenn es c s gibt, dann ersetzt M_3 das linkeste durch ein x , sonst weist M_3 zurück.
 6. Beginnt wieder mit Punkt 2.

Formale Spezifikation der Turingmaschine

- Eine (deterministische) Turingmaschine M ist ein 6-Tupel $(K, \Sigma, \Gamma, \delta, s, \square)$; dabei ist
 1. K eine endliche Menge von Zuständen
 2. Σ das Eingabealphabet
 3. $\Gamma \supset \Sigma$ das Arbeitsalphabet
 4. $\delta : K \times \Gamma \mapsto K \times \Gamma \times \{L, R, N\}$ die Übergangsfunktion
 5. s der Startzustand
 6. $\square \in (\Gamma - \Sigma)$ das **Blank**
- δ muss nicht für jedes Paar aus $K \times \Gamma$ definiert sein.

Formale Spezifikation der Turingmaschine 2

- Intuitiv bedeutet $\delta(q, a) = (q', b, x)$:
 1. Wenn sich M im Zustand q befindet und
 2. unter dem Kopf das Symbol a steht, dann
 3. geht M im nächsten Schritt in den Zustand q' ,
 4. überschreibt a mit b und
 5. führt dann die Kopfbewegung $x \in \{L, R, N\}$ aus, wobei
 - (a) L für **links**,
 - (b) R für **rechts** und
 - (c) N für **neutral** (also stehenbleiben) steht

Konfiguration einer TM

- Eine **Konfiguration** einer TM M ist ein Element $k \in \Gamma^* K \Gamma^*$.
- Wie immer ist die Konfiguration eine Momentaufnahme der Maschine.
- $k = \alpha q \beta$ wird so interpretiert, dass
 1. $\alpha \beta$ der nicht-leere bzw. schon besuchte Teil des Bandes ist
 2. q der Zustand ist, in dem sich M gerade befindet
 3. der Kopf von M sich über dem ersten Symbol von β befindet

Konfiguration einer TM 2

- Eine TM M startet mit einer Eingabe $x \in \Sigma^*$ auf dem Eingabeband.
- Der Kopf ist am Anfang über dem ersten Zeichen von x plziert.
- Dies ist dargestellt durch die Startkonfiguration sx .
- Alle Felder des Eingabebandes, welche nicht mit Eingabesymbolen $\in \Sigma$ beschrieben sind, enthalten \square .

Konfigurationen einer TM 3

- Die Relation \vdash_M zwischen zwei Konfigurationen von M ist definiert wie folgt:
- $a_1 \dots a_m q b_1 \dots b_n \vdash_M$
 1. $a_1 \dots a_m q' c b_2 \dots b_n$,
wenn $\delta(q, b_1) = (q', c, N)$, $m \geq 0, n \geq 1$
 2. $a_1 \dots a_m c q' b_2 \dots b_n$,
wenn $\delta(q, b_1) = (q', c, R)$, $m \geq 0, n \geq 2$
 3. $a_1 \dots a_{m-1} q' a_m c b_2 \dots b_n$,
wenn $\delta(q, b_1) = (q', c, L)$, $m \geq 1, n \geq 1$
- Es fehlen noch zwei Sonderfälle:
 1. $a_1 \dots a_m q b_1 \vdash_M a_1 \dots a_m c q \square$, wenn $\delta(q, b_1) = (q', c, R)$
($n = 1$, M läuft nach rechts und trifft auf ein \square)
 2. $q b_1 \dots b_n \vdash_M q' \square c b_2 \dots b_n$, wenn $\delta(q, b_1) = (q', c, L)$
($m = 0$, M läuft nach links und trifft auf \square)

Akzeptierte Sprachen

- Die Relation \vdash_M^* verbindet zwei Konfigurationen einer TM in beliebig aber endlich vielen Schritten. \vdash_M^* ist der reflexiv-transitive Abschluss von \vdash_M .
- Eine **Berechnung** der Länge n von M ist eine Sequenz von Konfigurationen k_0, k_1, \dots, k_n mit $n \geq 0$, so dass $k_0 \vdash_M k_1 \vdash_M k_2 \vdash_M \dots \vdash_M k_n$.
- Die von einer Turingmaschine M **akzeptierte Sprache** ist definiert als $T(M) = \{w \in \Sigma^* \mid s x \vdash_M^* \alpha q \gamma \beta$ wobei $\alpha, \beta \in \Gamma^*, \gamma \in \Gamma$ und δ ist nicht definiert für q und $a\}$.
- Das heißt, M akzeptiert w , wenn M **anhält**, da es keinen weiteren Übergang gibt. M lehnt die Eingabe ab, wenn M **unendlich weiterrechnet**.
- Eine Sprache L heißt **turingakzeptierbar** genau dann, wenn es eine Turingmaschine gibt, die L akzeptiert.

Beispiel TM

- Sei $M_1 = (K, \Sigma, \Gamma, \delta, q_0, \square)$ eine TM mit
 1. $K = \{q_0, q_1\}$
 2. $\Sigma = \{a, b\}$
 3. $\Gamma(\supset \Sigma) = \{a, b, \square\}$
 4. δ ist definiert wie folgt:
$$\begin{aligned}\delta(q_0, a) &= (q_1, b, N) \\ \delta(q_0, b) &= (q_1, a, N) \\ \delta(q_0, \square) &= (q_1, \square, N) \\ \delta(q_1, a) &= (q_0, a, R) \\ \delta(q_1, b) &= (q_0, b, R)\end{aligned}$$
- M_1 transformiert von links nach rechts as in bs und umgekehrt, bis M_1 das erste \square findet.
- M_1 lässt \square unverändert, wechselt in Zustand q_1 und hält an, da δ nicht für (q_1, \square) definiert ist.

Beispiel TM 2

- Da der Kopf sicher irgendwann \square liest, hält M immer an.
- Enthielte δ aber etwa die Abbildung $\delta(q_1, \square) = (q_1, \square, R)$, dann würde M_1 immer weiter laufen, wenn sie einmal das rechte Ende der Eingabe erreicht hat.
- Etwas ähnliches könnte man erreichen durch die Abbildung $\delta(q_1, \square) = (q_1, \square, N)$ (nur würde sich der Kopf nicht mehr bewegen).
- Oft werden Turingmaschinen mit finalen Zuständen definiert, und Akzeptanz durch Anhalten **und** Erreichen eines finalen Zustandes. Diese Definitionen sind äquivalent.

Beispiel TM 3

- Sei $M_2 = (K, \Sigma, \Gamma, \delta, q_0, \square)$ eine TM mit
 1. $K = \{q_0, q_1\}$
 2. $\Sigma = \{a, b\}$
 3. $\Gamma(\supseteq \Sigma) = \{a, b, \square\}$
 4. δ ist definiert wie folgt:
$$\begin{aligned}\delta(q_0, a) &= (q_1, a, R) \\ \delta(q_0, b) &= (q_0, b, R) \\ \delta(q_0, \square) &= (q_0, \square, R) \\ \delta(q_1, a) &= (q_1, a, R) \\ \delta(q_1, b) &= (q_1, b, R)\end{aligned}$$
- Was M_2 macht:
 1. M_2 bleibt in q_0 solange sie bs liest.
 2. Wenn M_2 ein a liest wechselt sie in q_1 und liest weiter, bis sie das erste \square findet und dann anhält.
 3. Liest M_2 \square in Zustand q_0 , hält M_2 nicht an.
 4. $T(M) = \{w \in \{a, b\}^* \mid w \text{ enthält wenigstens ein } a\}$

Entscheidbare Sprachen

- Eine TM $M = (K, \Sigma, \Gamma, \delta, s, \square, F)$ mit einer **Menge von finalen Zuständen** $F \subseteq K$ **entscheidet** eine Sprache $L \subseteq \Sigma^*$ wenn für jede Kette $w \in \Sigma^*$ gilt:
 1. Wenn $w \in L$, dann hält M in einem finalen Zustand an
 2. Wenn $w \notin L$, dann hält M in einem nicht-finalen Zustand an
- Eine TM M , die eine Sprache L entscheidet muss bei beliebigen Eingaben ($w \in L$ oder $w \notin L$) schlussendlich immer **anhalten**.
- Eine Sprache L heißt **turingentscheidbar** genau dann, wenn es eine Turingmaschine gibt, die L entscheidet.

Entscheidbare Sprachen 2

- Die TM M_2 kann man konvertieren in eine TM M'_2 , die dieselbe Sprache **entscheidet**, welche M_2 **akzeptiert** (entferne die Abbildung $\delta(q_0, \square) = (q_0, \square, R)$ und mache q_1 zum einzigen finalen Zustand).
- Frage: Kann eine solche Konversion **immer mechanisch** durchgeführt werden (z.B. durch eine Turingmaschine)?
- Antwort: Nein! Das heißt, es gibt Sprachen, die turingakzeptierbar sind, aber nicht turingentscheidbar.

Berechenbare Funktionen

- TMn werden oft als Werkzeug angesehen, um **Funktionen** zu berechnen.
- Eine Funktion $f : \Sigma^* \mapsto \Sigma^*$ heißt **turingberechenbar**, falls es eine (deterministische) TM M gibt, so dass für alle $x, y \in \Sigma^*$ gilt:

$$f(x) = y \text{ genau dann, wenn} \\ s \ x \vdash_M^* \square \dots \square q_e y \square \dots \square,$$

wobei $q_e \in F$ und M hält an.

- x ist die Eingabe und y die Ausgabe.
- Wenn M eine Sprache L entscheidet, dann berechnet M die **charakteristische** Funktion von L .
- Eine Funktion f heißt **rekursiv**, wenn es eine TM M gibt, so dass M f berechnet und bei jedem Input hält.

Berechenbare Funktionen 2

- Wenn M für bestimmte Eingaben nicht anhält, dann ist die Funktion für diese Eingaben nicht definiert.
- M berechnet in diesem Fall eine **partiell rekursive** Funktion.
- Anders gesagt: M kann in eine **Schleife** geraten und rechnet unendlich lange weiter.
- Solange M nicht angehalten hat, weiß man nicht, ob
 1. M im nächsten Moment anhalten wird
 2. M immer weiter rechnen wird
- Man nennt den Wertebereich einer solchen Funktion (der eine Menge, eine Sprache ist) in so einem Falle **semientscheidbar**.

Numerische berechenbare Funktionen

- Eine Funktion $f : \mathcal{N}^k \mapsto \mathcal{N}$ heißt **turingberechenbar**, falls es eine (deterministische) TM M gibt, so dass für alle $n_1, \dots, n_k, m \in \mathcal{N}$ gilt:

$$f(n_1, \dots, n_k) = m \text{ genau dann, wenn} \\ s \text{ } bin(n_1)\#bin(n_2)\#\dots\#bin(n_k) \vdash_M^* \\ \square \dots \square_{q_e} bin(m) \square \dots \square,$$

wobei $q_e \in F$, M bei jeder Eingabe hält und $bin(n)$ die binäre Darstellung der Zahl n ist.

Rekursiv aufzählbare Sprachen

- Man kann eine TM auch als Generator betrachten.
- Sei M eine TM, deren Eingabe die unäre Kodierung der natürlichen Zahlen ist (n *as* stehen für Zahl n).
- M generiert jeweils die Kette w (welche etwas anderes als *as* enthalten kann), die nach einer Berechnung auf dem Eingabeband steht (die Ausgabe).
- Die Menge aller dieser Outputketten, mit den natürlichen Zahlen als Input, nennt man eine von M **rekursiv aufzählbare** Menge.
- Eine Sprache L ist rekursiv aufzählbar genau dann, wenn es eine TM M gibt, die L semientscheidet (äquivalent: M turingakzeptiert L).

Synopsis

- Die folgenden Feststellungen sind äquivalent.
 1. L wird von einer TM akzeptiert.
 2. L wird von einer TM semientschieden.
 3. L wird von einer TM rekursiv aufgezählt.
 4. L ist der Wertebereich einer partiell rekursiven Funktion.

Rekursive und rekursiv aufzählbare Mengen

- Theorem: Eine Sprache L ist **rekursiv** genau dann, wenn sowohl L als auch \bar{L} rekursiv aufzählbar (turingakzeptierbar) sind.
- Theorem: Die **rekursiven** Mengen sind genau die **turingentscheidbaren** Mengen.
- Theorem: Die **rekursiven** Mengen sind genau die Mengen, die **rekursive charakteristische Funktionen** haben.
- Theorem: Die **rekursiv aufzählbaren** Mengen sind genau die Mengen, die **partiell rekursiv charakteristische** Funktionen haben.

Synopsis 2

- Die folgenden Feststellungen sind äquivalent:
 1. L ist eine rekursive Menge.
 2. L ist turingentscheidbar.
 3. L und \bar{L} sind rekursiv aufzählbar.
 4. L hat eine charakteristische Funktion, die (total) rekursiv ist.

Typ-0-Grammatiken

- Bei einer **unbeschränkten** Grammatik $G = \{V, \Sigma, R, S\}$ gilt: die LS jeder Regel muss mindestens ein Nichtterminal enthalten.
- Man nennt solche Grammatiken auch **Typ-0-Grammatiken**.
- Typ-0-Grammatiken können Sprachen generieren, die nicht kontextfrei sind.

Typ-0-Grammatiken 2

- Beispiel 1: Sei $G_1 = \{V, \Sigma, R, S\}$ eine Typ-0-Grammatik, wobei
 1. $V = \{S, A, B, C, a, b, c\}$
 2. $\Sigma = \{a, b, c\}$
 3. $R = \{S \rightarrow SABC, S \rightarrow \epsilon, AB \rightarrow BA, BA \rightarrow AB, AC \rightarrow CA, CA \rightarrow AC, BC \rightarrow CB, CB \rightarrow BC, A \rightarrow a, B \rightarrow b, C \rightarrow c\}$
- G_1 generiert die Sprache $L = \{w \in \{a, b, c\}^* \mid \text{Anzahl der } as, bs \text{ und } cs \text{ ist gleich}\}$.
- G_1 geht folgendermaßen vor:
 1. Zunächst generiert $G_1 (ABC)^n$
 2. G_1 permutiert die generierten Nichtterminale (durch Regeln wie $BA \rightarrow AB$ etc.).
 3. Schließlich werden die Nichtterminale in Terminale umgewandelt.

Typ-0-Grammatiken 3

- Beispiel 2: Sei $G_2 = \{V, \Sigma, R, S\}$ vom Typ 0.
 1. $V = \{S, S' A, B, \#, a, b\}$
 2. $\Sigma = \{a, b\}$
 3. $R = \{S \rightarrow \#S'\#, S' \rightarrow aAS', S' \rightarrow bBS', S' \rightarrow \epsilon, Aa \rightarrow aA, Ab \rightarrow bA, Ba \rightarrow aB, Bb \rightarrow bB, \#a \rightarrow a\#, \#b \rightarrow b\#, A\# \rightarrow \#a, B\# \rightarrow \#b, \#\# \rightarrow \epsilon\}$
- G_2 generiert $L = \{ww \mid w \in \{a, b\}^*\}$ wie folgt:
 1. Zunächst generiert G_2 aAs und bBs zwischen zwei Grenzsymbolen $\#$.
 2. Dann wandern die Nichtterminalsymbole nach rechts, hüpfen über die Grenze $\#$ und werden zu Terminalsymbolen.
 3. Die Terminale hüpfen über das linke $\#$.
 4. Schließlich werden die beiden nun aneinander-grenzenden Symbole $\#$ aufgelöst.

Typ-0-Grammatiken und Turingmaschinen

- Theorem: Die Sprachen, die von Typ-0-Grammatiken generiert werden, sind genau die Sprachen, die von Turingmaschinen akzeptiert werden (also die rekursiv aufzählbaren Sprachen).
- Es gilt also, dass folgende Feststellung äquivalent sind:
 1. L wird von einer TM akzeptiert.
 2. L wird von einer TM semientschieden.
 3. L wird von einer TM rekursiv aufgezählt.
 4. L ist der Wertebereich einer partiell rekursiven Funktion.
 5. L wird von einer Typ-0-Grammatik generiert.
- Konsequenz: Typ-0-Sprachen sind nicht generell entscheidbar.

Linear beschränkte TMn

- Angenommen der linke und der rechte Rand der Eingabe einer TM ist durch die Grenzsymbole [und] markiert.
- Eine TM M heißt **linear beschränkt**, wenn für alle $a_1a_2 \dots a_{n-1}a_n \in \Sigma^+$ und alle Konfigurationen $\alpha q\beta$ mit $s[a_1a_2 \dots a_{n-1}a_n] \vdash_M^* \alpha q\beta$ gilt: $|\alpha\beta| = n$.
- Das heißt: Linear beschränkte TMn (**lbTMn**) können den Teil des Eingabebandes, auf dem die Eingabe steht, niemals verlassen.
- Die von einer lbTM **akzeptierte Sprache** ist definiert als $T(M) = \{a_1a_2 \dots a_{n-1}a_n \in \Sigma^* \mid s[a_1a_2 \dots a_{n-1}a_n] \vdash_M^* \alpha q\beta, \text{ wobei } \alpha, \beta \in \Gamma^*, q \in F \text{ und } M \text{ hält an}\}$
- Es ist nicht bekannt ob deterministische und nicht-deterministische lbTM äquivalent sind!

Typ-1-Grammatiken und lbTMn

- Für **kontextsensitive** Grammatiken, die man auch **Typ-1-Grammatiken** nennt, gilt: jede Regel ist von der Form $\alpha A\beta \rightarrow \alpha\psi\beta$, wobei $\psi \neq \epsilon$.
- Bei Typ-1-Grammatiken muss also die rechte Seite einer Regel mindestens so lang sein, wie die linke Seite.
- Typ-1-Grammatiken können Sprachen generieren, die nicht kontextfrei sind.
- Theorem: Die Sprachen, die von Typ-1-Grammatiken generiert werden, sind genau die Sprachen, die von nichtdeterministischen lbTMn akzeptiert werden.

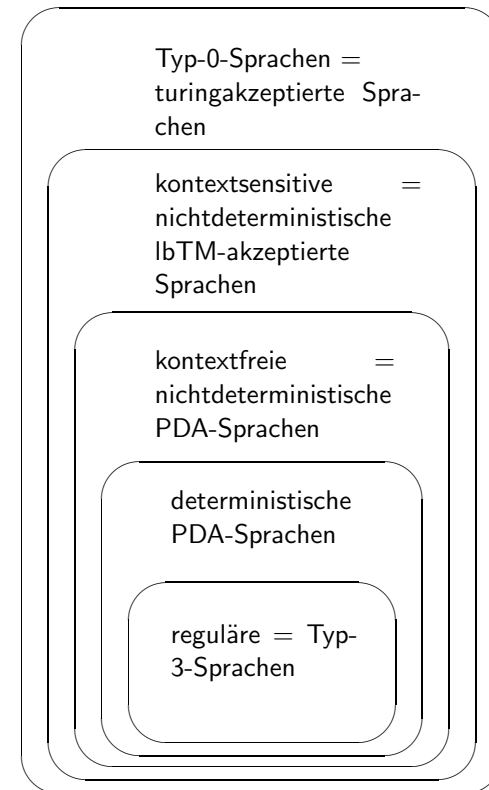
Die Chomskyhierarchie

- Kontextfreie Grammatiken nennt man auch **Typ-2**-Grammatiken, reguläre Grammatiken auch **Typ-3**-Grammatiken.



- Die Einteilung in Typ-0-, Typ-1-, Typ-2- und Typ-3-Grammatiken geht auf Noam Chomsky zurück.

Die Chomskyhierarchie 2



Übersicht

Konzepte	
Typ-3	reguläre Grammatik DEA NDEA regulärer Ausdruck
Det.kf.	deterministischer Kellerautomat
Typ-2	kontextfreie Grammatik nichtdeterministischer Kellerautomat
Typ-1	kontextsensitive Grammatik linear beschränkte Turingmaschine
Typ-0	unbeschränkte Grammatik Turingmaschine

Übersicht 2

Nicht-/Determinismus		
Nichtdet. Automat	Det. Automat	äquivalent?
NDEA	DEA	ja
PDA	DPDA	nein
lbTM	det. lbTM	?
TM	det. TM	ja

Abschlusseigenschaften					
	Schnitt	Verein.	Kompl.	Prod.	Stern
Typ-3	ja	ja	ja	ja	ja
det.ktf.	nein	nein	ja	nein	nein
Typ-2	nein	ja	nein	ja	ja
Typ-1	ja	ja	ja	ja	ja
Typ-0	ja	ja	nein	ja	ja

Übersicht 3

Entscheidbarkeit		
	Wortproblem	Leerheitsproblem
Typ-3	ja	ja
det.ktf.	ja	ja
Typ-2	ja	ja
Typ-1	ja	nein
Typ-0	nein	nein

Literatur

Turing, Alan M. (1936): 'On computable numbers with an application to the Entscheidungsproblem', *Proceedings of the London Mathematics Society* 2(42), 230–265.