

Teil 4 Turingmaschinen, Entscheidbarkeit

Kontextfreie Grammatiken sind (siehe Teil 3) allgemeiner als deterministische Kellerautomaten. Schon weil nicht alle Grammatiken kontextfrei sind, ist klar, daß Kellerautomaten nicht "alle" formalen Probleme lösen können. Also werden wir nicht umhin kommen, nach mächtigeren Automaten Ausschau zu halten.

Betrachten wir zunächst ein weiteres Beispiel:

$$L = \{0^n 1^n 2^n \mid n = 0, 1, 2, \dots\}$$

Hierbei handelt es sich um die Menge aller Wörter der Art

012, 001122, 000111222 ...

Ein Automat mit *einem* Keller reicht für diese Sprache nicht. Man bräuchte zwei Keller, je einen für die Nullen und die Einsen, um dann beim Lesen der Zweien zu prüfen, ob beide Keller gleichzeitig leerlaufen.

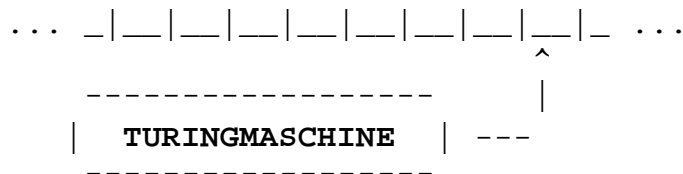
Für das obige L reicht ein Zweikellerautomat; aber bei

$$\{0^n 1^n 2^n 3^n \mid n = 0, 1, 2, \dots\}$$

wäre es bequem, drei Keller zu haben..., und daraus erkennen wir, daß wir so nicht weiter kommen. Offensichtlich brauchen wir etwas grundlegend Besseres.

Wir wählen die sogenannten *Turingmaschinen*. Diese virtuellen Maschinen haben auf den ersten Blick nur sehr bescheidene Möglichkeiten. Sie haben überhaupt keine Keller oder sonstige externen Speicher. Eine Turingmaschine ist ein endlicher Automat mit der zusätzlichen Eigenschaft, daß sie den Lesekopf nicht nur nach rechts, sondern auch nach links bewegen kann und daß sie außerdem auf's Band schreiben kann.

Ferner nehmen wir an, daß das Band nach beiden Seiten und nicht nur nach rechts unbeschränkt ist.



Wie ein endlicher Automat funktioniert eine Turingmaschine in ganz natürlicher Weise. In Abhängigkeit

- vom inneren Zustand und
- vom gelesenen Zeichen

wird

- die Turingmaschine in einen neuen Zustand versetzt

- das gelesene Zeichen durch ein (möglicherweise anderes) Zeichen ersetzt und
- der Lesekopfkopf eine Position nach links, nach rechts oder auch gar nicht bewegt.

Die früheren Automaten stoppten, wenn der Eingabestring zu Ende war. Bei Turingmaschinen muß ein gelesenes Blank keineswegs STOP bedeuten. Die Zustandsüberföhrungsfunktion (das *Programm*) muß also auch auf Blanks reagieren, die wir deswegen explizit in den Zeichensatz mitaufnehmen:

Definition (deterministische Turingmaschine, Allan Turing)

Eine Turingmaschine $M = (Q, \Sigma, \Gamma, b, q_0, \rho, F)$

- befindet sich stets in einem Zustand einer endlichen Zustandsmenge. Diese Zustandsmenge Q enthält einem speziellen Startzustand q_0 und eine Teilmenge F von Q der **Ja-Zustände**.
- liest von einem Band Zeichen eines endlichen Bandalphabeths Γ
- wird in Gang gehalten von einer Zustandsüberföhrungsfunktion (von einem **Programm**)

$$\rho: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, N\}$$

die also einen Zustand abhängig von aktuell gelesenen Zeichen in einen neuen Zustand überföhrt, ein Zeichen aufs Band schreibt und den Kopf möglicherweise bewegt.

Der Kopf der Turingmaschine weist zu Beginn auf das erste Symbol des Eingabestrings. Von den übrigen Bandpositionen können wir uns vorstellen, daß sie mit einem speziellen **Blank-Symbol** $b \in \Gamma - \Sigma$ gefüllt sind. Der Eingabestring enthält nur Zeichen des Eingabealphabets Σ (also keine Blanks).

In vielen Fällen ist $\Sigma = \{0, 1\}$ und $\Gamma = \{0, 1, b\}$.

Die früheren Automaten stoppen nach Abarbeiten des Eingabestrings. Turingmaschinen *stoppen* nicht in diesem Sinne; es kann höchstens sein, daß "nichts mehr passiert". Wir sagen: Ein Zustand ist ein **Stoppzustand**, wenn unabhängig vom gelesenen Symbol

- der Zustand der gleiche bleibt
- das gelesene Symbol nicht geändert wird und
- der Kopf sich nicht bewegt.

Wie früher sagen wir: Ein Eingabestring w wird **akzeptiert**, wenn die Maschine bei der Bearbeitung von w in einem Ja-Zustand stoppt.

Eine weitere Subtilität wird notwendig beim Begriff der von einer Turingmaschine akzeptierten Sprache. Es kann ja sein, daß eine Turingmaschine überhaupt nicht stoppt (weil sie beispielsweise zwischen zwei Zuständen hin- und her oszilliert). Beim Laufen der Turingmaschine weiß man dann nicht, ob der

String abgelehnt ist oder ob er irgendwann vielleicht doch noch akzeptiert wird. Aus diesem Grund benutzen wir zwei unterschiedliche Entscheidbarkeits-Begriffe:

Definition

- Eine Menge (Sprache) L ist **(strikt) entscheidbar**, wenn es eine Turingmaschine gibt, die bei allen Eingabestrings stoppt und die den String akzeptiert genau dann, wenn er zu L gehört.
- Eine Menge (Sprache) L ist **semi-entscheidbar** (auch **aufzählbar**, oder **rekursiv aufzählbar** genannt), wenn es eine Turingmaschine gibt, die den String genau dann akzeptiert, wenn er zu L gehört (und die bei anderen Strings vielleicht überhaupt nicht stoppt).

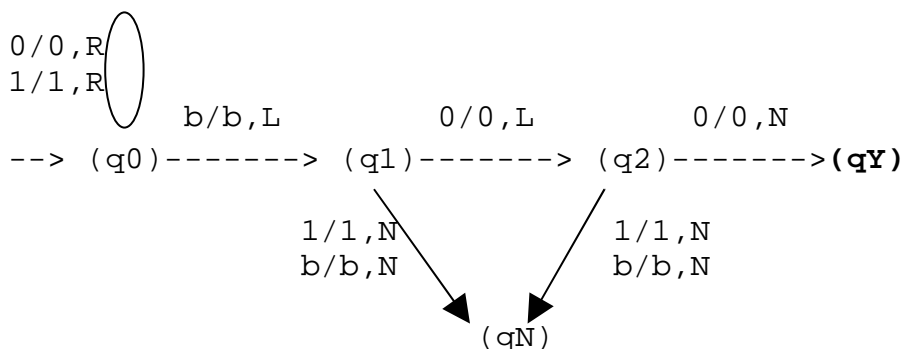
Beispiel (Teilbarkeit durch 4)

Das folgende Turingprogramm entscheidet, ob ein gegebener Binärstring mindestens zwei Nullen am Ende hat (ob also die durch den String dargestellte natürliche Zahl durch 4 teilbar ist).

	0	1	b	
--> q0	q0,0,R	q0,1,R	q1,b,L	nach rechts
q1	q2,0,L	qN,1,N	qN,b,N	letztes Bit testen
q2	qY,0,N	qN,1,N	qN,b,N	vorletztes Bit testen
qY	-	-	-	Ja-Zustand
qN	-	-	-	

Hinweis: qY und qN sollen Stopzustände sein (siehe oben), wofür wir in der Tabelle kurz Striche "-" notieren.

Dieselbe Turingmaschine statt Tabelle als Graph:



Ein Programmlauf:

String 100:

$q_0 100 \rightarrow 1q_0 00 \rightarrow 10q_0 0 \rightarrow 100q_0 \rightarrow 10q_1 0 \rightarrow 1q_2 00 \rightarrow 1q_r 00$
 akzeptiert

Beispiel Die von der Grammatik

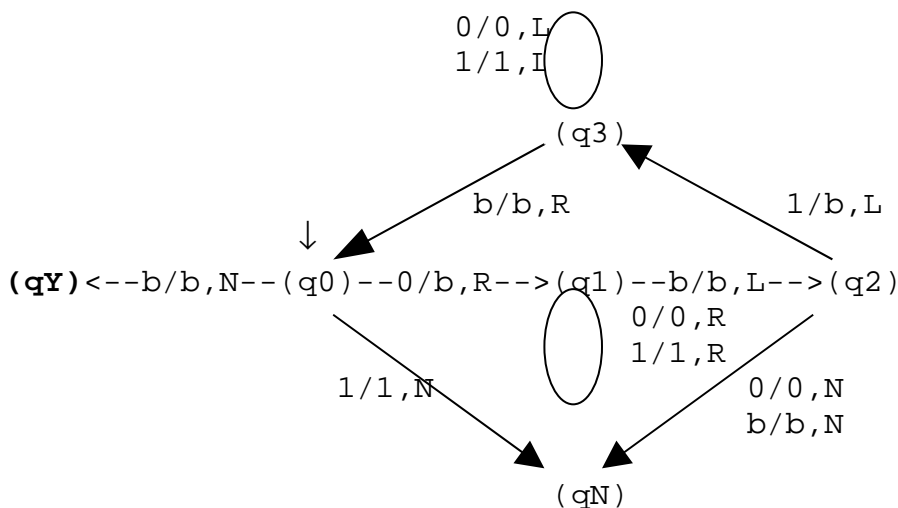
$$S \rightarrow \epsilon \mid (S)$$

abgeleiteten einfach geschachtelten korrekten Klammern konnten mit endlichen Automaten nicht erkannt werden. Mit Kellerautomaten ging es. Mit Turingmaschinen geht es leicht. Aus praktischen Gründen ersetzen wir zunächst die Klammernmenge durch die formale Sprache

$$L = \{0^n 1^n \mid n=0,1,2,\dots\}$$

Die Idee für die Turingmaschine ist: Der Lesekopf läuft beständig hin und her, löscht links eine 0, löscht rechts eine korrespondierende 1 und akzeptiert den String, wenn schließlich keine Nullen und Einsen übrigbleiben:

		0	1	b	
-->	q0	q1,b,R	qN,1,N	qY,b,N	linke 0 löschen
	q1	q1,0,R	q1,1,R	q2,b,L	ganz nach rechts
	q2	qN,0,N	q3,b,L	qN,b,N	rechte 1 löschen
	q3	q3,0,L	q3,1,L	q0,b,R	ganz nach links
	qY	-	-	-	
	qN	-	-	-	



Einige Programmläufe:

String 0011:

$q_0 0011 \rightarrow q_1 011 \rightarrow 0 q_1 11 \rightarrow 01 q_1 1 \rightarrow 011 q_1 \rightarrow 01 q_2 1 \rightarrow 0 q_3 1 \rightarrow q_3 01$
 $\rightarrow q_3 01 \rightarrow q_0 01 \rightarrow q_1 1 \rightarrow 1 q_1 \rightarrow q_2 1 \rightarrow q_3 \rightarrow q_0 \rightarrow q_Y$
 String akzeptiert

String 001:

$q_0 001 \rightarrow q_1 01 \rightarrow 0 q_1 1 \rightarrow 01 q_1 \rightarrow 0 q_2 1 \rightarrow q_3 0 \rightarrow q_3 0 \rightarrow q_0 0 \rightarrow$
 $q_1 \rightarrow q_2 \rightarrow q_N$
 String abgelehnt

String ϵ (leerer String):

$q_0 \rightarrow q_Y$ akzeptiert

String 011: ... (wird abgelehnt)

Beispiel

Die binären Palindrome (also die binären Strings, die sich von hinten genauso lesen wie von vorn) konnten mit deterministischen Kellerautomaten nicht erkannt werden. Mit Turingmaschinen geht es ohne große Probleme.

	0	1	b	
q_0	10,b,R	11,b,R	q_Y ,b,N	Startzustand q_0
10	10,0,R	10,1,R	r_0 ,b,L	Akzeptierender Endzustand q_Y (YES)
11	11,0,R	11,1,R	r_1 ,b,L	
r_0	r, b,L	q_N ,1,N	q_Y ,b,N	$\Sigma = \{0,1\}, \Gamma = \{0,1,b\}$
r_1	q_N ,0,N	r, b,L	q_Y ,b,N	
r	r, 0,L	r, 1,L	q_0 ,b,R	
q_Y	-	-	-	
q_N	-	-	-	

Die Maschine funktioniert auf folgende Weise: Beim Start sitzt der Lesekopf im Zustand q_0 auf dem ersten Zeichen. Liest der Kopf dort eine Zahl, so löscht er sie, und die Maschine geht (je nach erstem gelesenen Zeichen) in den Zustand 10 bzw. 11 (sie "merkt" sich also das erste Zeichen) und bleibt in diesem Zustand, bis der Kopf das Ende des Eingabestrings erreicht.

Erreicht die Maschine im Zustand $l_i, i=0,1$ den ersten Blank, so geht sie in den Zustand r_i ("rechts_i") und einen Schritt zurück. Findet die Maschine dort gerade das zu i komplementäre Zeichen, so lehnt sie den String ab. Andernfalls löscht sie das aktuelle Zeichen (also das letzte Zeichen des Strings), geht in den Zustand r und läuft zurück bis vor den Anfang des Reststrings. Dort geht der Kopf eine Position nach rechts (also auf den Anfang des Reststrings), die Maschine nimmt wieder Zustand q_0 an, und eine neue Runde beginnt. - Die obige Tabelle zeigt die technischen Einzelheiten.

[Darstellung als Graph]

Wir testen unser Programm an einigen Strings:

String 010:

$q_0 010 \rightarrow l_0 10 \rightarrow l_1 0 \rightarrow l_0 1 \rightarrow l_1 r_0 \rightarrow r_1 \rightarrow r \ 1 \rightarrow q_0 1 \rightarrow l_1$
 $\rightarrow r_1 \rightarrow q_r$ String akzeptiert

0110: \rightarrow akzeptiert
0100: \rightarrow abgelehnt
01: \rightarrow abgelehnt
1: \rightarrow akzeptiert
11: \rightarrow akzeptiert

Randbemerkung: Unser bisherigen drei Turingprogramme stoppen für beliebige Eingabestrings, d.h. sie stoppen immer. Die betreffenden formalen Sprachen sind also im strikten Sinne entscheidbar und nicht nur *semi*entscheidbar.

Daß die Unterscheidung zwischen strikter und semistrikter Entscheidbarkeit tatsächlich wirksam werden kann, wird sich gleich an einem überraschend einfachen Beispiel herausstellen.

Schon weil Turingmaschinen derartig simpel sind (jedenfalls simpel scheinen), ist es wenig überraschend, daß auch sie nicht alle formalen Probleme lösen können. Ein in ganz einfachen Worten beschreibbares Problem, daß sich der Lösbarkeit mittels Turingmaschinen entzieht, kennen wir schon (siehe Teil 1):

ZEHNTES HILBERTSCHES PROBLEM (1900)

Gegeben: Eine Polynomgleichung mit ganzzahligen Koeffizienten.

Frage: Hat die Gleichung eine ganzzahlige Lösung?

Ein einheitlicher Algorithmus zur Lösung dieses 10. Hilbertschen Problems, der also für beliebige ganzzahlige Polynome in endli-

cher Zeit die korrekte Ja/Nein-Antwort liefert, ist nicht nur nicht bekannt, sondern ist für Turingmaschinen *nachgewiesenermaßen* unmöglich. Dieses Problem ist im Sinne des oben erläuterten *strikten* Entscheidbarkeits-Begriffs für Turingmaschinen unentscheidbar.

Randbemerkung: Das 10. Hilbertsche Problem ist durchaus semi-entscheidbar (wie nämlich?).

[Post'sches Korrespondenzproblem und Halteproblem (Schöning S.131/32) erläutern.]

Diese Unentscheidbarkeit des 10. Hilbertschen Problems für Turingmaschinen klingt zunächst wenig dramatisch: Turingmaschinen sind eben zu simpel, und wir brauchen Maschinenmodelle, die mächtiger sind.

Danach wurde auch gesucht. Untersucht wurden

- Maschinen mit mehr als einer Speicherspurauf dem Arbeitsband
- Maschinen mit mehreren Arbeitsbändern und pro Band einem Kopf, der sich selbständig bewegen kann.
- Maschinen, die andere Turingmaschinen als Unterprogramme benutzen.
- RAM: Random Access Machine (Registermaschinen, die mit Befehlen nach Assembler-Art programmiert werden können)

Mit solchen verallgemeinerten Turingmaschinen ist die Programmierung eines konkreten Problems oft sehr viel bequemer als mit dem eingeschränkten Turingmodell, wie wir es oben vorgestellt haben.

Es hat sich aber gezeigt: Alle Probleme, die sich mit irgendwelchen verallgemeinerten Maschinen lösen lassen, können mit unserem Grundmodell im Prinzip auch schon gelöst werden. Es hat sich daher die (natürlich unbeweisbare) Überzeugung festgesetzt (**These von Church, 1936**), daß die Turingmaschine das Universalwerkzeug zur algorithmischen Problemlösung ist und daß alles, was mit Turingmaschinen "nicht geht", überhaupt nicht geht.

Unsere Kette *Endlicher Automat-Kellerautomat-Turingmaschine (Chomsky-Hierarchie)* ist bei der Turingmaschine ganz überraschend und definitiv zu Ende. Was Besseres gibt es nicht und wird es niemals geben (behauptet die These von Church). Alles, was die allerbesten Rechner können und jemals können werden, das können Turingmaschinen auch schon, im Prinzip jedenfalls - falls man genügend Geduld hat, die Turingmaschine zu programmieren und genügend Zeit, um auf das Ergebnis zu warten.

Übungen zu TI, Teil 4

Zur Übung sollten Sie einige der Beispiele aus Teil 1 (endliche Automaten) mittels Turingmaschinen programmieren. Die folgenden Übungen haben teilweise höheren Schwierigkeitsgrad.

(1) Man schreibe ein Turingprogramm zur Erkennung der binären Strings mit ebensovielen Nullen wie Einsen.

(2a) Sei L die Menge der verdoppelten und durch ein Trennzeichen getrennten binären Strings $w\#w$, also beispielsweise

1#1 , 011#011 , 1011100#1011100 , #
 |--w--| |--w--|

Man schreibe ein Turingprogramm für L .

(b) Man skizziere ein Turingprogramm für verdoppelte binäre Strings, also Strings der Art ww ohne Trennzeichen in der Mitte.

(3) Man schreibe ein Turingprogramm, das die Strings unserer von früher bekannten Beispielsprache $L: (01+010)^*$ akzeptiert.

(4) Man skizziere Turingprogramme, die Dateinamen der folgenden Muster erkennen:

- $a^*.txt$ (Dateinamen, die mit "a" beginnen und mit ".txt" enden
- wie (a), nur soll kein zweiter Punkt im Dateinamen vorkommen

(5) Mit Turingmaschinen kann man nicht nur formale Sprachen erkennen, sondern auch Funktionen berechnen.

Man vereinbart: Der Funktionswert ist das, was sich bei Stop der Maschine zwischen Lesekopf (inklusive) und erstem Blank (exklusive) auf dem Band befindet. - Man überlege sich Turingprogramme zur Berechnung der Funktionen

$n \mapsto 2n$, $n=1,2,3,\dots$

$n \mapsto n+1$, $n=1,2,3,\dots$

$n \mapsto n\%4$, $n=1,2,3,\dots$ ("n modulo 4")

(*) $n \mapsto 3n$, $n=1,2,3,\dots$

wobei unterstellt ist, daß die Zahlen in Dualform auf's Band geschrieben sind.