Arbeiten mit Turingmaschinen

Dr. Eva Richter

19. April 2012

- Varianten von Turingmaschine sind in Bezug auf die von ihnen berechneten Sprachen äquivalent
- TM können als Rechenmaschinen betrachtet werden, wenn man den Bandinhalt bei Erreichen eines Haltezustandes als Funktionswert für die Eingabe betrachtet
- nicht jede TM hält auf jeder Eingabe an: d.h. Funktionen sind nicht überall definiert
- in der Berechenbarkeitstheorie sind Funktionen partielle Funktionen

Definition

1. $f : \Sigma^* \to \Gamma^*$ *heißt total berechenbar, wenn es eine TM gibt, die auf jeder Eingabe* $w \in \Sigma^*$ *anhält mit* $f(w)$ *als aktuellem Bandinhalt.*
2. f *heißt partiell berechenbar, wenn es eine Turingmaschine gibt, die auf jeder Eingabe aus dem Definitionsbereich von* f *anhält und den Wert* $f(w)$ *als aktuellen Bandinhalt hat.*

- neben Turingmaschinen gibt es viele weitere Modelle für allgemeine Berechnungen – solche die TM sehr ähnlich sind und solche, die weniger ähnlich sind
- Gemeinsamkeiten, die sie alle teilen
  1. unbeschränkten Zugang
  2. zu unbegrenzt viel Speicher
  3. in einem einzelnen Schritt kann nur ein endliches Pensum von Arbeit erledigt werden.

- informell: ein Algorithmus ist eine Ansammlung von einfachen Anweisungen, um eine bestimmte Aufgabe auszuführen
- Begriff des Algorithmus nicht vor dem 20. Jahrhundert genau definiert
- intuitiver Begriff war nicht ausreichend, um ein tieferes Verständnis von Algorithmen zu bekommen
- heute wissen wir, dass es Funktionen gibt, die nicht berechenbar sind
- zu beweisen, dass man kein mechanisches Verfahren für eine Problemlösung angeben kann, ist unmöglich, wenn man die Klasse der Algorithmen nicht beschreiben kann
- (zum Finden eines Algorithmus braucht man nicht unbedingt mehr als eine gute Idee von der Sache)

- 1900 hielt David Hilbert (1862 – 1942) eine programmatische Rede auf dem Internationalen Mathematikerkongress in Paris
- stellte 23 Problemen vor, die die vordringlichsten Aufgaben des neuen Jahrhunderts sein sollten
- das zehnte Problem auf dieser Liste hat mit Algorithmen zu tun

Polynome und Wurzeln

Definition

Ein Polynom ist eine Summe von Termen, wobei jeder Term ein Produkt bestimmter Variablen und einer Konstanten (Koeffizienten) ist.

$$p = 6x^3yz^2 + 3xy^2 - x^3 - 10$$

ist ein Polynom mit vier Termen in den Variablen x, y und z

Definition

Die Wurzel eines Polynoms ist eine Belegung der Variablen, so dass der Wert des Polynoms 0 ist. Eine Wurzel ist ganzzahlig , wenn die Belegung mit ganzen Zahlen erfolgt

Hilbert wollte einen Algorithmus, der feststellt, ob ein Polynom ganzzahlige Lösungen hat.

- es gibt keinen Algorithmus, der entscheidet, ob ein Polynom ganzzahlige Wurzeln hat
- Erkenntnis war ohne klaren Algorithmusbegriff unmöglich
- Unentscheidbarkeit wurde 1970 durch Juri Matijasevich bewiesen
- Definition des Begriffes Algorithmus wurde 1936 von Alan Turing und Alonzo Church gegeben
- Turing– Turingmaschinen, Church–λ-Kalkül, beide Definitionen sind äquivalent

Intuitive Berechenbarkeit ist äquivalent zu Turing-Berechenbarkeit.

$$D = \{p \mid p \text{ ist ein Polynom mit einer ganzzahligen Wurzel}\}$$

Ist Menge D entscheidbar ?

$D = \{p \mid p \text{ ist ein Polynom mit einer ganzzahligen Wurzel}\}$

Ist Menge D entscheidbar ?Nein, aber D ist Turing-akzeptierbar.

$$D = \{p \mid p \text{ ist ein Polynom mit einer ganzzahligen Wurzel}\}$$

Ist Menge D entscheidbar ?Nein, aber D ist Turing-akzeptierbar.

Sei $D_1 = \{p \mid p \text{ ist ein Polynom in } x \text{ mit einer ganzzahligen Wurzel}\}$.

M_1 =„Bei Eingabe eines Polynoms $< p >$ in x

1. belege x der Reihe nach mit den Werten
   $0, -1, 1, -2, 2, -3, \ldots$ und werte P aus
   falls an einer Stelle das Polynom 0 ergibt, **accept**."

1. Falls p keine ganzzahlige Wurzel hat, wird die Maschine niemals anhalten, aber wenn es eine Wurzel hat, dann wird sie sie finden.
2. ähnliche Turingmaschine M für Polynome in mehreren Variablen
3. M und M_1 sind keine Entscheider, aber M_1 kann in einen Entscheider umgewandelt werden, da wir Schranken berechnen können, innerhalb derer die Wurzeln eines Polynoms in einer Variablen liegen müssen
4. Matijasevich beweist, dass das Berechnen solcher Schranken für allgemeine Polynome unmöglich ist

Unser Hauptaugenmerk liegt auf den durch sie zu beschreibenden Algorithmen.

1. Die *formale Beschreibung:* Darstellung der Zustände, Überführungsfunktion usw.
2. *Implementierungsbeschreibung:* in Umgangssprache wird beschrieben, wie sich der Kopf bewegt und in welcher Weise die Daten auf dem Band gespeichert werden
3. *High-level Beschreibung:* nur der Algorithmus, keine Angaben über Bewegung der Köpfe und Bänder

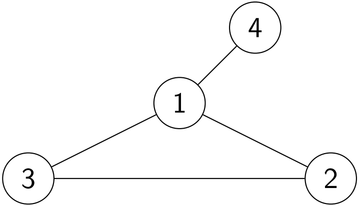
- Eingaben sind immer Zeichenketten
- andere Objekte können als String dargestellt werden, eine TM kann programmiert werden, diese Darstellung zu dekodieren
  O für das Objekt, $< O >$ für dessen Kodierung als Zeichenkette
- mehrere Objekte $O_1, \dots O_n$ werden als eine Zeichenkette $< O_1, \dots O_n >$ dargestellt
- Art der Kodierung spielt keine Rolle, jede Kodierung kann(von einer TM!) in eine jede andere übersetzen werden.

- Turingmaschinen-Algorithmus innerhalb von Anführungszeichen,
- Numerierung erfolgt nach Stadien
- Blockstruktur wird durch Einrücken erkennbar gemacht
- erste Zeile $=$ Eingabe(Zeichenkette); falls die Eingabe die Kodierung eines Objektes ist, z.B. $< A >$ wird im ersten Schritt geprüft, ob die Kodierung korrekt ist, falls nicht, wird die Eingabe abgelehnt.

Verbundene Graphen

Definition

Ein Graph heißt verbunden, wenn jeder Knoten von jedem anderen aus durch Verfolgen der Kanten erreichbar ist.



- Graphen werden durch zwei Listen dargestellt
- Knotenliste ohne Wiederholungen
- Kantenliste aus Paaren von Knoten
- $< G >= (1, 2, 3, 4), ((1, 2), (2, 3), (3, 1), (1, 4))$

$M =$„bei Eingabe von $< G >$

1. wähle den ersten Knoten und markiere ihn,
2. wiederhole folgende Anweisungen bis keine neuen Knoten mehr markiert werden,
3. gehe alle Knoten in G durch und markiere sie, wenn sie von irgendeinem bereits markierten Knoten erreichbar sind,
4. überprüfe, ob alle Knoten von G markiert sind, falls ja akzeptiere, falls nein lehne ab.

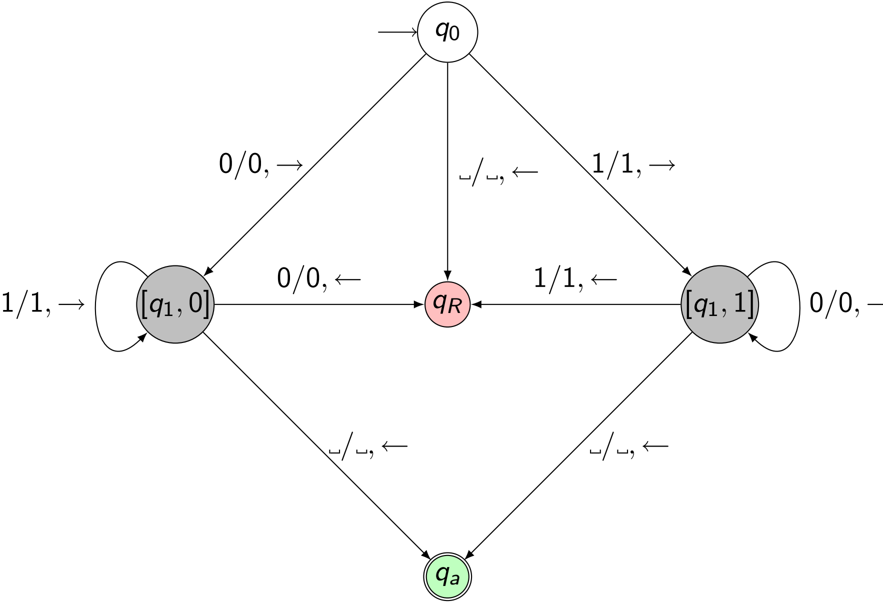
- Zustandsmenge kann verwendet werden um (eine endliche Menge von) Daten zu speichern
- Zustände werden mit Paaren bezeichnet, erster Teil stellt Position im Programmablauf dar, der zweite dient als Speicher

$$M = (\{0,1\}, \{0,1,\llcorner\!\lrcorner\}, \{q_0, q_a, q_r, [q_1,0], [q_1,1]\}\delta, q_0, q_a, q_r)$$

ist TM für $L(10^* + 01^*)$

δ	0	1	$\llcorner\!\lrcorner$
q_0	$([q_1,0],R)$	$([q_1,1],R)$	q_r
$[q_1,0]$	(q_r,R)	$([q_1,0],R)$	(q_a,R)
$[q_1,1]$	$([q_1,1],R)$	(q_r,R)	(q_a,R)
q_a	-	-	-
q_r	-	-	-

TM für die Sprache $L(10^* + 01^*)$

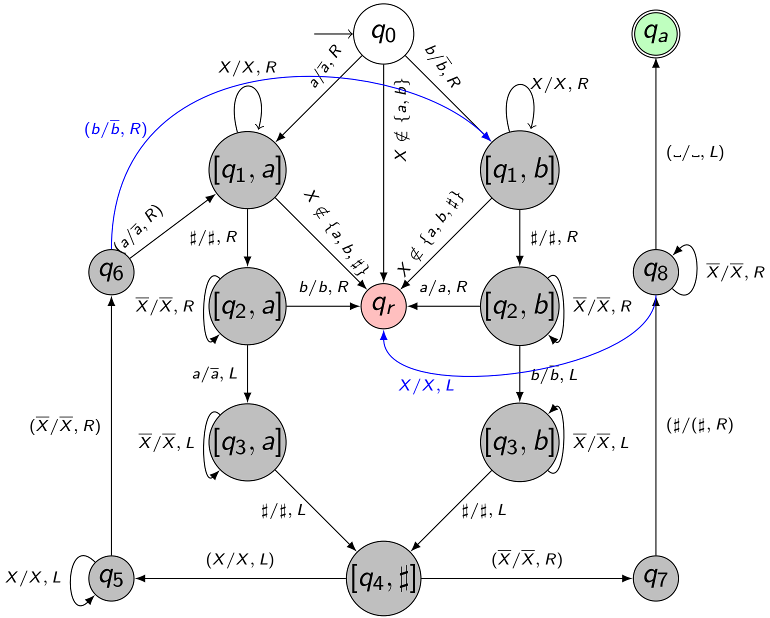


- Bei Sprachen, die sich durch Wiederholung von Strings auszeichnen, ist es manchmal notwendig „mitzuzählen“
- Bandalphabet Γ wird um zusätzliche Symbole erweitert; für jedes Σ-Zeichen a wird $\bar{a}$ hinzufügt. $\Gamma = \Sigma \cup \{\bar{a} \mid a \in \Sigma\}$
- einige Zustände werden als Paare beschrieben, wobei der erste Teil den Stand der Abarbeitung im Algorithmus kodiert und der zweite Teil als Speicherplatz für ein eingelesenes Symbol dient.

Turingmaschine für $L = \{w\sharp w \mid w \in L(\{0,1\}^+)\}$

- q_0 erstes Zeichen X wird gelesen und markiert, Übergang nach (q_1, X), bei ungültiger Eingabe– Übergang zu q_r.
- (q_1, X) Kopf nach rechts bewegen bis $\sharp$ – Übergang zu (q_2, X); falls kein $\sharp$ vorhanden –Übergang zu q_r
- (q_2, X) nach rechts bis zum ersten unmarkiertes Zeichen hinter $\sharp$, vergleichen mit dem gespeicherten Zeichen X, bei Übereinstimmung nach (q_3, X), sonst zu q_r.
- (q_3, X) nach links gehen bis zum $\sharp$, Wechsel in Zustand $(q_4, \sharp)$
- $(q_4, \sharp)$ falls es keine unmarkierten Zeichen links von $\sharp$ gibt, gehe in q_7 über; falls es noch welche gibt nach q_5
- q_5 zum „linkesten" unmarkierten Zeichen–Übergang nach q_6
- q_6 X lesen und markieren– Übergang nach (q_1, X).
- q_7 $\sharp$ lesen, nach rechts gehen – Übergang nach q_8
- q_8 nach rechts gehen, alle markierten Zeichen lesen, wenn noch unmarkierte Zeichen da sind, Übergang zu q_r sonst zu q_a

TM für $L = \{w\sharp w \mid w \in L(\{0,1\}^+)\}$

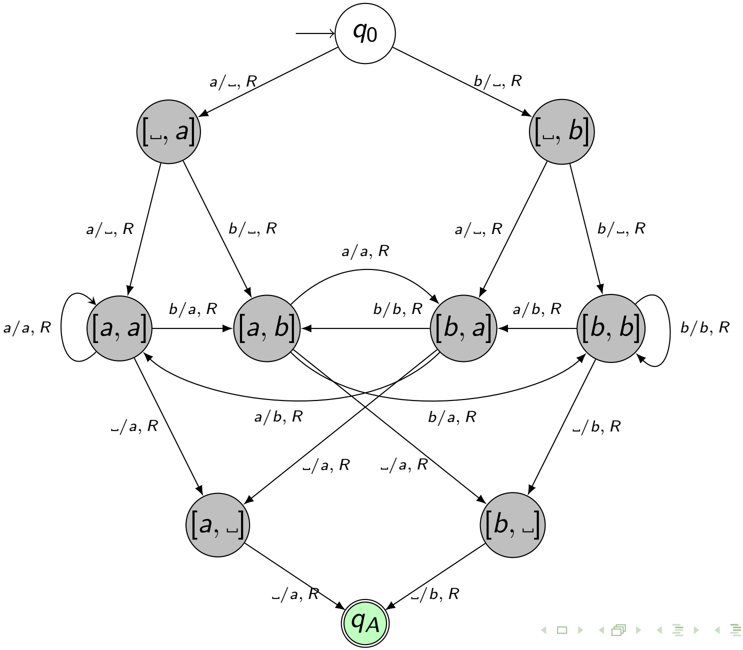


- um Platz auf dem Band zu schaffen, kann man Bandinhalt um eine feste Anzahl x nach rechts verschieben
- mit Hilfe der Zustände legt man einen Zwischenspeicher an
- Zeichen werden auf einer Seite hinein-, auf der anderen Seite herausgeschoben
- wir setzen voraus, dass keine Leerzeichen innerhalb des zu verschiebenden Teils erlaubt sind.

Für $\Sigma = \{a, b\}$ und $x = 2$ brauchen wir

$$Q = \{q_0, [\textvisiblespace, a], [\textvisiblespace, b], [a, a], [a, b][b, a], [b, b][a, \textvisiblespace], [b, \textvisiblespace][q_A], [q_R]\}$$

Bandinhalt um zwei Zeichen nach rechts verschieben

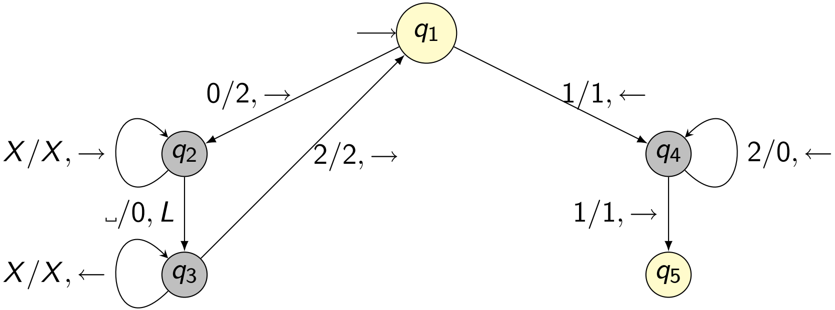


- Weiterverwendung von bereits programmierten Teilen
- TM M_1 wird als Teil in andere TM M_2 eingebaut
- vorher: gewünschten Anfangszustand herstellen
- hinterher: Haltezustände müssen zu einem vorher festgelegten Rückkehrzustand führen
- Aufrufe können dann sowohl rekursiv als auch nicht rekursiv erfolgen

Zahl m durch den String 0^m und das Paar (m, n) durch 0^m10^n dargestellt.
$M =$„Auf Eingabe von 0^m10^n

1. hinter das rechte Ende wird eine Eins gesetzt
2. für jede der Nullen aus dem vorderen Block wird der hintere Block aus Nullen einmal kopiert und die vordere Null gelöscht
3. wenn keine vordere Null mehr vorhanden ist, steht das Ergebnis hinter der zweiten Eins, die Maschine akzeptiert."

startet mit $0^m 1 q_1 0^n 1 0^i$ und endet mit $0^m 1 q_5 0^n 1 0^{i+n}$.



Ganzes Programm für Multiplikation