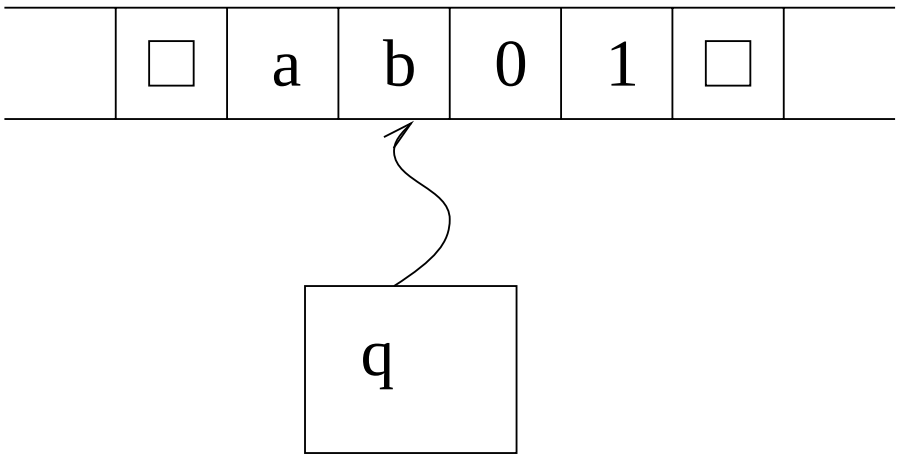
Turingmaschine



Das Band ist unendlich und kann gelesen und beschrieben werden, der Schreiblesekopf kann in jedem Schritt um ein Feld nach rechts oder links versetzt werden oder stehen bleiben.
Das Eingabealphabet ist eine Teilmenge des Arbeitsalphabets.

Turingmaschine, formal

Definition

Eine *Turingmaschine* ist ein 7-Tupel
$M = (K, \Sigma, \Gamma, \delta, q_0, \square, F)$ mit:

K die endliche Menge der Zustände,

kontextsensitive Sprachen

Eine *nichtdeterministische linear beschränkte Turingmaschine* (*linear bounded Automaton, LBA*) ist eine Turingmaschine, die sich nicht über den Teil des Bandes, auf dem die Eingabe steht, hinausbewegt (dazu muss das letzte Zeichen der Eingabe besonders markiert werden):

Für alle $a_1a_2...a_{n-1}a_n \in \Sigma^*$ und alle Situationen $\alpha q\beta$ mit $q_0a_1a_2...a_{n-1}a'_n \vdash^* \alpha q\beta$ gilt: $\mid \alpha\beta \mid = n$.

(a'_n ist das markierte letzte Zeichen a_n)

Die von nichtdeterministischen linear beschränkten TMs akzeptierten Sprachen sind (maximal) kontextsensitiv.

Die Frage, ob LBAs und deterministische LBAs äquivalent sind, ist noch unentschieden!

Typ-1-Sprachen sind unter Schnitt, Vereinigung, Komplement, Konkatenation und Stern-Operation abgeschlossen.

rekursiv aufzählbare Sprachen

Allgemeine Turingmaschinen akzeptieren Typ-0-Sprachen, auch *rekursiv aufzählbar* genannt.

Nichtdeterministische Turingmaschinen können durch deterministische simuliert werden (durchsuche nach einem festen Schema den Berechnungsbaum...), die beiden Automatensorten sind äquivalent.

- **Entscheidbar (decidable):** Eine Menge heisst entscheidbar, wenn ihre charakteristische Funktion berechenbar ist.
- **Semi-entscheidbar (semi-decidable):** Eine Menge heisst semi-entscheidbar, wenn ihre charakteristische Funktion für ihre Elemente berechenbar ist. Für andere Objekte kann diese Funktion undefiniert sein.

Eine Sprache ist semi-entscheidbar, wenn sie rekursiv aufzählbar ist.

Das Wortproblem (ist ein gegebenes Wort Element der Sprache?) ist für diese Sprachen nicht entscheidbar (da die Turingmaschine evt. nicht anhält).

Eine Sprache, die selbst Turing-akzeptierbar ist und deren Komplement ebenfalls, ist *entscheidbar*.

Spezielles Halteproblem

Jede Turingmaschine lässt sich als Wort über $\{0, 1, \#\}$ kodieren:

L, R, N seien 0, 1, bzw. 2; Durchnumerierung der Elemente der Alphabete, des Startzustands und des Leerzeichens (beginnend bei 3), dann Binärdarstellung davon.

Regeln der Form $\delta(q_i, a_j) \rightarrow (q_{i'}, a_{j'}, y)$ als Wort aus den Binärdarstellungen der Symbole, getrennt von #, beginnend mit ##.

Diese Codierung lässt sich wiederum auf ein Wort über $\{0, 1\}$ abbilden (wie?).

Sei M' eine beliebige feste Turingmaschine.

$$M_w = \begin{cases} M, \text{ falls } w \text{ eine TM codiert} \\ M', \text{ sonst} \end{cases}$$

Dann heisst die Sprache K mit

$$K = \{w \in \{0, 1\}^* \mid M_w \text{ angesetzt auf } w \text{ hält.}\}$$

spezielles Halteproblem oder *Selbstanwendbarkeitsproblem*.

Dieses Problem ist nicht entscheidbar!

Die Turingmaschine U, die sich auf der Eingabe $w\#x$ so verhält wie die TM M_w auf x, heisst *Universelle Turingmaschine*.

Berechenbarkeitsbegriff

Eine (partielle) Funktion ist intuitiv berechenbar, wenn es eine Rechenvorschrift gibt, die auf den Argumenten, auf denen die Funktion definiert ist, nach endlich vielen Schritten den Funktionswert liefert.

So eine Rechenvorschrift heisst *Algorithmus*: endlich beschreibbar, mechanisch ausführbar, deterministisch, endet auf definierten Eingaben.

Formale Definitionen der Berechenbarkeit: Turingmaschine, μ-Rekursivität, WHILE-Programme, GOTO-Programme, ...

Church'sche These: Diese Formalisierungen erfassen genau den intuitiven Berechenbarkeitsbegriff.

Berechenbarkeit

Sind folgende Beispiele berechenbar?

$f, g, h : N \to \{0, 1\}$

$f(n) = 1$, falls n ein Anfangstück der Dezimalbruchentwicklung von π ist, 0 sonst.

(ja)

$g(n) = 1$, falls n irgendwo in der Dezimalbruchentwicklung von π vorkommt, 0 sonst.

(evt. nein, möglicherweise ist π aber soo zufällig, dass jede Ziffernfolge darin vorkommt. Dann ist $g(n) = 1$.)

$h(n) = 1$, falls n mal 7 in der Dezimalbruchentwicklung von π vorkommt, 0 sonst.

(ja. Entweder gibt es beliebig lange 7er-Folgen, oder es gibt dafür eine obere Grenze...)

O-Notation

Mit der O-Notation kann der Berechnungsaufwand für einen Algorithmus unabhängig von einer bestimmten Programmiersprache oder Rechnerarchitektur angegeben werden.

Angegeben wird eine obere Schranke, konstante Faktoren werden ignoriert.

Für eine Funktion $f : N \to N$ ist definiert:

$O(f) = \{g : N \to N \mid$ es gibt c und n_0, so dass für alle $n > n_0$ gilt: $g(n) \leq c f(n)\}$

Beispiel:

$$O(3\,n^4 + 34\,n \log n - 17) = O(n^4)$$

Komplexitätstheorie

In der Komplexitätstheorie wird versucht, den Berechnungsaufwand für die Lösung von Problemen (i.e. das Wortproblem für formale Sprachen) abzuschätzen.

Eine *obere* Grenze lässt sich durch die Angabe eines Algorithmus bestimmen, eine untere Grenze ist immer n, die Länge der Eingabe.

Probleme lassen sich in Komplexitätsklassen einordnen, neue Probleme können durch *Reduktion* auf bekannte ebenfalls klassifiziert werden.

Komplexitätsklassen

- **P (tractable):** Ein Problem kann mit polynomialem Zeitaufwand bzgl. der Länge der Eingabe gelöst werden.
- **NP-vollständig (NP-complete):** Ein Problem ist *NP-vollständig*, wenn es (noch?) nicht mit polynomialem Zeitaufwand gelöst werden kann.
- **NP-hart (NP-hard):** Ein Problem ist *NP-hart*, wenn es mindestens das Zeitverhalten wie ein NP-vollständiges Problem hat.
- **co-NP-hart (co-NP-hard):** Ein Problem ist *co-NP-hart*, falls das komplementäre Problem NP-hart ist.
- **PSPACE-vollständig (PSPACE-complete):** Ein Problem ist *PSPACE-vollständig*, falls es NP-vollständig ist und polynomialen Speicheraufwand hat.

$P \subseteq NP$

$P = NP$?

Die meisten NP-vollständigen Probleme hängen so zusammen, dass entweder jedes oder keines von ihnen doch polynomial gelöst werden kann.

Reduzierbarkeit

Das Problem **3SAT**:

Gegeben: eine aussagenlogische Formel F in konjunktiver Normalform mit höchstens 3 Literalen pro Klausel. Ist F erfüllbar?

Das Problem **VertexCover**:

Gegeben: ein endlicher Graph G und eine Zahl k. Gibt es eine Teilmenge der Knoten von G mit maximal k Elementen, so dass mindestens ein Endpunt jeder Kante von G darin enthalten ist?

3SAT und **VertexCover** sind als NP-vollständig bekannt.

Durch Reduktion auf **3SAT** oder **VertexCover** lassen sich auch andere Probleme als NP-vollständig charakterisieren...