5.2 Linear beschränkte Automaten

Definition 98

Eine Turingmaschine heißt linear beschränkt (kurz: LBA), falls für alle $q \in Q$ gilt:

$$(q', c, d) \in \delta(q, \square) \qquad \Longrightarrow \qquad c = \square\,.$$

Ein Leerzeichen wird also nie durch ein anderes Zeichen überschrieben. Mit anderen Worten: Die Turingmaschine darf ausschliesslich die Positionen beschreiben, an denen zu Beginn die Eingabe x steht.

Satz 99

Die von linear beschränkten, nichtdeterministischen Turingmaschinen akzeptierten Sprachen sind genau die kontextsensitiven (also Chomsky-1) Sprachen.

Beweis:

Wir beschreiben nur die Beweisidee.

„$\Longrightarrow$“: Wir benutzen eine Menge von Nichtterminalsymbolen, die $Q \times \Sigma$ enthält, für die Grammatik. Die Grammatik erzeugt zunächst alle akzeptierenden Konfigurationen (der Form $\alpha q_f \beta$ mit $q_f \in F$), wobei q_f und β_1 zusammen als ein Zeichen codiert sind. Sie enthält weiterhin Regeln, die es gestatten, aus jeder Satzform, die eine Konfiguration darstellt, alle möglichen unmittelbaren Vorgängerkonfigurationen abzuleiten. Die zur Anfangskonfiguration (ϵ, q_0, w) gehörige Satzform ist damit ableitbar gdw der LBA das Wort w akzeptiert.

Satz 99

Die von linear beschränkten, nichtdeterministischen Turingmaschinen akzeptierten Sprachen sind genau die kontextsensitiven (also Chomsky-1) Sprachen.

Beweis:

Wir beschreiben nur die Beweisidee.

„$\Longleftarrow$“: Wir simulieren mittels des LBA nichtdeterministisch und in Rückwärtsrichtung die möglichen Ableitungen der kontextsensitiven Grammatik und prüfen, ob wir die Satzform S erreichen können. Da die Grammatik längenmonoton ist, nimmt der vom LBA benötigte Platz nie zu. □

Beispiel 100

Die Sprache $L = \{a^m b^m c^m \mid m \in \mathbb{N}_0\}$ ist kontextsensitiv.

Satz 101

Das Wortproblem für LBAs bzw. für Chomsky-1-Grammatiken ist entscheidbar.

Beweis:

Siehe z.B. die Konstruktion zum vorhergehenden Satz bzw. Übungsaufgabe. □

Satz 102

Die Familie CSL der kontextsensitiven (bzw. Chomsky-1) Sprachen ist abgeschlossen unter den folgenden Operationen:

$$\cap\,,\ \cup\,,\ \cdot\,,\ ^*\,,\ ^-$$

Beweis:

Der Beweis für die ersten vier Operationen ergibt sich unmittelbar aus den Eigenschaften linear beschränkter Automaten, der Abschluss unter Komplement wird später gezeigt („Induktives Zählen“). □

Satz 103

Folgende Probleme sind für die Familie der Chomsky-1-Sprachen nicht entscheidbar:

1. *Leerheit*
2. *Äquivalenz*
3. *Durchschnitt*

Beweis:

ohne Beweis. □

5.3 Chomsky-0-Sprachen

Satz 104

Zu jeder (nichtdeterministischen) TM N gibt es eine deterministische TM (DTM) M mit

$$L(N) = L(M)\,.$$

Beweis:

Die DTM erzeugt in BFS-Manier, für $k = 0, 1, \ldots$, alle Konfigurationen, die die TM N in k Schritten erreichen kann. Sie hält gdw sich dabei eine akzeptierende Konfiguration ergibt. □

Satz 105

Die von (nichtdeterministischen oder deterministischen) Turingmaschinen akzeptierten Sprachen sind genau die Chomsky-0-Sprachen.

Beweis:

Wir beschreiben nur die Beweisidee.

„$\Longrightarrow$“: Die Grammatik erzeugt zunächst alle akzeptierenden Konfigurationen (der Form $\alpha q_f \beta$ mit $q_f \in F$). Sie enthält weiterhin Regeln, die es gestatten, aus jeder Satzform, die eine Konfiguration darstellt, alle möglichen unmittelbaren Vorgängerkonfigurationen abzuleiten. Die zur Anfangskonfiguration (ϵ, q_0, w) gehörige Satzform ist damit ableitbar gdw die TM das Wort w akzeptiert. Die Produktionen ergeben sich kanonisch aus der Übergangsrelation der TM. Sie sind i.a. nicht mehr längenmonoton!

Satz 105

Die von (nichtdeterministischen oder deterministischen) Turingmaschinen akzeptierten Sprachen sind genau die Chomsky-0-Sprachen.

Beweis:

Wir beschreiben nur die Beweisidee.

„$\Longleftarrow$“: Wir simulieren mit der TM alle Ableitungen der Chomsky-0-Grammatik in BFS-Manier und akzeptieren, falls eine solche Ableitung das Eingabewort x ergibt.

Man beachte, dass die konstruierte TM nicht unbedingt immer hält!

Eine andere Idee ist, wie im LBA-Fall die Ableitungen rückwärts zu simulieren. □

6. Übersicht Chomsky-Hierarchie

6.1 Die Chomsky-Hierarchie

Typ 3	reguläre Grammatik DFA NFA regulärer Ausdruck
DCFL	$LR(k)$-Grammatik deterministischer Kellerautomat
Typ 2	kontextfreie Grammatik (nichtdeterministischer) Kellerautomat
Typ 1	kontextsensitive Grammatik (nichtdet.) linear beschränkter Automat (LBA)
Typ 0	Chomsky-Grammatik, Phrasenstrukturgrammatik det./nichtdet. Turingmaschine

6.2 Wortproblem

Typ 3, gegeben als DFA	lineare Laufzeit
DCFL, gegeben als DPDA	lineare Laufzeit
Typ 2, CNF-Grammatik	CYK-Algorithmus, Laufzeit $O(n^3)$
Typ 1	exponentiell
Typ 0	—

6.3 Abschlusseigenschaften

	Schnitt	Vereinigung	Komplement	Produkt	Stern
Typ 3	ja	ja	ja	ja	ja
DCFL	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

6.4 Entscheidbarkeit

	Wortproblem	Leerheit	Äquivalenz	Schnittproblem
Typ 3	ja	ja	ja	ja
DCFL	ja	ja	ja	nein
Typ 2	ja	ja	nein	nein
Typ 1	ja	nein	nein	nein
Typ 0	nein	nein	nein	nein

Kapitel II Berechenbarkeit, Entscheidbarkeit

1. Der Begriff der Berechenbarkeit

Unsere Vorstellung ist:
$f : \mathbb{N}_0^k \to \mathbb{N}_0$ ist berechenbar, wenn es einen Algorithmus gibt, der f berechnet, und genauer, der bei Eingabe $(n_1, \ldots, n_k) \in \mathbb{N}_0^k$ nach endlich vielen Schritten mit dem Ergebnis $f(n_1, \ldots, n_k) \in \mathbb{N}_0$ hält.

Was bedeutet „Algorithmus“ an dieser Stelle?

AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, ...-Programme?
Gibt es einen Unterschied, wenn man sich auf eine bestimmte Programmiersprache beschränkt?

Analog für partielle Funktionen

$$f : \mathbb{N}_0^k \supseteq D \to \mathbb{N}_0$$

bedeutet berechenbar Folgendes:

1. Algorithmus soll mit dem richtigen Ergebnis stoppen, wenn $(n_1, \ldots, n_k) \in D$
2. und nicht stoppen, wenn $(n_1, \ldots, n_k) \notin D$.

Beispiel 106

Wir definieren folgende Funktionen:

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ interpretiert als Ziffernfolge in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

$$f_3(n) = \begin{cases} 1 & \text{falls mindestens } n \text{ aufeinanderfolgende Ziffern} \\ & \text{in } \pi \text{ gleich 1 sind} \\ 0 & \text{sonst} \end{cases}$$

$\pi = 3,14159265358979323846264338327950288419716939937\ldots$

Einige Beispiele sind damit:

$$f_1(314) = 1, f_1(415) = 0, f_2(415) = 1\ .$$

Beispiel 106

$$f_1(n) = \begin{cases} 1 & \text{falls } n \text{ als Ziffernfolge Anfangsstück von } \pi \text{ ist} \\ 0 & \text{sonst} \end{cases}$$

Wie man leicht einsieht, ist f_1 berechenbar, denn um festzustellen, ob eine Ziffernfolge ein Anfangsstück von π ist, muss π nur auf entsprechend viele Dezimalstellen berechnet werden.

Beispiel 106

$$f_2(n) = \begin{cases} 1 & \text{falls } n \text{ interpretiert als Ziffernfolge in } \pi \text{ vorkommt} \\ 0 & \text{sonst} \end{cases}$$

Für f_2 wissen wir nicht, ob es berechenbar ist. Um festzustellen, dass die Ziffernfolge in π vorkommt, müsste man π schrittweise immer genauer approximieren. Der Algorithmus würde stoppen, wenn die Ziffernfolge gefunden wird. Aber was ist, wenn die Ziffernfolge in π nicht vorkommt?
Vielleicht gibt es aber einen (noch zu findenden) mathematischen Satz, der genaue Aussagen über die in π vorkommenden Ziffernfolgen macht.
Wäre π vollkommen zufällig, was es aber nicht ist, dann würde jedes n als Ziffernfolge irgendwann vorkommen.

Beispiel 106

$$f_3(n) = \begin{cases} 1 & \text{falls mindestens } n \text{ aufeinanderfolgende Ziffern} \\ & \text{in } \pi \text{ gleich 1 sind} \\ 0 & \text{sonst} \end{cases}$$

f_3 ist berechenbar, denn $f_3 \equiv f_4$, mit

$$f_4(n) = \begin{cases} 1 & n \leq n_0 \\ 0 & \text{sonst} \end{cases}$$

wobei n_0 die maximale Anzahl von aufeinanderfolgenden 1en in π ist. Hierbei ist es nicht von Bedeutung, wie die Zahl n_0 berechnet werden kann - wichtig ist nur, dass eine solche Zahl $n_0 \in \mathbb{N}$ existiert.

Weitere Vorschläge, den Begriff der Berechenbarkeit zu präzisieren und zu formalisieren:

1. Turing-Berechenbarkeit
2. Markov-Algorithmen
3. λ-Kalkül
4. μ-rekursive Funktionen
5. Registermaschinen
6. AWK, B, C, Euler, Fortran, Id, JAVA, Lisp, Modula, Oberon, Pascal, Simula, . . .-Programme
7. while-Programme
8. goto-Programme
9. DNA-Algorithmen
10. Quantenalgorithmen
11. u.v.a.m.

Es wurde bewiesen: Alle diese Beschreibungsmethoden sind in ihrer Mächtigkeit äquivalent.

Church'sche These
Dieser formale Begriff der Berechenbarkeit stimmt mit dem intuitiven überein.

Definition 107

Eine (partielle) Funktion

$$f : \mathbb{N}_0^k \supseteq D \to \mathbb{N}_0$$

heißt Turing-berechenbar, falls es eine deterministische Turingmaschine gibt, die für jede Eingabe $(n_1, \ldots, n_k) \in D$ nach endlich vielen Schritten mit dem Bandinhalt

$$f(n_1, \ldots, n_k) \in \mathbb{N}_0$$

hält. Falls $(n_1, \ldots, n_k) \notin D$, hält die Turingmaschine nicht!

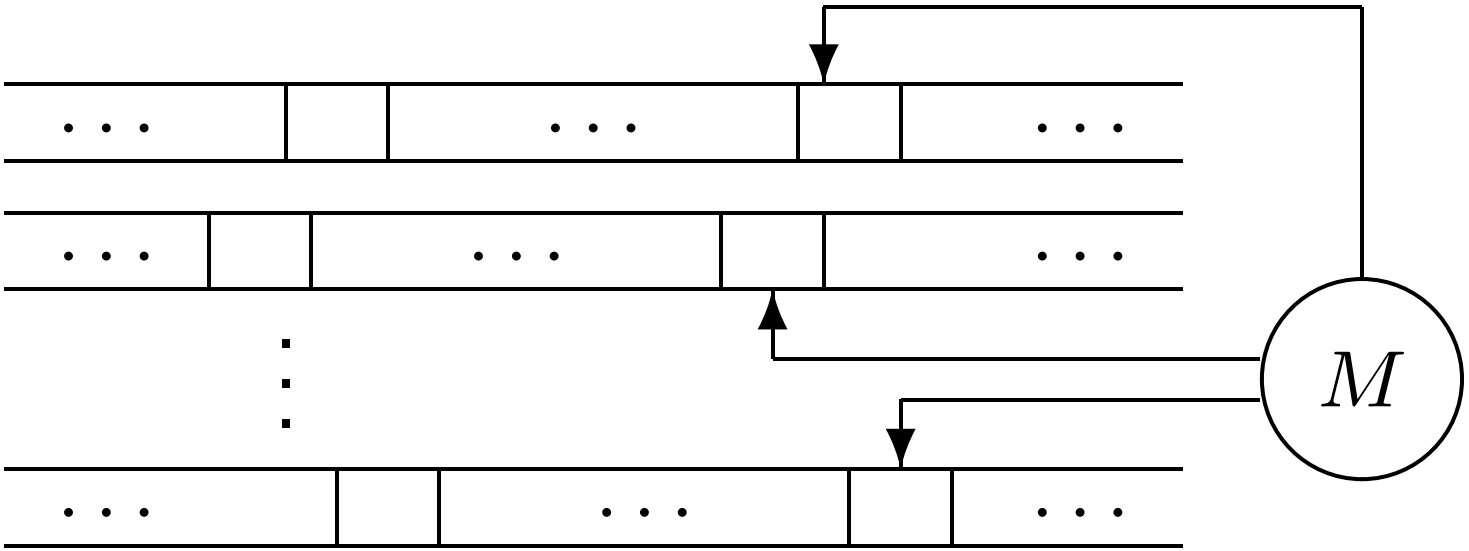
Dabei nehmen wir an, dass Tupel wie $(n_1, \ldots, n_k)$ geeignet codiert auf dem Band der Turingmaschine dargestellt werden.

Eine beliebte Modellvariante ist die k-Band-Turingmaschine, die statt einem Band k, $k \geq 1$, Arbeitsbänder zur Verfügung hat, deren Lese-/Schreibköpfe sie unabhängig voneinander bewegen kann.

Oft existiert auch ein spezielles Eingabeband, das nur gelesen, aber nicht geschrieben werden kann (read-only). Der Lesekopf kann jedoch normalerweise in beiden Richtungen bewegt werden.

Ebenso wird oft ein spezielles Ausgabeband verwendet, das nur geschrieben, aber nicht gelesen werden kann (write-only). Der Schreibkopf kann dabei nur nach rechts bewegt werden.

Beispiel 108 (k-Band-Turingmaschine)



Satz 109

Jede k-Band-Turingmaschine kann effektiv durch eine 1-Band-TM simuliert werden.

Beweis:

Wir simulieren die k Bänder auf k Spuren eines Bandes, wobei wir das Teilalphabet für jede Spur auch noch so erweitern, dass die Kopfposition des simulierten Bandes mit gespeichert werden kann. □

Definition 110

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv oder entscheidbar, falls es eine deterministische TM M gibt, die auf allen Eingaben $\in \Sigma^*$ hält und A erkennt.

Definition 111

Eine Sprache $A \subseteq \Sigma^*$ heißt rekursiv aufzählbar (r.a.) oder semi-entscheidbar, falls es eine TM N gibt, für die

$$L(N) = A\,,$$

also, falls A Chomsky-0 ist.

Definition 112

Sei $A \subseteq \Sigma^*$. Die charakteristische Funktion χ_A von A ist

$$\chi_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ 0 & \text{sonst} \end{cases}$$

Definition 113

Sei $A \subseteq \Sigma^*$. χ'_A ist definiert durch

$$\chi'_A(w) = \begin{cases} 1 & \text{falls } w \in A \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Satz 114

A ist rekursiv $\Leftrightarrow$ χ_A ist berechenbar.

Beweis:

Folgt aus der Definition von rekursiv: es gibt eine TM, die `ja` oder `nein` liefert. Wandle das Ergebnis in 1 oder 0. □

Satz 115

A ist rekursiv aufzählbar $\Leftrightarrow$ χ'_A ist berechenbar.

Beweis:

Folgt unmittelbar aus der Definition. □

Satz 116

A ist rekursiv $\Leftrightarrow$ χ'_A und $\chi'_{\bar{A}}$ sind berechenbar ($\Leftrightarrow$ A und $\bar{A}$ sind r.a.)

Beweis:

Nur $\Leftarrow$ ist nichttrivial. Wir lassen hier eine TM für A und eine TM für $\bar{A}$ Schritt für Schritt parallel laufen. □