Kontextsensitive und Typ 0 Sprachen

Hans U. Simon (RUB)

mit Modifikationen von

Maike Buchin (RUB)

Lehrstuhl Mathematik und Informatik

Homepage: http://www.ruhr–uni–bochum.de/lmi

Kuroda Normalform

Erinnerung: Die Regeln einer kontextsensitiven Grammatik haben die Form

$$\alpha \to \beta \text{ mit } \alpha, \beta \in (V \cup \Sigma)^+ \text{ und } |\alpha| \leq |\beta|$$

d.h., die rechte Seite einer Regel darf nicht kürzer als die linke sein (abgesehen von ein paar zusätzlichen Vereinbarungen das leere Wort betreffend).

Definition: Eine kontextsensitive Grammatik ist in Kuroda Normalform, wenn sie nur Regeln der Form

$$A \to a \ , \quad A \to BC \ , \ AB \to CD \text{ mit } A, B, C, D \in V, a \in \Sigma$$

besitzt.

Anmerkung: Manchmal werden auch Regeln der Form $A \to B$ zugelassen.

Transformation in Kuroda Normalform

Satz: Jede kontextsensitive Grammatik G mit $\varepsilon \notin L(G)$ kann in eine äquivalente kontextsensitive Grammatik G' in Kuroda Normalform transformiert werden.

Analog zur Transformation einer kontextfreien Grammatik in Chomsky Normalform erfolgt der Beweis in mehreren Phasen, die folgendes erreichen:

1. Nur Regeln der Form $A \to a$ dürfen ein Zeichen aus Σ verwenden. (D.h. Regeln einer anderen Form haben auf der linken und rechten Seite jeweils einen String über V stehen.) Die Grammatik heißt dann separiert.
2. Die rechte Seite einer Regel hat maximal die Länge 2.
3. Es gibt keine Kettenregeln (der Form $A \to B$) mehr.

Danach ist die Grammatik in Kuroda Normalform.

Transformation in Kuroda Normalform (fortgesetzt)

Dir folgenden Ziele können ähnlich wie bei der Transformation einer kontext-freien Grammatik in Chomsky Normalform erreicht werden:

- Separiertheit der Grammatik (Ziel von Phase 1)
- Verkürzung zu langer rechter Seiten, wenn die linke Seite nur aus einer Variablen besteht (Teilziel von Phase 2)
- Elimination von Kettenregeln (Ziel von Phase 3)

Der wesentlich neue Punkt ist also die Verkürzung rechter Seiten, wenn die linke Seite eine Länge $m \geq 2$ und die rechte Seite eine Länge $n \geq \max\{3, m\}$ besitzt.

Transformation in Kuroda Normalform (fortgesetzt)

Zur Transformation einer Regel R der Form

$$A_1 \cdots A_m \to B_1 \cdots B_n \ , \ 2 \leq m \leq n \ , \ n \geq 3$$

in Kuroda Normalform verwende folgendes Schema (mit neuen Variablen $R_1, \ldots, R_{n-2}$):

$A_1 A_2$	$\to$	$B_1 R_1$	R_{m-1}	$\to$	$B_m R_m$
$R_1 A_3$	$\to$	$B_2 R_2$	R_m	$\to$	$B_{m+1} R_{m+1}$
	.			.	
	.			.	
	.			.	
$R_{m-2} A_m$	$\to$	$B_{m-1} R_{m-1}$	R_{n-2}	$\to$	$B_{n-1} B_n$

Hierbei ist der Fall $m = n$ so zu verstehen, dass die rechte Spalte von Regeln entfällt und wir in der linken Spalte R_{m-1} durch B_m substituieren.

Beispiel

Die uns bekannte kontextsensitive Grammatik mit den Regeln

$$S \to aSBC,\ S \to aBC,\ CB \to BC,$$
$$aB \to ab,\ bB \to bb,\ bC \to bc,\ cC \to cc$$

für die Sprache $\{a^n b^n c^n |\ n \geq 1\}$ lässt sich umformen durch

1. **Separieren**

$$S \to X_a SBC,\ S \to X_a BC,\ CB \to BC,$$
$$X_a B \to X_a X_b,\ X_b B \to X_b X_b,\ X_b C \to X_b X_c,\ X_c C \to X_c X_c$$
$$X_a \to a, X_b \to b, X_c \to c$$

2. **Verkürzen** der ersten zwei Regeln

$$S \to X_a D,\ D \to SE,\ E \to BC,\ S \to X_a E$$

3. **Kettenregeln** sind nicht vorhanden.

Maschinenmodell für Typ 0 und Typ 1 Sprachen

Als nächstes wollen wir ein Maschinenmodell für kontextsensitive und Typ 0 Sprachen sehen. Die wesentliche Einschränkung von Kellerautomaten ist, dass sie immer nur auf das oberste Element im Keller zugreifen können.

Das Maschinenmodell für Typ 0 und Typ 1 Sprachen hat hingegen einen dynamischen Speicher mit freiem Zugriff. Diese Modell wurde 1936 von Alan Turing vorgeschlagen.

Die Turingmaschine

- DTM = Deterministische Turingmaschine
- NTM = Nichtdeterministische Turingmaschine
- TM = DTM oder NTM

Intuitiv gilt:

- DTM = (DFA + dynamischer Speicher)
- NTM = (NFA + dynamischer Speicher)
- Der dynamische Speicher ist ein (in Zellen unterteiltes) zweiseitig unendliches Band versehen mit einem Lese–Schreibkopf. Es enthält anfangs die Eingabe, dient aber auch als Arbeitsspeicher.
- Die Einschränkung, den Kopf auf dem Band nur von links nach rechts bewegen zu dürfen, wird fallen gelassen.
- Die Einschränkung, den Arbeitsspeicher kellerartig organisieren zu müssen, wird ebenfalls fallen gelassen.

DTM (formale Definition)

Eine DTM M besteht aus den folgenden Komponenten:

- Z, die Zustandsmenge (eine endliche Menge)
- Σ, das Eingabealphabet (ebenfalls endlich)
- $\Gamma \supset \Sigma$, das Arbeitsalphabet (ebenfalls endlich)
- $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$, die partiell definierte Überführungsfunktion
- $z_0 \in Z$, der Startzustand
- $\square \in \Gamma \setminus \Sigma$, das Blank (auch Leerzeichen genannt)
- $E \subseteq Z$ die Menge der Endzustände:
  $\delta(z_e, A)$ ist undefiniert für alle $z_e \in E$ und alle $A \in \Gamma$.

Arbeitsweise der DTM

- Anfangs befindet sich M im Startzustand z_0, ihr Band enthält das Eingabewort $w \in \Sigma^*$ (umrahmt von Blanks) und der Kopf steht auf dem ersten Zeichen von w (bzw. auf einem Blank, falls $w = \varepsilon$).
- Falls sich M im Zustand $z \in Z$ befindet, der Kopf das Zeichen $A \in \Gamma$ liest und

$$\delta(z, A) = (z', A', d) \in Z \times \Gamma \times \{L, R, N\} \ ,$$

  dann geht M in den Zustand z' über und ersetzt A durch A'. Für $d = L$ bzw. $d = R$ erfolgt zusätzlich eine Kopfbewegung auf die linke bzw. rechte Nachbarzelle des Bandes.
- M stoppt **gdw** M in einem Zustand z ist, ein Symbol A liest und $\delta(z, A)$ undefiniert ist.
- Die Eingabe wird akzeptiert **gdw** M im Laufe der Rechnung in einen Endzustand gerät (und dann automatisch stoppt).

NTMs

Eine NTM M ist analog definiert.

Unterschied: Die Überführungsfunktion δ einer NTM hat die Form

$$\delta : Z \times \Gamma \to \mathcal{P}(Z \times \Gamma \times \{L, R, N\}) \ ,$$

wobei (in Analogie zu DTMs)

$$\forall z_e \in E, A \in \Gamma : \delta(z_e, A) = \emptyset \ .$$

Arbeitsweise von NTMs

Wie bei nicht-deterministischen Maschinen üblich hat die NTM die „Qual der Wahl“ (i.A. mehrere mögliche nächste Rechenschritte):

- Anfangs befindet sich M im Startzustand z_0, ihr Band enthält das Eingabewort $w \in \Sigma^*$ (umrahmt von Blanks) und der Kopf steht auf dem ersten Zeichen von w (bzw. auf einem Blank, falls $w = \varepsilon$).
- Falls sich M im Zustand $z \in Z$ befindet, der Kopf das Zeichen $A \in \Gamma$ liest und

  $$(z', A', d) \in \delta(z, A) \subseteq Z \times \Gamma \times \{L, R, N\} \ ,$$

  dann darf M in den Zustand z' übergehen, A durch A' ersetzen und die $d \in \{L, R, N\}$ entsprechende Kopfbewegung ausführen.
- Die Eingabe wird akzeptiert **gdw** M durch geeignete Wahl der möglichen Rechenschritte in einen Endzustand geraten kann.

Konfigurationen einer TM

Die Konfiguration einer TM besteht aus

- dem aktuellen Zustand $z \in Z$,
- der Position des Kopfes auf dem Band.
- dem Bandinhalt $\gamma \in \Gamma^*$ (im Bereich der Eingabe sowie der im Laufe der Rechnung bereits besuchten Zellen)

Notation: $\alpha z \beta$, wobei $\gamma = \alpha\beta$ der aktuelle Bandinhalt und der Kopf auf dem ersten Zeichen von β positioniert ist

Anfangskonfiguration bei Eingabe w: $z_0 w$ (hier: $\alpha = \varepsilon, \beta = w$)

Akzeptierende Endkonfiguration: $\alpha z \beta$ für jedes $z \in E, \alpha, \beta \in \Gamma^*$

Stoppkonfiguration: $\alpha z A \beta'$ für jedes $z \in Z, \alpha, \beta' \in \Gamma^*$, $A \in \Gamma$ mit $\delta(z, A)$ ist undefiniert.

Beobachtung: Da δ auf Endzuständen undefiniert ist, ist jede akzeptierende Endkonfiguration auch eine Stoppkonfiguration.

Folgekonfigurationen

Eine „Rechnung“ einer TM lässt sich als Folge von Konfigurationen beschreiben.

Definition:

1. $\alpha' z' \beta'$ heißt unmittelbare Folgekonfiguration von $\alpha z \beta$ **gdw** $\alpha' z' \beta'$ aus $\alpha z \beta$ durch einen „Rechenschritt“ (einmalige Verwendung der Überführungsfunktion) resultieren kann.
   Notation: $\alpha z \beta \vdash \alpha' z' \beta'$.

2. $\alpha' z' \beta'$ heißt Folgekonfiguration von $\alpha z \beta$ **gdw** $\alpha' z' \beta'$ aus $\alpha z \beta$ durch eine (evtl. leere) Folge von Rechenschritten resultieren kann.
   Notation: $\alpha z \beta \vdash^* \alpha' z' \beta'$.
   Formal ist „$\vdash^*$” die reflexive–transitive Hülle von „$\vdash$“.

Im Falle einer DTM ist die unmittelbare Folgekonfiguration stets eindeutig bestimmt und es gibt nur eine mögliche Rechnung auf der Eingabe.

Sprache einer TM

Die folgende Definition der von der TM M erkannten Sprache $T(M)$ entspricht unserer Vereinbarung über das Akzeptieren mit Endzustand:

$$T(M) := \{w \in \Sigma^* | \ \exists z \in E, \alpha, \beta \in \Gamma^* : z_0 w \vdash^* \alpha z \beta\}$$

In Worten: Wort w gehört zur Sprache $T(M)$ **gdw** M durch Verarbeitung der Eingabe w aus der Anfangskonfiguration in eine akzeptierende Endkonfiguration gelangen kann.

Haltebereich einer DTM

Wir definieren den Haltebereich einer DTM M wie folgt:

$H(M) :=$

$$\{w \in \Sigma^* | \ \exists z \in Z, \alpha, \beta' \in \Gamma^*, A \in \Gamma : z_0 w \vdash^* \alpha z A \beta', \ \delta(z, A) \text{ ist undefiniert}\}$$

In Worten: Wort w gehört zum Haltebereich $H(M)$ **gdw** M durch Verarbeitung der Eingabe w aus der Anfangskonfiguration in eine Stoppkonfiguration gelangt.

Beobachtung: Da jede Endkonfiguration auch eine Stoppkonfiguration ist, gilt $L(M) \subseteq H(M)$.

Beispiel

bin(n) bezeichne die Binärdarstellung einer Zahl $n \geq 0$.

Aufgabe: Implementiere einen Binärzähler, der, gestartet auf bin(n),

- bin($n+1$) berechnet,
- den Kopf auf dem ersten Zeichen von bin($n+1$) positioniert
- und sich dann in einen Endzustand begibt und stoppt.

Idee: Verwende vier Zustände für folgende Phasen der Berechnung:

- z_0: Suche das Bit am weitesten rechts.
- z_1: Inkrementiere den Zähler (unter Beachtung des Übertrages).
- z_2: Suche das Bit am weitesten links.
- z_e: Stoppe.

Beispiel (fortgesetzt)

Komponenten der „Binärzähler“–DTM:

- Zustandsmenge $\{z_0, z_1, z_2, z_e\}$
- Eingabealphabet $\{0, 1\}$
- Arbeitsalphabet $\{0, 1, \square\}$
- Überführungsfunktion δ (weiter unten spezifiziert)
- Startzustand z_0
- Blank $\square$
- Menge $\{z_e\}$ der Endzustände

Beispiel (fortgesetzt)

„Turing–Tafel“ von M (tabellarische Angabe von δ):

δ	0	1	$\square$
z_0	$(z_0, 0, R)$	$(z_0, 1, R)$	$(z_1, \square, L)$
z_1	$(z_2, 1, L)$	$(z_1, 0, L)$	$(z_e, 1, N)$
z_2	$(z_2, 0, L)$	$(z_2, 1, L)$	$(z_e, \square, R)$

DTMs versus realistischere Rechnermodelle

Die „Programmiersprache“ für eine Turingmaschine ist

- leicht zu erlernen,
- aber wenig problemorientiert und daher mühselig zu handhaben.

Wir werden an einem späteren Punkt der Vorlesung folgendes aufzeigen:

1. DTMs sind „universelle“ Rechnermodelle: alles was in einem intuitiven Sinne berechenbar ist, ist auch durch eine DTM berechenbar.
2. DTMs können ohne wesentlichen Effizienzverlust realistischere Modelle moderner Rechner simulieren.

Da die Angabe von Turing–Tafeln sehr mühselig ist, werden wir im folgenden die Strategie einer TM zur Lösung eines Problems mehr informell beschreiben (eine Vorgehensweise, die später durch den Nachweis der Universalität der DTM gerechtfertigt wird).

Linear beschränkte TMs

Definition: Für ein Zeichen $a \in \Sigma$ heißt $\hat{a}$ das zugehörige markierte Zeichen. Bei Wörtern der Form

$$a_1 \cdots a_{n-1}\hat{a}_n \text{ mit } a_1, \ldots, a_n \in \Sigma$$

sprechen wir von einem Eingabewort mit Endmarkierung.

Definition: Eine TM M heißt linear beschränkt, wenn sie, gestartet auf einem Eingabewort mit Endmarkierung, im Laufe ihrer Rechnung nur die Zellen besucht, welche im Eingabebereich liegen. Ihr Eingabealphabet hat dann die Form $\Sigma \cup \hat{\Sigma}$ und die von ihr erkannte Sprache ist gegeben durch

$$T(M) := \{a_1 \cdots a_{n-1}a_n \in \Sigma^+ | \ \exists z \in E, \alpha, \beta \in \Gamma^* : z_0 a_1 \cdots a_{n-1}\hat{a}_n \vdash^* \alpha z \beta\}$$

Bemerkung: Da eine linear beschränkte TM auf dem ersten Zeichen des Eingabewortes gestartet wird, kann sie für eine etwaige Markierung des linken Wortendes gleich im ersten Rechenschritt selber sorgen.

LBA und DLBA

- Eine linear beschränkte NTM heißt kurz LBA (Linear Bounded Automaton).
- Eine linear beschränkte DTM heißt kurz DLBA (Deterministic Linear Bounded Automaton).

Wir werden die folgenden Resultate zeigen:

- Die Klasse der von LBAs erkennbaren Sprachen stimmt überein mit der Klasse der kontextsensitiven Sprachen.
- Die Klasse der von NTMs erkennbaren Sprachen stimmt überein mit der Klasse der Sprachen vom Typ 0.

Von der kontextsensitiven Grammatik zum LBA

Satz: Jede kontextsensitive Grammatik G kann in einen äquivalenten LBA M transformiert werden.

Eine Beweisskizze

Gestartet auf der (mit Endmarkierung versehenen) Eingabe $w = a_1 \cdots a_{n-1}\hat{a}_n$ versucht M eine Ableitung $S \Rightarrow_G^* a_1 \cdots a_{n-1}a_n$ „rückwärts“ zu rekonstruieren:

- Die letzte „Satzform“ dieser Ableitung steht (in Form der Eingabe) bereits zu Anfang auf dem Band (mal abgesehen von der Endmarkierung).
- Nehmen wir induktiv an, eine Satzform β der Ableitung steht auf dem Band. Um zur Vorgängersatzform α gelangen zu können, darf M
  - eine Regel $u \to v \in P$ und einen Teilstring v in β (sofern vorhanden) auswählen,
  - v durch u ersetzen
  - und die Bandinschrift wieder komprimieren falls $|u| < |v|$.
- M versetzt sich in einen Endzustand **gdw** bei der beschriebenen Vorgehensweise irgendwann die „Satzform“ S auf dem Band steht.

Beweisskizze (fortgesetzt)

Beachte: Da für jede Regel $u \to v \in P$ die Bedingung $|u| \leq |v|$ erfüllt ist, führen die von M vorgenommenen Stringersetzungen niemals aus dem Eingabebereich heraus.

Offensichtlich sind die folgenden Aussagen äquivalent:

1. $w \in L(G)$.
2. Es existiert eine Ableitung von w aus S mit Regeln von G.
3. Es existiert eine Rechnung von M, die die Satzformen der Ableitung in umgekehrter Reihenfolge rekonstruiert (von w in Richtung S).
4. $w \in T(M)$.

Die technische Umsetzung der Beweisskizze durch Angabe der Komponenten von M lassen wir aus.

Von der Grammatik vom Typ 0 zur NTM

Satz: Jede Grammatik G vom Typ 0 kann in eine äquivalente NTM M transformiert werden.

Beweis analog zum entsprechenden Beweis für kontextsensitive Grammatiken und LBAs:

- M, gestartet auf Eingabe w, versucht eine Ableitung $S \Rightarrow_G^* w$ „rückwärts“ (von w in Richtung S) zu rekonstruieren.
- Diesmal ist die resultierende NTM **nicht** linear beschränkt, da bei Typ 0 Grammatiken die Satzformen, welche bei der Ableitung von w auftreten, (viel!) länger als w sein können.

Vom LBA zur kontextsensitiven Grammatik

Satz: Jeder LBA M kann in eine äquivalente kontextsensitive Grammatik G transformiert werden.

Aufbau des Beweises:

- Fasse eine Konfiguration als Satzform auf.
- Entwerfe kontextsensitive Regeln für G, die Rechenschritte von M simulieren.
- Erweitere das Regelsystem, damit das Eingabewort erhalten bleibt.
- Folgere schließlich die Äquivalenz von G und M.

Von der Konfiguration zur Satzform

Eine Konfiguration $\alpha z \beta$ ist ein String über $Z \cup \Gamma$.

Problem: Dieser String kann Länge $n+1$ haben (für $n =$ Eingabelänge). Eine kontextsensitive Grammatik darf aber bei der Ableitung eines Wortes der Länge n keine Satzformen einer n überschreitenden Länge verwenden.

Lösung: Fasse z und das erste Zeichen von β, sagen wir A, zu **einem** „Superzeichen“ (z, A) zusammen. Die Grammatik benötigt dazu Zeichen aus $\Gamma \cup (Z \times \Gamma)$.

Beispiel Konfiguration $azbcd$ aufgefasst als Satzform liest sich als $a(z, b)cd$ (bestehend aus **vier** Zeichen des erweiterten Alphabetes).

„Rechnende“ grammatische Regeln

Ein möglicher Rechenschritt von M kann nach folgendem Schema in einen möglichen Ableitungsschritt von G übersetzt werden:

Eintrag der Turing–Tafel	grammatische Regeln
$(z', A', L) \in \delta(z, A)$	$B(z, A) \to (z', B)A'$ für jedes $B \in \Gamma$
$(z', A', R) \in \delta(z, A)$	$(z, A)B \to A'(z', B)$ für jedes $B \in \Gamma$
$(z', A', N) \in \delta(z, A)$	$(z, A) \to (z', A')$

Das auf diese Weise aus der Turing–Tafel resultierende (kontextsensitive!) Regelsystem notieren wir als P'.

Für eine Konfiguration K bezeichne $\tilde{K}$ die zugehörige Satzform. Es gilt

$$K \vdash^* K' \text{ gdw } \tilde{K} \Rightarrow^* \tilde{K}' \ .$$

Insbesondere gilt $w = av \in T(M)$ gdw $z_0av \vdash^* \alpha z_e A\beta$ gdw $(z_0, a)v \Rightarrow^* \alpha(z_e, A)\beta$.

Erweiterung des Regelsystems

Problem: Die Ableitung soll nicht die zur Endkonfiguration passende Satzform generieren (die evtl. mit dem Eingabewort nicht viel gemein hat) sondern gerade das Eingabewort von M (abgesehen von der Endmarkierung).

Lösung: Blähe die Zeichen der Grammatik zu Paaren auf,

- deren erste Komponenten die Konfiguration repräsentieren (auf die oben besprochene Weise)
- und deren zweite Komponenten das Eingabewort konservieren.

Zur Umsetzung dieser Idee benötigen wir

Anfangsregeln zur Erzeugung von Startkonfiguration plus Eingabewort,

erweiterte Regeln zum Rechnen Regeln von P' erweitert um die zweite Komponente,

Schlussregeln zur Generierung des Eingabewortes von M (Wegschmeißen der ersten Komponenten).

Erweiterung des Regelsystems (fortgesetzt)

Die von P' verwendeten Zeichen waren aus der Menge $\Delta := \Gamma \cup (Z \times \Gamma)$.
Die zu M passende Grammatik G hat Variablenmenge $V = \{S, T\} \cup (\Delta \times \Sigma)$
mit S als Startsymbol und ein Regelsystem P bestehend aus folgenden Regeln:

Anfangsregeln: Für jedes $a \in \Sigma$:

$S \to T(\hat{a}, a) \mid ((z_0, \hat{a}), a),\ T \to T(a, a) \mid ((z_0, a), a)$

Falls $\varepsilon \in T(M)$, dann zusätzlich: $S \to \varepsilon$

Die Anfangsregeln erlauben die Generierung der Satzform, welche der Anfangskonfiguration von M zur Eingabe w (und dem Anlegen eines „Duplikates“ von w) entspricht.

Beispiel: Für $\Sigma = \{0, 1\}$ und $w = 010$ könnten die Anfangsregeln genutzt werden wie folgt:

$$S \Rightarrow T(\hat{0}, 0) \Rightarrow T(1, 1)(\hat{0}, 0) \Rightarrow ((z_0, 0), 0),)(1, 1)(\hat{0}, 0)$$

Erweiterung des Regelsystems (fortgesetzt)

Erweiterte Regeln zum Rechnen: Die Regeln aus P' (s. oben) haben entweder die Form $XY \to X'Y'$ oder die Form $X \to Y$ mit $X, Y, X', Y' \in \Delta$. Zum Beispiel: $XY = B(z, A)$ und $X'Y' = (z', B)A'$.

alte Regel aus P'	erweiterte Regeln aus P
$XY \to X'Y'$	$(X, a)(Y, b) \to (X', a)(Y', b)$ für alle $a, b \in \Sigma$
$X \to X'$	$(X, a) \to (X', a)$ für alle $a \in \Sigma$

Das sind die „guten alten“ rechnenden Grammatikregeln erweitert um die Konservierung des Eingabewortes w.

Schlussregeln: $(A, a) \to a$ und $((z_e, A), a) \to a$ für alle $z_e \in E, A \in \Gamma, a \in \Sigma$. Die Schlussregeln erlauben die Ableitung des konservierten Wortes w, sofern die zugrunde liegende Rechnung von M eine akzeptierende Endkonfiguration erreicht hat.

Das große Finale

Folgende Aussagen sind äquivalent:

1. $a_1 \cdots a_{n-1} a_n \in T(M)$.
2. Es existiert eine Rechnung von M auf Eingabe $a_1 \cdots a_{n-1} \hat{a}_n$, die eine akzeptierende Endkonfiguration K erreicht.
3. Es existiert eine Ableitung mit Regeln aus G, die eine Satzform mit Zeichen aus $\Delta \times \Sigma$ generiert. Dabei liefern die ersten Komponenten die Satzform $\tilde{K}$ zu einer akzeptierenden Endkonfiguration K von M und die zweiten Komponenten liefern $a_1 \cdots a_{n-1} a_n$. Mit den Schlussregeln ist dann hieraus $a_1 \cdots a_{n-1} a_n$ ableitbar.
4. $a_1 \cdots a_{n-1} a_n \in L(G)$.

Von der NTM zur Grammatik vom Typ 0

Satz: Jede NTM M kann in eine äquivalente Grammatik G vom Typ 0 transformiert werden.

Beweis analog zum entsprechenden Beweis für LBAs und kontextsensitive Grammatiken:

- Kernstück beim Entwurf der Grammatik sind wiederum die „rechnenden“ grammatischen Regeln.
- Wenn jedoch die NTM, gestartet auf einer Eingabe w der Länge n, Bandinhalte der Länge $m > n$ produziert, dann liefert die korrespondierende grammatische Ableitung auch Satzformen der Länge $N > n$. Um daraus schließlich w abzuleiten, werden **nicht–kontextsensitive** ε–Regeln (als Schlussregeln) zugelassen, welche die Löschung von den Teilen der Satzform außerhalb des Eingabebereiches erlauben.

Eine Zwischenbilanz

Eine DTM ist eine spezielle NTM. Wir werden bald sehen, dass jede NTM von einer DTM simuliert werden kann (allerdings nicht zeiteffizient). Abgesehen von Effizienzfragen haben also DTMs und NTMs „die gleiche Rechenkraft“. Zusammen mit den bisher bewiesenen Resultaten ergibt sich:

- Eine Sprache ist genau dann kontextsensitiv, wenn sie von einem LBA erkennbar ist.
- Eine Sprache ist genau dann vom Typ 0, wenn sie von einer NTM erkennbar ist oder, äquivalent dazu, von einer DTM.

Das LBA-Problem: Kann jeder LBA von einem DLBA simuliert werden ?

Das ist eine berühmte offene Frage!

Nichtdeterminismus und „Raten eines Beweises“

- Jede NTM M kann (ohne Abänderung der von ihr erkannten Sprache) so modifiziert werden, dass sie in jedem Schritt genau zwei Wahlmöglichkeiten hat.
- Wir können uns anschaulich vorstellen, dass eine NTM in jedem Schritt ein Bit, 0 oder 1, rät und dann die aktuelle Berechnung entsprechend fortsetzt.
- Jede Rechnung der Länge t auf einer festen Eingabe ist daher eindeutig durch einen Binärstring der Länge t (die Folge der geratenen Bits) beschreibbar.
- Eine akzeptierende Rechnung auf Eingabe w entspricht einer Rechnung von der Anfangskonfiguration z_0w zu einer Endkonfiguration. Den zugehörigen Binärstring (Folge der geratenen Bits) können wir als einen Beweis für $w \in T(M)$ ansehen.

Raten unterwegs oder ganz am Anfang — egal

Raten unterwegs: Die NTM nach unserer bisherigen Definition rät in jeder Konfiguration ein Bit, 0 oder 1, und setzt ihre Berechnung dann entsprechend fort.

Raten ganz am Anfang: Jede NTM M, die pro Schritt zwei Wahlmöglichkeiten hat, kann modifiziert werden (ohne dabei die Sprache $N(M)$ abzuändern) wie folgt:
Sie rät am Anfang der Rechnung einen Binärstring $u \in \{0,1\}^*$, den sie auf die Zellen $-1, \ldots, -|u|$ schreibt. Danach rechnet sie $|u|$ Schritte lang deterministisch, indem sie u auf die offensichtliche Weise als Wegbeschreibung durch den Konfigurationsdigraphen interpretiert. Genau dann, wenn sie dabei zu einer Endkonfiguration gelangt, akzeptiert sie ihre Eingabe.

Die NTM mit „Raten ganz am Anfang“ arbeitet nach dem sogenannten „Rate-Verifikationsprinzip“: rate einen „Beweis“ u für $w \in T(M)$ und verifiziere mit Hilfe von u anschließend deterministisch, dass die Eingabe w akzeptabel ist.

Determinismus versus Nondeterminismus

Jede DTM kann als Spezialfall einer NTM aufgefasst werden. Es gilt aber auch umgekehrt der

Satz: Jede NTM M kann von einer DTM M' simuliert werden.

Determinismus versus Nondeterminismus (fortgesetzt)

Beweis: Wir dürfen annehmen, dass M gemäß dem Rate-Verifikationsprinzip vorgeht. M' arbeitet wie folgt:

- M' probiert alle Strings $u = \varepsilon, 0, 1, 00, 01, 10, 11, 000 \ldots$ der Reihe nach aus.
- Für jeden festen String u rechnet M' so wie M in ihrer deterministischen Verifikationsphase mit „Beweis“ u. Falls dabei eine Endkonfiguration erreicht wird, akzeptiert M' und stoppt die Simulation.

Es ist offensichtlich, dass folgendes gilt:

- Wenn M eine akzeptierende Rechnung auf Eingabe w besitzt, dann wird w auch von M' nach endlich vielen Schritten akzeptiert.
- Auf Eingaben $w \notin N(M)$ rechnet M' endlos (ohne zu akzeptieren).

Ein Beispiel

Die Sprache der Nicht-Primzahlen kann nach dem Rate-Verifikationsprinzip von einer NTM M erkannt werden wie folgt:

- Rate zwei ganze Zahlen $a, b \geq 2$ (bzw. die Bits ihrer Binärdarstellung).
- Verifiziere deterministisch, durch Multiplizieren von a und b, dass die Eingabezahl n sich gemäß $n = ab$ zerlegen lässt (und somit keine Primzahl ist).
- Das Zahlenpaar (a, b) ist in diesem Beispiel der Beweis dafür, dass die Eingabezahl n keine Primzahl ist. (Natürlich besitzen nur Nicht-Primzahlen einen solchen Beweis.)
- Die deterministische Simulation würde alle denkbaren Beweise (hier: Zahlenpaare (a, b)) der Reihe nach durchprobieren.

Abschlusseigenschaften

Satz: Die Klasse der kontextsensitiven Sprachen und die Klasse der Sprachen vom Typ 0 sind abgeschlossen unter den Operationen „$\cup, \cap, \cdot, *$“.

Beweisidee: Es ist leicht NTMs M_1, M_2 mit $T(M_1) = L_1$ und $T(M_2) = L_2$ zu NTMs zusammenzusetzen, welche $L_1 \cup L_2$, $L_1 \cap L_2$, oder $L_1 \cdot L_2$ erkennen. Weiterhin kann M_1 so modifiziert werden, dass ein Akzeptor von L_1^* entsteht.

Wir verzichten auf die Angabe technischer Details.

Wir werden später zeigen, dass Folgendes gilt:

Satz: Die Klasse der Sprachen vom Typ 0 ist nicht abgeschlossen unter der Operation $\neg$.

Die entsprechende Frage fur kontextsensitive Sprachen (zum LBA–Problem verwandt) war lange Zeit offen, wurde dann aber im Jahre 1987 von Immerman und Szelepcsényi gelöst.

Satz von Immerman und Szelepcsényi

Satz: Die Klasse der kontextsensitiven Sprachen ist unter Komplementbildung abgeschlossen.

Beweisidee: Für eine gegebene kontextsensitive Sprache L mit einer kontextsensitiven Grammatik G wird ein LBA konstruiert, der die Sprache $\bar{L}$ (Komplement von L) erkennt.

Sei $n := |w|$ die Länge der Eingabe. Der LBA berechnet zunächst die Anzahl $a(n)$ aller ableitbaren Satzformen einer Länge $\leq n$. Dann versucht der LBA $w \notin L$ zu verifizieren, indem er alle Satzformen Länge $\leq n$ aufzählt. Wenn er $a(n)$ von w verschiedene solche Satzformen zählt, akzeptiert der LBA w.

Die Zahl $a(n)$ wird durch die Methode des induktiven Zählens berechnet.

Beweis des Satzes

Zwei technische Probleme müssen wir lösen:

1. Gegeben $a(n)$ implementiere die besprochene Strategie auf einem LBA.
2. Erweitere diesen LBA um die Vorausberechnung von $a(n)$.

Das zweite technische Problem wird mit der sogenannten Methode des induktiven Zählens gelöst werden.

Die Lösung der beiden technischen Probleme muss platzeffizient erfolgen:

- Wir zitieren ohne Beweis das folgende Resultat: Eine NTM (bzw. DTM), die auf Eingaben der Länge n maximal $O(n)$ Zellen besucht, kann in einen äquivalenten LBA (bzw. DLBA) transformiert werden.
- $O(n)$ Zellen reichen aus, zum Beispiel um die aktuelle Satzform der Ableitung eines Strings der Maximallänge n zu speichern (plus evtl. konstant viele weitere solcher Satzformen) sowie konstant viele Binärzähler, die bis $a(n)$ zählen können.

Nichtdeterministische Tests auf Ableitbarkeit

Analog zur Konstruktion eines LBA, der zu einer kontextsensitiven Grammatik äquivalent ist, lässt sich folgende nichtdeterministische Prozedur MEMBER(α, G) für eine Satzform α und kontextsensitive Grammatik G auf einem LBA umsetzen:

- wenn die Satzform α nicht aus S ableitbar ist, antwortet MEMBER(α, G) stets FALSE
- wenn die Satzform α aus S ableitbar ist, dann liefert mindestens eine mögliche Rechnung der Prozedur TRUE

Diese Prozedur lässt sich verfeinern zu der Prozedur MEMBER(α, G, m), welche Ableitbarkeit in m Schritten testet.

Erkennen von $\bar{L}$ mit Hilfe von $a(n)$

Wir setzen zur Abkürzung für Satzformen der Länge $\leq n$ ohne w:

$$W := \bigcup_{l=0}^{n} (V \cup \Sigma)^l \setminus \{w\} \ .$$

Hier nun ein LBA, der, mit Hilfe von $a(n)$, $w \in \bar{L}$ verifizieren kann:

Eingabe: $w \in \Sigma^*$ und die Zahl $a(n)$
Ausgabe: ACCEPT sofern $w \in \bar{L}$ verifiziert wurde
begin
a:=0;
for all $\alpha \in W$ **do if** MEMBER(α, G) **then** a:=a+1;
if $a = a(n)$ **then** ACCEPT
end

Vorausberechnung von $a(n)$

Es bezeichne $a(m,n)$ die Anzahl aller Satzformen einer Länge $\leq n$ die in $\leq m$ Schritten aus Startsymbol S mit Regeln aus G ableitbar sind.

Offensichtlich gilt:

- $a(0,n) = 1$.
- $a(n) = a(m_*, n)$, wobei m_* das kleinste m bezeichne mit der Eigenschaft

$$a(m,n) = a(m+1,n).$$

Die wesentliche Schwierigkeit bei der Berechnung von $a(n)$ besteht also darin, $a(m,n)$ aus $a(m-1,n)$ auszurechnen. Die nichtdeterministische Prozedur dafür (s. folgende Seite) nutzt die Äquivalenz der folgenden beiden Aussagen aus:

(1) $\beta \in W$ kann in $\leq m$ Schritten aus Startsymbol S abgeleitet werden.

(2) $\beta = S$ oder es gibt ein $\alpha \in W$ und eine Ableitung der Form $S \Rightarrow_G^* \alpha \Rightarrow_G \beta$, die zur Ableitung von α höchstens $m-1$ Schritte benötigt.

Berechnung von $a(m,n)$ aus $a(m-1,n)$

begin

b:=1; `Kommentar:` b `soll am Ende den Wert` $a(m,n)$ `haben.`

for all $\beta \in W$ **do**

begin `Kommentar: Äußere Schleife`

a:=0; `Kommentar:` a `soll am Ende den Wert` $a(m-1,n)$ `haben.`

for all $\alpha \in W$ **do**

begin `Kommentar: Innere Schleife`

if MEMBER$(\alpha, G, m-1)$ **then**

(a) a:=a+1;

(b) **if** $\alpha \Rightarrow_G \beta$ **then**

$b := b+1$; verzweige direkt zur nächsten Iteration der äußeren Schleife

end;

if $a \neq a(m-1,n)$ **then** melde „Berechnung misslungen“ und stoppe

end

end `Kommentar:` $b = a(m,n)$ `sofern nicht „Berechnung misslungen“.`

Korrektheit des skizzierten LBA

Es gilt:

- Falls $w \in L$, dann gibt es keine akzeptierende Rechnung auf Eingabe w.
- Falls $w \in \bar{L}$, dann gibt es eine Rechnung, die
  - $a(n)$ mit der Technik des induktiven Zählens korrekt bestimmt
  - grammatische Ableitungen von $a(n)$ vielen Satzformen aus W ausfindig macht (was, wegen $w \notin W$, indirekt $w \in \bar{L}$ verifiziert),
  - und schließlich w akzeptiert.

Der skizzierte LBA ist daher ein Akzeptor von $\bar{L}$.

Überblick Entscheidbarkeit

Für kontextsensitive und Typ 0 Sprachen sind das Leerheits-, Äquivalenz- und Schnittproblem nicht entscheidbar. Das Wortproblem für kontextsensitive Sprachen ist (wie wir wissen) in exponentieller Zeit entscheidbar. Für Typ 0 Sprachen ist ebenfalls das Wortproblem nicht entscheidbar.

Sprache	Wort-problem	Leerheits-problem	Äquivalenz-problem	Schnitt-problem
Typ 0	nein	nein	nein	nein
Typ 1	ja	nein	nein	nein
Typ 2	ja	ja	nein	nein
det. kontextfrei	ja	ja	ja	nein
Typ 3	ja	ja	ja	ja

Überblick Formale Sprachen & Maschinenmodelle

Grammatik	Automat
regulär	DFA, NFA
LR(k)	DPDA
kontextfrei	PDA
kontextsensitiv	LBA
Typ 0	DTM, NTM

Deterministisch	Nicht-deterministisch	Äquivalenz
DFA	NFA	ja
DPDA	PDA	nein
DLBA	LBA	?
DTM	NTM	ja

Lernziele (kontextsensitive und Typ 0 Sprachen)

- Kenntnis der wesentlichen Konzepte und Resultate auf dem Level der kontextsensitiven und Typ 0 Sprachen besitzen und Zusammenhänge verstehen
- zu einer kontextsensitiven Sprache die passende kontextsensitive Grammatik bzw. den passenden LBA (falls möglich DLBA) angeben können (Angabe der Turing–Tafel oder evtl. auch nur informelle Beschreibung der Arbeitsweise)
- zu einer gegebenen kontextsensitiven Grammatik (bzw. einem gegebenen LBA oder DLBA) die zugehörige Sprache ableiten können
- zu einer Typ 0 Sprache die passende Typ 0 Grammatik bzw. die passende TM (mit Turing–Tafel oder evtl. auch nur mit informeller Beschreibung der Arbeitsweise) angeben können
- zu einer gegebenen Typ 0 Grammatik (bzw. einer gegebenen TM) die zugehörige Sprache ableiten können

Lernziele (fortgesetzt)

Zu folgenden Punkten genügt es die „zündende Idee“ zu kennen:

- Äquivalenz von LBAs und kontextsensitiven Grammatiken
- Äquivalenz von NTMs und Typ 0 Grammatiken
- NTMs und das Rate–Verifikationsprinzip
- Äquivalenz von NTMs und DTMs
- die besprochenen Abschlusseigenschaften