Kontextsensitive und Typ 0 Sprachen

Hans U. Simon (RUB)

Email: simon@lmi.rub.de

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

Kuroda Normalform

Erinnerung: Die Regeln einer kontextsensitiven Grammatik haben die Form

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

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

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

besitzt.

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 $A \rightarrow a$ 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 $A \rightarrow B$) mehr.

Danach ist die Grammatik in Kuroda Normalform.

Transformation in Kuroda Normalform (fortgesetzt)

Dir folgenden Ziele können ähnlich wie bei der Transformation einer kontextfreien 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

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

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

$$\begin{array}{rcc|ccc}
A_1 A_2 & \rightarrow & B_1 R_1 & R_{m-1} & \rightarrow & B_m R_m \\
R_1 A_3 & \rightarrow & B_2 R_2 & R_m & \rightarrow & B_{m+1} R_{m+1} \\
 & \cdot & & & \cdot & \\
 & \cdot & & & \cdot & \\
 & \cdot & & & \cdot & \\
R_{m-2} A_m & \rightarrow & B_{m-1} R_{m-1} & R_{n-2} & \rightarrow & B_{n-1} B_n
\end{array}$$

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.

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.

Diesmal ist aber der Zugriff auf den dynamischen Speicher nicht durch kellerartige Organisation eingeschränkt !!

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)
- $\delta : Z \times \Gamma \rightarrow \ Z \times \Gamma \times \{L, R, N\}$, die Überführungsfunktion
- $z_0 \in Z$, der Startzustand
- $\square \in \Gamma \setminus \Sigma$, das Blank (auch Leerzeichen genannt)
- $E \subseteq Z$ die Menge der Endzustände

Eine NTM M ist analog definiert.

Unterschied: Die Überführungsfunktion δ einer NTM hat die Form

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

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.

- Die Eingabe wird akzeptiert **gdw** M im Laufe der Rechnung in einen Endzustand gerät.

Arbeitsweise der NTM

Wie üblich ist das nichtdeterministische Modell ähnlich definiert wie das deterministische, hat aber 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 $\gamma \in \Gamma^*$ (inklusive des Eingabebereiches und den bereits während der Rechnung 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 $z \in E, \alpha, \beta \in \Gamma^*$

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.

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.

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: Entspanne Dich (Endzustand).

Beispiel (fortgesetzt)

Komponenten der „Binärzähler“–DTM:

- Zustandsmenge $\{z_0, z_1, z_2, z_e\}$
- Eingabealphabet $\{0.1\}$
- Arbeitsalphabet $\{0, 1, \Box\}$
- Überführungsfunktion δ (weiter unten spezifizierrt)
- Startzustand z_0
- Blank $\Box$
- Menge $\{z_e\}$ der Endzustände

Konvention: Wir erlauben im Folgenden $\delta(z, A)$ undefiniert zu lassen, falls z ein Endzustand ist.

Interpretation: Falls M im Zustand z Symbol A liest und $\delta(z, A)$ undefiniert ist, dann wird die Rechnung abgebrochen.

Beispiel (fortgesetzt)

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

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

`--- Macht das Sinn ?? Erläutere !! ---`

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 $N(M)$ **gdw** M durch Verarbeitung der Eingabe w aus der Anfangskonfiguration in eine akzeptierende Endkonfiguration gelangen kann.

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 müselig 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 a' das zugehörige markierte Zeichen. Bei Wörtern der Form

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

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 $\Sigma \cup \Sigma'$ und die von ihr erkannte Sprache ist gegeben durch

$$T(M) := \{a_1 \cdots a_{n-1}a_n \in \Sigma^+ |\ \exists z \in E, \alpha, \beta \in \Gamma^* : z_0 a_1 \cdots a_{n-1}a'_n \vdash^* \alpha z \beta\}$$

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 $w = a_1 \cdots a_{n-1}a'_n$ 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 $u \rightarrow v \in P$ 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 $u \rightarrow v \in P$ 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. $w \in T(M)$.

Technische Umsetzung der Beweisskizze durch Angabe der Komponenten von M lassen wir aus (wird aber teilweise in den Übungen thematisiert).

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. Offensichtlich gilt

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

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 folgende Regeln:

Anfangsregeln $S \rightarrow T(a', a)$, $T \rightarrow T(a, a) \mid ((z_0, a), a)$ für jedes $a \in \Sigma$.

Erweiterte Regeln zum Rechnen $(X, a)(Y, b) \rightarrow (X', a)(Y', b)$ bzw. $(X, a) \rightarrow (X', a)$ für jede Regel $XY \rightarrow X'Y' \in P'$ bzw. für jede Regel $X \rightarrow X' \in P'$ und für alle $a, b \in \Sigma$.

Schlussregeln $(A, a) \rightarrow a$ und $((z, A), a) \rightarrow a$ für alle $z \in E, A \in \Gamma, a \in \Sigma$.

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 $a_1 \cdots a_{n-1}a'_n$, 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 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 (wegen der bei Typ 0 Grammatiken fehlenden Monotonie–Eigenschaft) die Satzformen, die bei der Generierung eines Eingabewortes auftreten, i.A. länger sind als das Eingabewort selbst.

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 den Teilen der Satzform außerhalb des Eingabebereiches erlauben.

Determinismus versus Nondeterminismus

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

Satz: Jede NTM kann in eine äquivalente DTM transformiert werden.

Beweis des Satzes

Die Simulation einer NTM M durch eine DTM M' beruht auf folgenden Ideen;

- Jede NTM kann (ohne Abänderung der von ihr erkannten Sprache) so modifiziert werden, dass sie in jedem Schritt genau zwei mögliche Rechenschritte hat (s. Übungen).
- Ein Bit $b \in \{0, 1\}$ kann als Vorschrift interpretiert werden, welcher von zwei möglichen Rechenschritten getätigt werden soll.
- Ein Bitstring $b_1 \cdots b_t$ liefert dann eine solche Vorschrift für t Schritte.
- Die deterministische Simulation von M hält einen Binärzähler BZ aufrecht, der auf 0 initialisiert und dann in einer „äußeren Schleife“ so hochgezählt wird, dass er die Binärstrings in der „natürlichen“ Reihenfolge

  $$0, 1, 00, 01, 10, 11, 000 \cdots$$

  durchläuft.

Beweis des Satzes (fortgesetzt)

- Wenn $b_1 \cdots b_t$ der aktuelle Zählerstand ist, dann wird zunächst M in der oben besprochenen Weise für t Schritte deterministisch simuliert, bevor BZ inkrementiert wird.
- Wenn jemals ein Endzustand von M erreicht wird, dann wird die Simulation abgebrochen und die Eingabe akzeptiert.

Die beschriebene Simulation hat zu jedem Zeitpunkt immer nur einen möglichen nächsten Rechenschritt und ist daher deterministisch. Man kann eine DTM M' entwerfen, welche diese Simulation durchführt.

Beweis des Satzes (fortgesetzt)

Offensichtlich sind die folgenden Aussagen äquivalent:

1. $w \in T(M)$.
2. Es existiert ein $t \geq 1$, so dass M auf Eingabe w und bei geeigneter Wahl der Rechenschritte nach t Schritten einen Endzustand erreicht.
3. Es existiert ein $t \geq 1$ und ein Binärstring $b_1 \cdots b_t$, so dass M' spätestens in der Iteration mit BZ=$b_1 \cdots b_t$ in einen Endzustand gerät.
4. $w \in T(M')$.

Bemerkungen:

- M' probiert im Prinzip alle möglichen Rechnungen von M auf Eingabe w systematisch aus (Technik der „Exhaustive Search“).
- Wenn M die Eingabe nach t Rechenschritten akzeptieren kann, dann benötigt M' mindestens $2^t - 1$ Iterationen der äußeren Schleife, um diese akzeptierende Rechnung aufzuspüren.

Folgerung und das LBA–Problem

Folgerung: Auch die Klasse der von DTMs erkennbaren Sprachen stimmt überein mit der Klasse der Sprachen vom Typ 0.

Die entsprechende Frage für LBAs, DLBAs und kontextsensitive Sprachen ist schon lange Zeit offen:

Das LBA–Problem: Kann jeder LBA (oder alternativ jede kontextsensitive Grammatik) in einen äquivalenten DLBA transformiert werden ??

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.

Zum Beweis konstruieren wir für eine gegebene kontextsensitive Sprache L mit einer kontextsensitiven Grammatik G einen LBA, der die Sprache $\bar{L}$ (Komplement von L) erkennt.

Problem: Erkennen, dass ein Eingabewort w zu L gehört, gelang durch „Raten“ der grammatischen Ableitung $S \Rightarrow^* w$. Aber wie können wir nichtdeterministisch erkennen, dass w **nicht** zu L gehört ?

Idee: Setze $n := |w|$. Angenommen wir kennen die Anzahl $a(n)$ aller ableitbaren Satzformen einer Länge $\leq n$. Dann können wir versuchen $w \notin L$ zu verifizieren, indem wir verifizieren, dass $a(n)$ von w verschiedene Satzformen einer Länge $\leq n$ ableitbar sind.

Fortführung des Beweises

Bleiben also zwei technische Probleme:

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.

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

Input: $w \in \Sigma^*$ and the number $a(n)$

Output: ACCEPT provided that $w \in \bar{L}$ has been verified

begin

a:=0

for all $\alpha \in \left(\cup_{l=1}^{n}(V \cup \Sigma)^l\right) \setminus \{w\}$

do GUESS a derivation $S \Rightarrow_G^* \alpha$

if SUCCESS **then** a:=a+1 **fi**

od

if $a := a(n)$ **then** ACCEPT w **fi**

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 mit der Eigenschaft

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

  bezeichne.

Die wesentliche Schwierigkeit bei der Berechnung von $a(n)$ besteht also darin, $a(m,n)$ aus $a(m-1,n)$ auszurechnen.

- „Pseudocode“ für die Berechnung von $a(m,n)$ aus $a(m-1,n)$ auf folgender Folie
- Zusätzliche Erläuterungen in der Vorlesung

begin

b:=0 (**Comment:** Variable b should have value $a(m,n)$ at the end.)

for all $\beta \in \cup_{l=1}^{n}(V \cup \Sigma)^l$

do (**Comment:** Outer Loop (OL))

a:=0

for all $\alpha \in \cup_{l=1}^{n}(V \cup \Sigma)^l$

do (**Comment:** Inner Loop (IL))

GUESS a derivation $S \Rightarrow_G^* \alpha$ of length at most $m-1$

if SUCCESS

then a:=a+1

if $\alpha \Rightarrow_G \beta$ **then** $b := b+1$; goto next iteration of OL **fi**

fi

od

if $a \neq a(m-1,n)$ **then** STOP (without success)

od

end

Korrektheit des skizzierten LBA

Offensichtlich gilt:

- Die beschriebene Vorgehensweise zum Erkennen von $\bar{L}$ ist auf einem LBA implementierbar (detailliertere Information in der Vorlesung).
- 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 detailliertere Information in der Vorlesung),
  - $a(n)$ grammatische Ableitungen von Satzformen $\neq w$ der Länge $\leq n$ ausfindig macht
  - und schließlich w akzeptiert

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