Informatik IV
Theoretische Informatik

Kapitel 5
Kontextsensitive und $\mathfrak{L}_0$-Sprachen
Sommersemester 2019

Dozent: Prof. Dr. J. Rothe

CF versus CS

Theorem

CF *ist echt in* CS *enthalten.*

Beweis:

- Nach dem Pumping-Lemma für kontextfreie Sprachen ist

$$L = \{a^m b^m c^m \mid m \geq 1\}$$

  nicht kontextfrei.
- Andererseits ist L kontextsensitiv, wie die Grammatik aus einem früheren Beispiel zeigt.
- Somit ist $L \in \mathrm{CS} - \mathrm{CF}$, also $\mathrm{CF} \subset \mathrm{CS}$. ❑

Ziel: Automatenmodelle für CS und für $\mathfrak{L}_0$.

Turingmaschinen

- Ein grundlegendes, einfaches, abstraktes Algorithmenmodell ist die Turingmaschine, die 1936 von Alan Turing (1912 bis 1954) in seiner bahnbrechenden Arbeit "*On computable numbers, with an application to the Entscheidungsproblem*" eingeführt wurde.
- Wir betrachten wieder zwei Berechnungsparadigma:
  - Determinismus und
  - Nichtdeterminismus.
- Es ist zweckmäßig, zuerst das allgemeinere Modell der nichtdeterministischen Turingmaschine zu beschreiben. Deterministische Turingmaschinen ergeben sich dann sofort als ein Spezialfall.
- PDAs und somit auch NFAs und DFAs sind spezielle TMs.

Turingmaschinen: Modell und Arbeitsweise

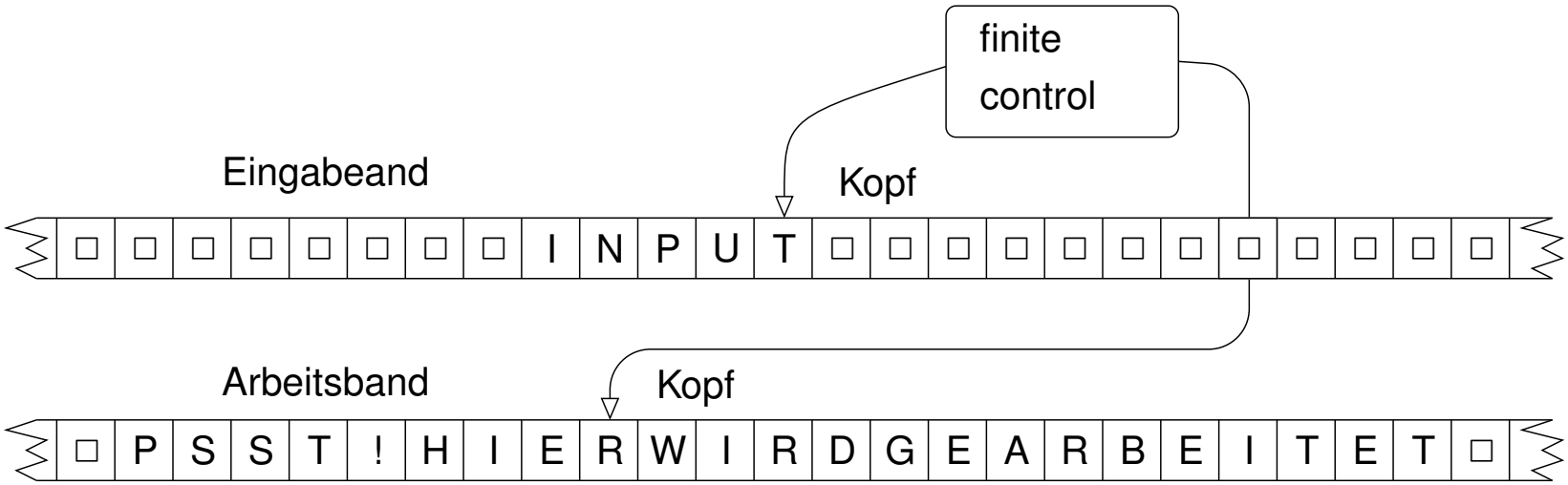

Abbildung: Eine Turingmaschine

- Eine Turingmaschine ist ausgestattet mit:
  - k beidseitig unendlichen Arbeitsbändern,
  - die in Felder unterteilt sind,
  - in denen Buchstaben oder das □-Symbol stehen können.
  - Das □-Symbol signalisiert ein leeres Feld.

Turingmaschinen: Modell und Arbeitsweise

- Auf den Arbeitsbändern findet die eigentliche Rechnung statt.
- Zu Beginn einer Rechnung:
  - steht das Eingabewort auf dem Eingabeband, und
  - alle anderen Felder enthalten das □-Zeichen.
- Am Ende der Rechnung erscheint das Ergebnis der Rechnung auf dem Ausgabeband.
- Auf jedes Band kann je ein Schreib-Lese-Kopf zugreifen. Dieser kann in einem Takt der Maschine
  - den aktuell gelesenen Buchstaben überschreiben und dann
    - eine Bewegung um ein Feld nach rechts oder
    - links ausführen oder aber
    - auf dem aktuellen Feld stehenbleiben.
  - Gleichzeitig kann sich der aktuelle Zustand der Maschine ändern, den sie sich in ihrem inneren Gedächtnis ("*finite control*") merkt.

Nichtdeterministische Turingmaschine: Syntax

Definition

Eine *(nichtdeterministische) Turingmaschine mit k Bändern* (kurz k-Band-TM) ist ein 7-Tupel $M = (\Sigma, \Gamma, Z, \delta, z_0, \Box, F)$, wobei

- Σ das Eingabe-Alphabet ist,
- Γ das Arbeitsalphabet mit $\Sigma \subseteq \Gamma$,
- Z eine endliche Menge von Zuständen mit $Z \cap \Gamma = \emptyset$,
- $\delta : Z \times \Gamma^k \to \mathfrak{P}(Z \times \Gamma^k \times \{L, R, N\}^k)$ die Überführungsfunktion,
- $z_0 \in Z$ der Startzustand,
- $\Box \in \Gamma - \Sigma$ das „Blank“-Symbol (oder Leerzeichen) und
- $F \subseteq Z$ die Menge der Endzustände.

Deterministische Turingmaschine: Syntax

Definition

Der Spezialfall der *deterministischen Turingmaschine mit k Bändern* ergibt sich, wenn die Überführungsfunktion δ von $Z \times \Gamma^k$ nach $Z \times \Gamma^k \times \{L, R, N\}^k$ abbildet.

Bemerkung:

- Für $k = 1$ ergibt sich die 1-Band-Turingmaschine (1-Band-TM).
- Jede k-Band-TM kann durch eine 1-Band-Turingmaschine simuliert werden, wobei sich die Rechenzeit höchstens quadriert.
- Spielt die Effizienz eine Rolle, kann es dennoch sinnvoll sein, mehrere Bänder zu haben.
- Wir beschränken uns im Folgenden auf 1-Band-Turingmaschinen.

Turingmaschine: Arbeitsweise

- Statt $(z', b, x) \in \delta(z, a)$ mit $z, z' \in Z$, $x \in \{L, R, N\}$, $a, b \in \Gamma$ schreiben wir kurz:

$$(z, a) \to (z', b, x),$$

oder (wenn es keine Verwechslungen geben kann):

$$za \to z'bx.$$

- Dieser Turingbefehl bedeutet: Ist im Zustand z der Kopf auf einem Feld mit aktueller Inschrift a, so wird:
  - a durch b überschrieben,
  - der neue Zustand z' angenommen und
  - eine Kopfbewegung gemäß $x \in \{L, R, N\}$ ausgeführt (d.h., ein Feld nach *L*inks, ein Feld nach *R*echts oder *N*eutral, also Stehenbleiben).

Turingmaschine: Semantik

- Turingmaschinen kann man sowohl als Akzeptoren auffassen, die Sprachen (also Wortmengen) akzeptieren,
- als auch zur Berechnung von Funktionen benutzen.
- In diesem Abschnitt betrachten wir Turingmaschinen als Akzeptoren;
- die Funktionsberechnung einer Turingmaschine wird später definiert.

Turingmaschine: Semantik

Definition

- Eine *Konfiguration einer TM* $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$ ist ein Wort

$$k \in \Gamma^* Z \Gamma^*.$$

Dabei bedeutet $k = \alpha z \beta$, dass

  - $\alpha\beta$ die aktuelle Bandinschrift ist (also das Wort auf dem bereits vom Kopf besuchten Teil des Bandes),
  - der Kopf auf dem ersten Symbol von β steht und
  - z der aktuelle Zustand von M ist.

- Auf der Menge $\mathfrak{K}_M = \Gamma^* Z \Gamma^*$ aller Konfigurationen von M definieren wir eine binäre Relation $\vdash_M$ wie folgt.
Intuitiv gilt $k \vdash_M k'$ für $k, k' \in \mathfrak{K}_M$ genau dann, wenn k' aus k durch eine Anwendung von δ hervorgeht.

Turingmaschine: Semantik

- Formal: Für alle $\alpha = a_1 a_2 \cdots a_m$ und $\beta = b_1 b_2 \cdots b_n$ in Γ^* ($m \geq 0$, $n \geq 1$) und für alle $z \in Z$ sei

$$\alpha z \beta \vdash_M \begin{cases} a_1 a_2 \cdots a_m z' c b_2 \cdots b_n & \text{falls } (z, b_1) \to (z', c, N),\ m \geq 0,\ n \geq 1 \\ a_1 a_2 \cdots a_m c z' b_2 \cdots b_n & \text{falls } (z, b_1) \to (z', c, R),\ m \geq 0,\ n \geq 2 \\ a_1 a_2 \cdots a_{m-1} z' a_m c b_2 \cdots b_n & \text{falls } (z, b_1) \to (z', c, L),\ m \geq 1,\ n \geq 1. \end{cases}$$

Sonderfälle:

1. $n = 1$ und $(z, b_1) \to (z', c, R)$ (d.h., M läuft nach rechts, trifft auf $\Box$):

$$a_1 a_2 \cdots a_m z b_1 \vdash_M a_1 a_2 \cdots a_m c z' \Box.$$

2. $m = 0$ und $(z, b_1) \to (z', c, L)$ (d.h., M läuft nach links, trifft auf $\Box$):

$$z b_1 b_2 \cdots b_n \vdash_M z' \Box c b_2 \cdots b_n.$$

Turingmaschine: Semantik

- Die *Startkonfiguration von M bei Eingabe x* ist stets $z_0 x$.
- Die *Endkonfigurationen von M bei Eingabe x* haben die Form $\alpha z \beta$ mit $z \in F$ und $\alpha, \beta \in \Gamma^*$. M hält an, falls eine Endkonfiguration erreicht wird, oder falls kein Turingbefehl mehr auf die aktuelle Konfiguration von M anwendbar ist.
- Sei $\vdash_M^*$ die reflexive, transitive Hülle von $\vdash_M$.
- Die *von der TM M akzeptierte Sprache* ist definiert durch

$$L(M) = \{x \in \Sigma^* \mid z_0 x \vdash_M^* \alpha z \beta \text{ mit } z \in F \text{ und } \alpha, \beta \in \Gamma^*\}.$$

Turingmaschine

Bemerkung:

- Da im Falle einer nichtdeterministischen TM jede Konfiguration mehrere Folgekonfigurationen haben kann, ergibt sich ein *Berechnungsbaum*,
  - dessen Wurzel die Startkonfiguration und
  - dessen Blätter die Endkonfigurationen sind.

- Die Knoten des Berechnungsbaums von $M(x)$ sind die Konfigurationen von M bei Eingabe x.

- Für zwei Konfigurationen k und k' aus $\mathfrak{K}_M$ gibt es genau dann eine gerichtete Kante von k nach k', wenn gilt:

$$k \vdash_M k'.$$

Turingmaschine

- Ein Pfad im Berechnungsbaum von $M(x)$ ist eine Folge

$$k_0 \vdash_M k_1 \vdash_M \cdots \vdash_M k_t \vdash_M \cdots$$

  von Konfigurationen, also eine Rechnung von $M(x)$.
- Der Berechnungsbaum einer nichtdeterministischen TM (NTM) kann unendliche Pfade haben.
- Im Falle einer deterministischen TM (DTM) wird jede Konfiguration außer der Startkonfiguration eindeutig (*deterministisch*) durch ihre Vorgängerkonfiguration bestimmt. Der Berechnungsbaum einer DTM entartet zu einer linearen Kette
  - von der Startkonfiguration zu einer Endkonfiguration, falls die Maschine bei dieser Eingabe hält;
  - andernfalls geht die Kette ins Unendliche.

Turingmaschine: Beispiel

Beispiel: Betrachte die Sprache $L = \{a^n b^n c^n \mid n \geq 1\}$.

Eine Turingmaschine, die L akzeptiert, ist definiert durch

$$M = (\{a, b, c\}, \{a, b, c, \$, \square\}, \{z_0, z_1, \ldots, z_6\}, \delta, z_0, \square, \{z_6\})$$

$(z_0, a) \mapsto (z_1, \$, R)$	$(z_2, \$) \mapsto (z_2, \$, R)$	$(z_5, c) \mapsto (z_5, c, L)$
$(z_1, a) \mapsto (z_1, a, R)$	$(z_3, c) \mapsto (z_3, c, R)$	$(z_5, \$) \mapsto (z_5, \$, L)$
$(z_1, b) \mapsto (z_2, \$, R)$	$(z_3, \square) \mapsto (z_4, \square, L)$	$(z_5, b) \mapsto (z_5, b, L)$
$(z_1, \$) \mapsto (z_1, \$, R)$	$(z_4, \$) \mapsto (z_4, \$, L)$	$(z_5, a) \mapsto (z_5, a, L)$
$(z_2, b) \mapsto (z_2, b, R)$	$(z_4, \square) \mapsto (z_6, \square, R)$	$(z_5, \square) \mapsto (z_0, \square, R)$
$(z_2, c) \mapsto (z_3, \$, R)$	$(z_4, c) \mapsto (z_5, c, L)$	$(z_0, \$) \mapsto (z_0, \$, R)$

Tabelle: Liste δ der Turingbefehle von M für die Sprache $L = \{a^n b^n c^n \mid n \geq 1\}$

Turingmaschine: Beispiel

Die folgende Tabelle gibt die Bedeutung der einzelnen Zustände von M sowie die mit den einzelnen Zuständen verbundene Absicht an:

Z	Bedeutung	Absicht
z_0	Anfangszustand	neuer Zyklus
z_1	ein a gemerkt	nächstes b suchen
z_2	je ein a, b gemerkt	nächstes c suchen
z_3	je ein a, b, c getilgt	rechten Rand suchen
z_4	rechter Rand erreicht	Zurücklaufen und Test, ob alle a, b, c getilgt
z_5	Test nicht erfolgreich	Zurücklaufen zum linken Rand und neuer Zyklus
z_6	Test erfolgreich	Akzeptieren

Tabelle: Interpretation der Zustände von M

Turingmaschine: Beispiel

Die (deterministische) Konfigurationenfolge von M bei Eingabe von *aabbcc* ist:

$$
\begin{array}{llll}
z_0aabbcc & \vdash_M \$z_1abbcc & \vdash_M \$az_1bbcc & \vdash_M \$a\$z_2bcc \vdash_M \$a\$bz_2cc \\
\$a\$b\$z_3c & \vdash_M \$a\$b\$cz_3\square & \vdash_M \$a\$b\$z_4c & \vdash_M \$a\$bz_5\$c \vdash_M \cdots \\
\$z_5a\$b\$c & \vdash_M z_5\$a\$b\$c & \vdash_M z_5\square\$a\$b\$c & \vdash_M z_0\$a\$b\$c \vdash_M \$z_0a\$b\$c \\
\$\$z_1\$b\$c & \vdash_M \$\$\$z_1b\$c & \vdash_M \$\$\$\$z_2\$c & \vdash_M \$\$\$\$\$z_2c \vdash_M \$\$\$\$\$\$z_3\square \\
\$\$\$\$z_4\$ & \vdash_M \cdots & \vdash_M z_4\$\$\$\$\$ & \vdash_M z_4\square\$\$\$\$\$ \vdash_M z_6\$\$\$\$\$
\end{array}
$$

(*Akzeptieren*)

Linear beschränkte Automaten

- Linear beschränkte Automaten sind spezielle Turingmaschinen, die nie den Bereich des Bandes verlassen, auf dem die Eingabe steht.

- Dazu ist es zweckmäßig, den rechten Rand der Eingabe wie folgt zu markieren (auf dem linken Rand steht der Kopf zu Beginn der Berechnung sowieso und kann diesen im ersten Takt markieren):

  1. Verdoppele das Eingabe-Alphabet Σ zu $\widehat{\Sigma} = \Sigma \cup \{\widehat{a} \,|\, a \in \Sigma\}$.

  2. Repräsentiere die Eingabe $a_1 a_2 \cdots a_n \in \Sigma^+$ durch das Wort

     $$a_1 a_2 \cdots a_{n-1} \widehat{a}_n$$

     über $\widehat{\Sigma}$.

Linear beschränkte Automaten

Definition

- Eine nichtdeterministische TM M heißt *linear beschränkter Automat* (kurz LBA), falls für alle Konfigurationen $\alpha z \beta$ und
  - für alle Wörter $x = a_1 a_2 \cdots a_{n-1} a_n \in \Sigma^+$ mit

$$z_0 a_1 a_2 \cdots a_{n-1} \widehat{a}_n \vdash_M^* \alpha z \beta$$

    gilt: $|\alpha\beta| = n$, und
  - für $x = \lambda$ mit $z_0 \Box \vdash_M^* \alpha z \beta$ gilt: $\alpha\beta = \Box$.
- Die *vom LBA M akzeptierte Sprache* ist definiert durch

$$L(M) = \left\{ a_1 a_2 \cdots a_{n-1} a_n \in \Sigma^* \;\middle|\; \begin{array}{l} z_0 a_1 a_2 \cdots a_{n-1} \widehat{a}_n \vdash_M^* \alpha z \beta \\ \text{mit } z \in F \text{ und } \alpha, \beta \in \Gamma^* \end{array} \right\}.$$

LBA versus Typ-1-Grammatik

Theorem

$L \in \mathrm{CS} \iff L = L(M)$ *für einen LBA* M.

Beweis: $(\Rightarrow)$ Es sei L eine kontextsensitive Sprache und

$$G = (\Sigma, N, S, P)$$

eine Typ-1-Grammatik mit $L(G) = L$.

Wir beschreiben den gesuchten LBA M für L informal wie folgt.

LBA versus Typ-1-Grammatik

1. Eingabe $x = a_1 a_2 \cdots a_n$.

2. Wähle nichtdeterministisch eine Regel $u \to v$ aus P und suche eine beliebiges Vorkommen von v in der aktuellen Bandinschrift von M.

3. Ersetze v durch u. Ist dabei $|u| < |v|$, so verschiebe entsprechend alle Symbole rechts der Lücke, um diese zu schließen.

4.
   - Ist die aktuelle Bandinschrift nur noch das Startsymbol S, so halte im Endzustand und akzeptiere;
   - andernfalls gehe zu (2) und wiederhole.

LBA versus Typ-1-Grammatik

Die so konstruierte Turingmaschine M ist ein LBA, weil alle Regeln in P nichtverkürzend sind.

Es gilt:

$$
\begin{aligned}
x \in L(G) \quad &\Longleftrightarrow \quad \text{es gibt eine Ableitung } S \vdash_G^* x \\
&\Longleftrightarrow \quad \text{es gibt eine Rechnung von } M, \text{ die diese} \\
&\qquad\quad\ \ \text{Ableitung in umgekehrter Reihenfolge simuliert} \\
&\Longleftrightarrow \quad x \in L(M).
\end{aligned}
$$

LBA versus Typ-1-Grammatik

($\Leftarrow$)

- Sei $M = (\Sigma, \Gamma, Z, \delta, z_0, \Box, F)$ ein LBA mit $L(M) = L$.
- Ist x die Eingabe und ist $k \in \mathfrak{K}_M = \Gamma^* Z \Gamma^*$ eine Konfiguration mit

$$z_0 x \vdash_M^* k,$$

so müssen wir sichern, dass gilt:

$$|k| \leq |x|.$$

- Um Konfigurationen durch Wörter der Länge $|x|$ darzustellen, verwenden wir für die zu konstruierende Typ-1-Grammatik G das Alphabet

$$\Delta = \Gamma \cup (Z \times \Gamma).$$

LBA versus Typ-1-Grammatik

- Beispielsweise hat die Konfiguration $k = azbcd$ mit $z \in Z$ und $a, b, c, d \in \Gamma$ über dem Alphabet Δ die Darstellung

$$k' = a(z, b)cd$$

und somit die Länge $4 = |abcd|$.

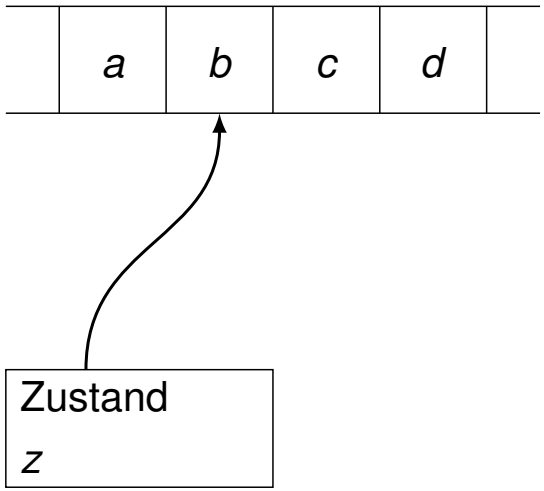

Abbildung: Darstellung von Konfigurationen durch Wörter.

LBA versus Typ-1-Grammatik

- Einen δ-Übergang von M wie etwa

$$(z, a) \to (z', b, L)$$

kann man durch nichtverkürzende Regeln der Form

$$c(z, a) \to (z', c)b$$

für alle $c \in \Gamma$ beschreiben.

- Die Menge aller solcher Regeln der Grammatik heiße P'.
- Es gilt für alle $k_1, k_2 \in \mathfrak{K}_M$:

$$k_1 \vdash_M^* k_2 \iff k_1' \vdash_{G'}^* k_2'$$

wobei k_i', $i \in \{1, 2\}$, die obige Darstellung der Konfiguration k_i bezeichnet und G' die Grammatik mit Regelmenge P' ist.

LBA versus Typ-1-Grammatik

- Definiere $G = (\Sigma, N, S, P)$ so

$$
\begin{aligned}
N &= \{S, A\} \cup (\Delta \times \Sigma); \\
P &= \{S \to A(\hat{a}, a) \mid a \in \Sigma\} \cup && (1)\\
&\quad \{A \to A(a, a) \mid a \in \Sigma\} \cup && (2)\\
&\quad \{A \to ((z_0, a), a) \mid a \in \Sigma\} \cup && (3)\\
&\quad \left\{ (\alpha_1, a)(\alpha_2, b) \to (\beta_1, a)(\beta_2, b) \;\middle|\; \begin{array}{l} \alpha_1\alpha_2 \to \beta_1\beta_2 \in P' \\ \text{für } a, b \in \Sigma \end{array} \right\} \cup && (4)\\
&\quad \{((z, a), b) \to b \mid z \in F, a \in \Gamma, b \in \Sigma\} \cup && (5)\\
&\quad \{(a, b) \to b \mid a \in \Gamma, b \in \Sigma\}. && (6)
\end{aligned}
$$

Offenbar ist G eine Typ-1-Grammatik.

LBA versus Typ-1-Grammatik

Die Idee hinter dieser Konstruktion von G ist die folgende:

- Die Regeln (1), (2) und (3) ermöglichen Ableitungen der Form

$$S \vdash_G^* ((z_0, a_1), a_1)(a_2, a_2) \cdots (a_{n-1}, a_{n-1})(\hat{a}_n, a_n).$$

  Dabei ist

$$((z_0, a_1), a_1)(a_2, a_2) \ldots (a_{n-1}, a_{n-1})(\hat{a}_n, a_n)$$

  ein Wort mit n Buchstaben über dem Alphabet $\Delta \times \Sigma \subseteq N$.
- Jeder Buchstabe ist ein Paar. Dabei stellen
  - die ersten Komponenten die Startkonfiguration dar:

$$z_0 a_1 a_2 \cdots a_{n-1} \hat{a}_n = (z_0, a_1) a_2 \cdots a_{n-1} \hat{a}_n$$

  - und die zweiten Komponenten das Eingabewort

$$x = a_1 a_2 \cdots a_{n-1} a_n.$$

LBA versus Typ-1-Grammatik

- Mit den Regeln der Form (4) simuliert G dann die Rechnung von M bei Eingabe $x = a_1 a_2 \cdots a_{n-1} a_n$, wobei
  - die Regeln aus P' auf die ersten Komponenten der Paare angewandt werden und
  - die zweiten Komponenten unverändert bleiben.

  Die Simulation der Rechnung von $M(x)$ ist beendet, sobald eine Endkonfiguration erreicht ist.

- Regeln der Form (5) und (6) löschen schließlich die ersten Komponenten der Paare weg. Übrig bleiben die zweiten Komponenten, also das akzeptierte Eingabewort x.

- Wird nie eine Endkonfiguration von $M(x)$ erreicht, so kommen die Löschregeln (5) der Grammatik G nie zur Anwendung.

LBA versus Typ-1-Grammatik

- Zusammengefasst folgt aus der obigen Idee formal die Äquivalenzenkette:

$$
\begin{aligned}
x \in L(M) \quad &\Longleftrightarrow \\
S \quad &\vdash_G^* \quad ((z_0, a_1), a_1)(a_2, a_2) \cdots (a_{n-1}, a_{n-1})(\hat{a}_n, a_n) \\
&\qquad\qquad\qquad\qquad \text{mit (1), (2) und (3)} \\
&\vdash_G^* \quad (\gamma_1, a_1) \ldots (\gamma_{k-1}, a_{k-1})((z, \gamma_k), a_k)(\gamma_{k+1}, a_{k+1}) \cdots (\gamma_n, a_n) \\
&\qquad\qquad\qquad\qquad \text{mit (4), wobei } z \in F,\ \gamma_i \in \Gamma,\ a_i \in \Sigma \\
&\vdash_G^* \quad a_1 a_2 \cdots a_n = x \\
&\qquad\qquad\qquad\qquad \text{mit (5) und (6)} \\
&\Longleftrightarrow \quad x \in L(G),
\end{aligned}
$$

womit der Satz bewiesen ist. ❑

TM versus Typ-0-Grammatik

Theorem

$L \in \mathfrak{L}_0 \iff L = L(M)$ *für eine Turingmaschine* M.

Beweis: $(\Rightarrow)$

- Wie im ersten Teil des Beweises des Satzes oben.
- Da G nun nicht nur nichtverkürzende Regeln enthält, erhält man i. A. keinen LBA, sondern eine TM.

TM versus Typ-0-Grammatik

($\Leftarrow$)

- Man kann die gleiche Konstruktion wie im zweiten Teil des Beweises des Satzes oben verwenden.
- Da eine TM keine lineare Beschränkung auf dem Band hat, können in Regeln vom Typ (4) Konfigurationen k mit $|k| > |x|$ aufgebaut werden. Solche Konfigurationen entsprechen Wörtern in G, die Nichtterminale der Form (α, a) mit $a = \lambda$ enthalten.
- Um solche Nichtterminale wieder zu löschen, müssen diese durch λ ersetzt werden, d.h., es werden verkürzende Regeln benötigt.
- Somit erhalten wir i. A. keine kontextsensitive, sondern eine Grammatik vom Typ 0. ❑

TM versus Typ-0-Grammatik

Bemerkung:

- Im Satz oben ist es dabei gleichgültig, ob die TM M deterministisch oder nichtdeterministisch ist.

- Da man jede nichtdeterministische TM durch deterministische TMs simulieren kann, folgt

$$\begin{aligned} \mathfrak{L}_0 &= \{L(M) \mid M \text{ ist eine deterministische TM}\} \\ &= \{L(M) \mid M \text{ ist eine nichtdeterministische TM}\}. \end{aligned}$$

- Im Gegensatz dazu ist es im Satz über LBA versus CS wesentlich, dass der LBA M eine *nicht*deterministische TM ist.

Erstes und zweites LBA-Problem

Bemerkung:

- Bis heute offen ist das

  Erste LBA-Problem: Sind deterministische und nichtdeterministische LBAs äquivalent?

- Hingegen ist das ebenfalls 1964 von Kuroda gestellte

  Zweite LBA-Problem: Ist die Klasse der durch nichtdeterministische LBAs akzeptierbaren Sprachen komplementabgeschlossen?

  inzwischen gelöst worden, und zwar unabhängig und etwa zeitgleich 1988 von Neil Immerman und Robert Szelepcsényi.

Charakterisierungen durch Automaten

Typ 3	reguläre Grammatik deterministischer endlicher Automat (DFA) nichtdeterministischer endlicher Automat (NFA) regulärer Ausdruck
deterministisch kontextfrei	LR(1)-Grammatik deterministischer Kellerautomat (DPDA)
Typ 2	kontextfreie Grammatik Kellerautomat (PDA)
Typ 1	kontextsensitive Grammatik linear beschränkter Automat (LBA)
Typ 0	Typ-0-Grammatik Turingmaschine (NTM bzw. DTM)

Determinismus vs. Nichtdeterminismus

Deterministischer Automat	Nichtdeterministischer Automat	äquivalent?
DFA	NFA	ja
DPDA	PDA	nein
DLBA	LBA	?
DTM	NTM	ja

Abschlusseigenschaften

	Typ 3	det. kf.	Typ 2	Typ 1	Typ 0
Schnitt	ja	nein	nein	ja	ja
Vereinigung	ja	nein	ja	ja	ja
Komplement	ja	ja	nein	ja	nein
Konkatenation	ja	nein	ja	ja	ja
Iteration	ja	nein	ja	ja	ja
Spiegelung	ja	nein	ja	ja	ja

Komplexität des Wortproblems

Definition (Wortproblem)

Für $i \in \{0, 1, 2, 3\}$ definieren wir das *Wortproblem für Typ-i-Grammatiken* wie folgt:

$$\text{Wort}_i = \{(G, x) \mid G \text{ ist Typ-}i\text{-Grammatik und } x \in L(G)\}$$

Typ 3 (DFA gegeben)	lineare Komplexität
det. kf.	lineare Komplexität
Typ 2 (CNF gegeben)	Komplexität $\mathcal{O}(n^3)$ (CYK-Algorithmus)
Typ 1	exponentielle Komplexität
Typ 0	unentscheidbar (d.h. algorithmisch nicht lösbar)