Theoretische Informatik
Kapitel 5 – Kontextsensitive und $\mathcal{L}_0$-Sprachen

Sommersemester 2024

Dozentin: Mareike Mutz

im Wechsel mit

Prof. Dr. M. Leuschel

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

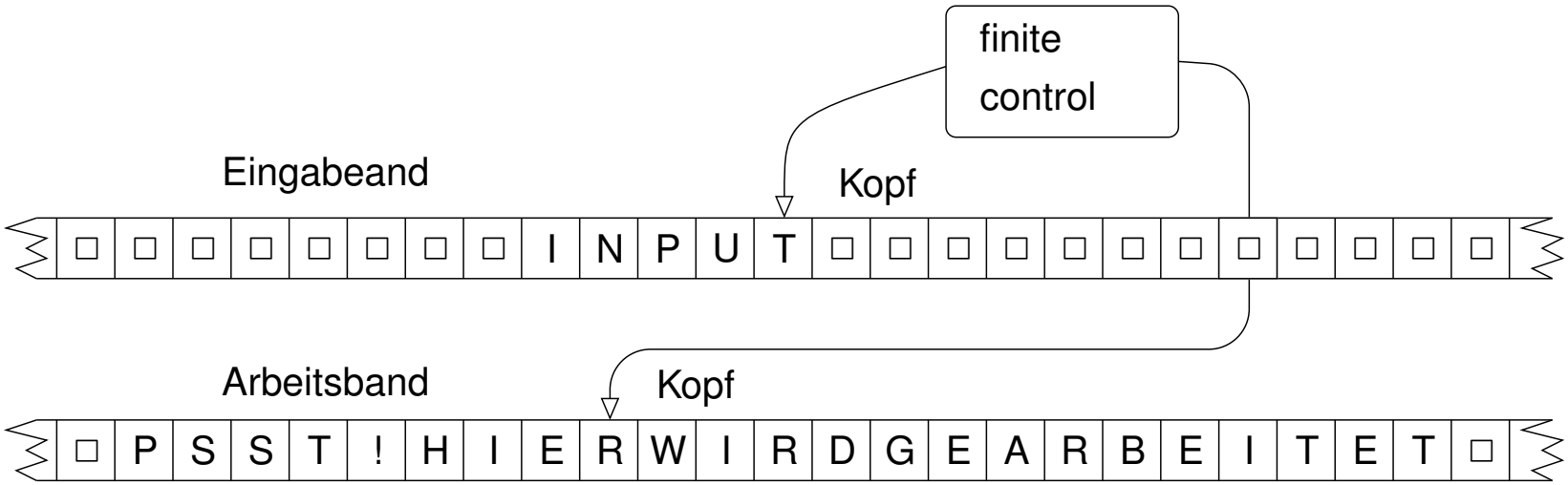


Abbildung: Eine Turingmaschine

- 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 $M = (\Sigma, \Gamma, Z, \delta, z_0, \square, F)$, 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,
- $\square \in \Gamma - \Sigma$ 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, in das die Eingabe bisher transofrmiert wurde),
  - 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: einfaches Beispiel

$M = (\{a, b, c\}, \{a, b, c, \square\}, \{z_0, z_1, z_2, z_3\}, \delta, z_0, \square, \{z_3\})$ mit δ:

$z_0 a \to z_0 aR \quad z_0 b \to z_1 bR \quad z_0 b \to z_2 bR$

$z_1 b \to z_1 bR \quad z_1 \square \to z_3 \square N$

$z_2 c \to z_0 cR$

Schritt	links	z	rechts	Schritt	links	z	rechts
1	λ	z_0	aabcabb	6	aabca	z_0	bb
2	a	z_0	abcabb	7	aabcab	z_1	b
3	aa	z_0	bcabb	8	aabcabb	z_1	$\square$
4	aab	z_2	cabb	9	aabcabb	z_3	$\square$
5	aabc	z_0	abb				

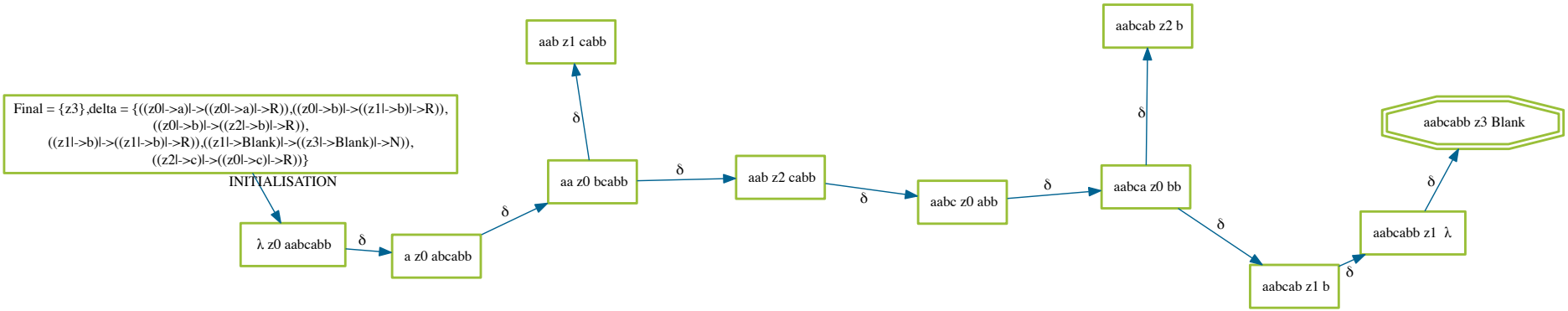
Turingmaschine: Beispiel - Berechnungsbaum

$M = (\{a, b, c\}, \{a, b, c, \square\}, \{z_0, z_2, z_2\}, \delta, z_0, \square, \{z_3\})$ mit δ:

$z_0 a \to z_0 aR \quad z_0 b \to z_1 bR \quad z_0 b \to z_2 bR$

$z_1 b \to z_1 bR \quad z_1 \square \to z_3 \square N \quad z_2 c \to z_0 cR$

Berechnungsbaum für Startkonfiguration = $\lambda\ z_0$ aabcabb



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

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

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

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:

$$
\begin{array}{llllll}
z_0 aabbcc & \vdash_M Az_1 abbcc & \vdash_M Aaz_1 bbcc & \vdash_M AaBz_2 bcc & \vdash_M AaBbz_2 cc & \vdash_M \\
AaBbCz_3 c & \vdash_M AaBbCcz_3\square & \vdash_M AaBbCz_4 c & \vdash_M AaBbz_5 C & \vdash_M \cdots & \vdash_M \\
Az_5 aBbCc & \vdash_M z_5 AaBbCc & \vdash_M z_5\square AaBbCc & \vdash_M z_0 AaBbCc & \vdash_M Az_0 aBbCc & \vdash_M \\
AAz_1 BbCc & \vdash_M AABz_1 bCc & \vdash_M AABBz_2 Cc & \vdash_M AABBCz_2 c & \vdash_M AABBCCz_3\square & \vdash_M \\
AABBCz_4 C & \vdash_M \cdots & \vdash_M z_4 AABBCC & \vdash_M z_4\square AABBCC & \vdash_M z_6 AABBCC &
\end{array}
$$

(*Akzeptieren*)

Turingmaschine: Beispiel *aabbcc* komplett

Schritt	links	z	rechts	Schritt	links	z	rechts
1	λ	z_0	aabbcc	15	$\square$	z_0	AaBbCc$\square$
2	A	z_1	abbcc	16	$\square$A	z_0	aBbCc$\square$
3	Aa	z_1	bbcc	17	$\square$AA	z_1	BbCc$\square$
4	AaB	z_2	bcc	18	$\square$AAB	z_1	bCc$\square$
5	AaBb	z_2	cc	19	$\square$AABB	z_2	Cc$\square$
6	AaBbC	z_3	c	20	$\square$AABBC	z_2	c$\square$
7	AaBbCc	z_3	$\square$	21	$\square$AABBCC	z_3	$\square$
8	AaBbC	z_4	c$\square$	22	$\square$AABBC	z_4	C$\square$
9	AaBb	z_5	Cc$\square$	23	$\square$AABB	z_4	CC$\square$
10	AaB	z_5	bCc$\square$	24	$\square$AAB	z_4	BBC$\square$
11	Aa	z_5	BbCc$\square$	25	$\square$AA	z_4	BBCC$\square$
12	A	z_5	aBbCc$\square$	26	$\square$A	z_4	ABBCC$\square$
13	λ	z_5	AaBbCc$\square$	27	$\square$	z_4	AABBCC$\square$
14	λ	z_5	$\square$ AaBbCc$\square$	28	λ	z_4	$\square$ AABBCC$\square$
				29	$\square$	z_6	AABBCC$\square$

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 $\widehat{\Sigma} = \Sigma \cup \{\widehat{a} \,\big|\, a \in \Sigma\}$.

  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:

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

Turingmaschine: Beispiel ksG nach TM (alternativ)

Beispiel kSG G = $(\Sigma, \{S, T\}, S, \{S \to aTb, aT \to ab\})$. Leicht abgeänderte Übersetzung: anstatt Verschieben, werden wir leere Zellen mit E markieren. Wir bauen auch streng genommen keinen LBA (wir erlauben eine Verschiebung mit Blanks).

$\Gamma = \{a, b, S, T, E, \square\}$ $Z = \{z0, z1, z11, z2, z3, z4, z5, z6\}$, $F = \{z6\}$.

$\delta = \{$ $(z0, a) \mapsto (z1, S, R)$, $(z0, a) \mapsto (z2, a, R)$, $(z0, a) \mapsto (z0, a, R)$, $(z0, b) \mapsto (z0, b, R)$,
$(z0, S) \mapsto (z0, S, R)$, $(z0, T) \mapsto (z0, T, R)$,
$(z1, T) \mapsto (z11, E, R)$, $(z1, E) \mapsto (z1, E, R)$, $(z11, b) \mapsto (z3, E, L)$,
$(z2, b) \mapsto (z3, T, L)$, $(z2, E) \mapsto (z2, E, R)$,
$(z3, a) \mapsto (z3, a, L)$, $(z3, b) \mapsto (z3, b, L)$, $(z3, S) \mapsto (z3, S, L)$, $(z3, T) \mapsto (z3, T, L)$,
$(z3, E) \mapsto (z3, E, L)$, $(z3, Blank) \mapsto (z0, Blank, R)$, $(z3, Blank) \mapsto (z4, Blank, R)$,

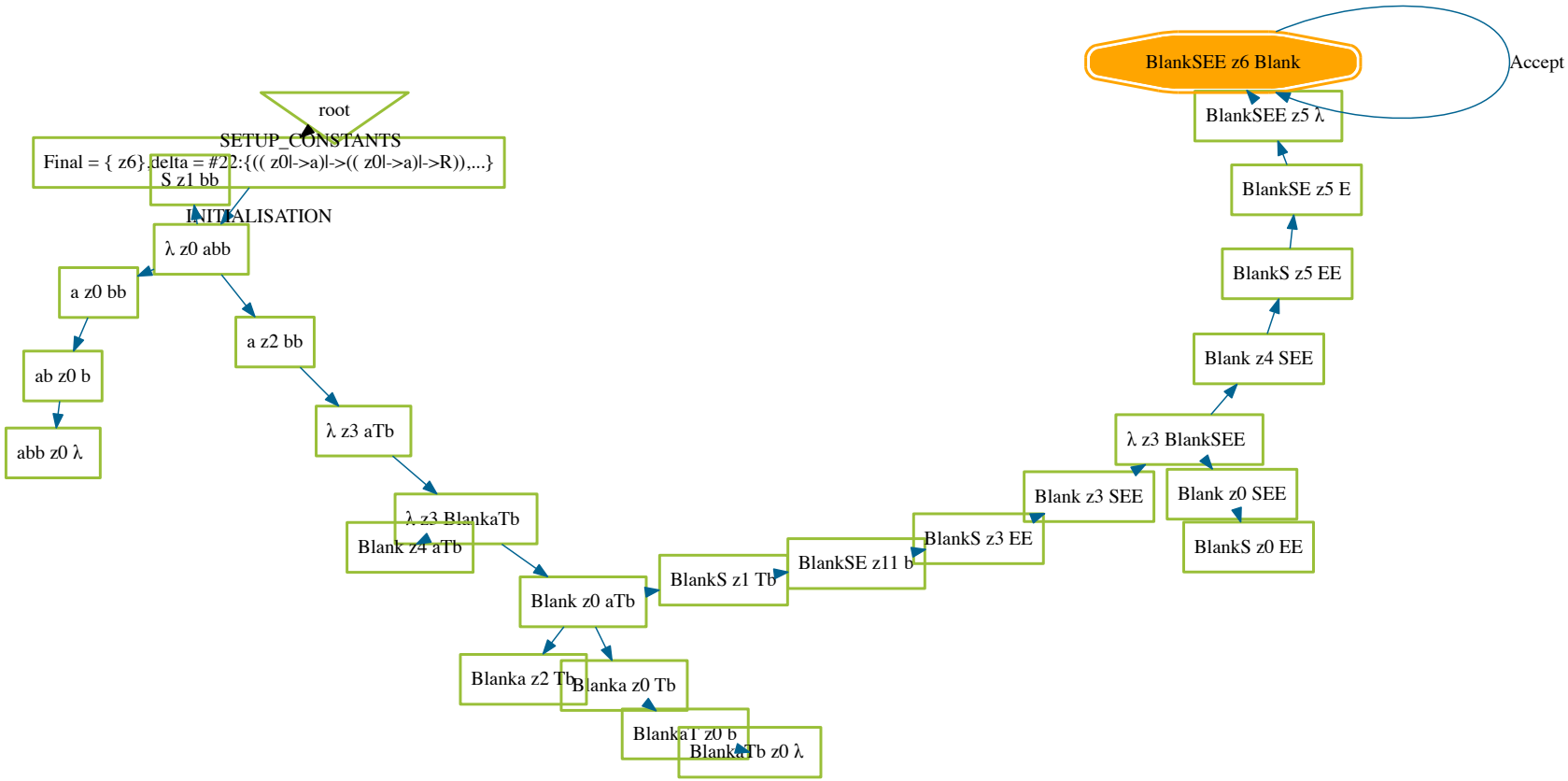
$(z4, E) \mapsto (z4, E, R)$, $(z4, S) \mapsto (z5, S, R)$, $(z5, E) \mapsto (z5, E, R)$, $(z5, Blank) \mapsto (z6, Blank, N)$

$\}$

Wie müsste dieser Automat geändert werden, damit hier wie gewünscht ein LBA entsteht?

Turingmaschine: Beispiel ksG nach TM

Beispiel kSG G = $(\Sigma, \{S, T\}, S, \{S \to aTb, aT \to ab\})$. Leicht abgeänderte Übersetzung: anstatt Verschieben, werden wir leere Zellen mit E markieren. Wir bauen auch streng genommen keinen LBA (wir erlauben eine Verschiebung mit Blanks).
Hier ist der Berechnungsbaum für die Eingabe abb:



Turingmaschine: Beispiel ksG nach TM

Beispiel kSG G = $(\Sigma, \{S, T\}, S, \{S \to aTb, aT \to ab\})$. Leicht abgeänderte Übersetzung: anstatt Verschieben, werden wir leere Zellen mit E markieren. Wir bauen auch streng genommen keinen LBA (wir erlauben eine Verschiebung mit Blanks).
Hier ist ein erfolgreicher Ausführungspfad für die Eingabe abb.

Schritt	links	z	rechts	Schritt	links	z	rechts
1	λ	$z0$	abb	17	$\square$	$z3$	SEE
3	a	$z2$	bb	19	λ	$z3$	$\square SEE$
5	λ	$z3$	aTb	21	$\square$	$z4$	SEE
6	λ	$z3$	$\square aTb$	22	$\square S$	$z5$	EE
8	$\square$	$z0$	aTb	23	$\square SE$	$z5$	E
12	$\square S$	$z1$	Tb	25	$\square SEE$	$z5$	$\square$
15	$\square SE$	$z11$	b	26	$\square SEE$	$z6$	$\square$
16	$\square S$	$z3$	EE				

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 $4 = |abcd|$.

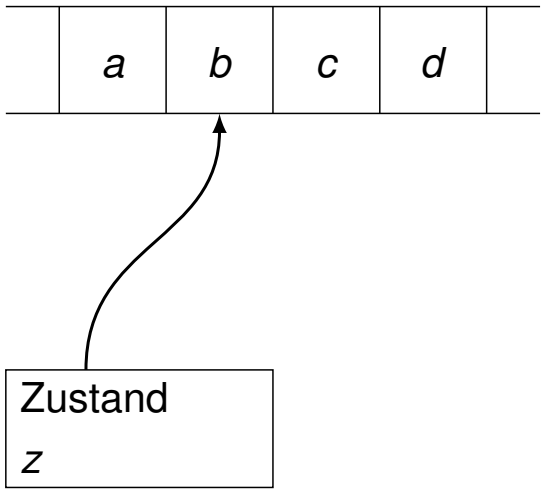


Abbildung: Darstellung von Konfigurationen durch Wörter.

LBA versus Typ-1-Grammatik

- Einen δ-Übergang von $(z, a) \to (z', b, N)$ in M stellt man als Regel $(z, a) \to (z', b)$ dar.
- 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.

- Einen δ-Übergang von $(z, a) \to (z', b, R)$ stellt man als Regeln $(z, a)c \to b(z', c)$ für alle $c \in \Gamma$ beschreiben.

LBA versus Typ-1-Grammatik

- 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.
- Wir brauchen jetzt ein paar "kleine Tricks" um, den Initialzustand des LBAs aufzubauen, die Kodierung des letzten Buchstabens ($\hat{a}$) zu behandeln, und bei Erreichen des Endzustands den Zustand wegzuwerfen (damit nur Buchstaben in Σ übrigbleiben).

TM versus Typ-1-Grammatik - Beispiel

Leicht angepasste Version von früherem Beispiel (ist formal kein LBA, da die TM $\Box$ rechts liest): $M =$
$(\{a, b\}, \{a, b, \Box\}, \{z_0, z_1, z_2, z_3\}, \delta, z_0, \Box, \{z_3\})$ mit δ:

$$z_0 a \to z_0 aR \quad z_0 b \to z_1 bR \quad z_0 b \to z_2 bR$$

$$z_1 b \to z_1 bR \quad z_1 \Box \to z_3 \Box N \quad z_2 a \to z_0 aR$$

So soll sich die Grammatik im Vergleich zur TM verhalten (abgesehen von $\Box$, wo wir $ab\hat{ab}$ anstatt $abab\Box$ benutzen müssen):

#	links	z	rechts	CSG	#	l.	z	r.	CSG
1	λ	z_0	abab	$(z_0, a)bab$	5	abab	z_1	$\Box$	$abab(z_1, \Box)$
2	a	z_0	bab	$a(z_0, b)ab$	6	abab	z_3	$\Box$	$abab(z_3, \Box)$
3	ab	z_2	ab	$ab(z_2, a)b$					
4	aba	z_0	b	$aba(z_0, b)$					

TM versus Typ-1-Grammatik - Beispiel

δ von M: $z_0a \to z_0aR$ $z_0b \to z_1bR$ $z_0b \to z_2bR$
$z_1b \to z_1bR$ $z_1\Box \to z_3\Box N$ $z_2a \to z_0aR$

Versuch einer Grammatik G mit $\Sigma = \{a, b, \Box\}$, N= $\{S, S_1, (z_0, a), \ldots, (z_2, \Box)\}$, Regeln R=
$\{ S \mapsto S_1, \Box$, $S_1 \mapsto S_1a$, $S_1 \mapsto S_1b$, $S_1 \mapsto (z_0, a)$, $S_1 \mapsto (z_0, b)$, $S_1 \mapsto (z_0\Box)$,
$(z_0, a)a \mapsto a(z_0, a)$, $(z_0, a)b \mapsto a(z_0, b)$, $(z_0, a)\Box \mapsto a(z_0, \Box)$,
$(z_0, b)a \mapsto b(z_1, a)$, $(z_0, b)b \mapsto b(z_1, b)$, $(z_0, b)\Box \mapsto b(z_1, \Box)$,
$(z_0, b)a \mapsto b(z_2, a)$, $(z_0, b)b \mapsto b(z_2, b)$, $(z_0, b)\Box \mapsto b(z_2, \Box)$,
$(z_1, b)a \mapsto b(z_1, a)$, $(z_1, b)b \mapsto b(z_1, b)$, $(z_1, b)\Box \mapsto b(z_1, \Box)$,
$(z_1, \Box) \mapsto (z_3, \Box)$,
$(z_2, a)a \mapsto a(z_0, a)$, $(z_2, a)b \mapsto a(z_0, b)$, $(z_2, a)\Box \mapsto a(z_0, \Box)$ $\}$

Ableitung: $S \vdash_G S_1\Box \vdash_G S_1b\Box \vdash_G S_1ab\Box \vdash_G S_1bab\Box \vdash_G (z_0, a)bab\Box \vdash_G a(z_0, b)ab\Box \vdash_G ab(z_2, a)b\Box \vdash_G aba(z_0, b)\Box \vdash_G abab(z_1, \Box) \vdash_G abab(z_3, \Box)$

Was fehlt noch: Unterstützung von $\hat{a}$, $\hat{b}$, $abab$ ableiten und nicht $abab(z_3, \Box)$. Bei einer Regel wie $z_1b \to z_1cR$ in δ müssen wir auch zwischen dem neuen Bandinhalt c und dem akzeptierten Wort mit b unterscheiden.

LBA versus Typ-1-Grammatik

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

$$
\begin{aligned}
N &= \{S, A\} \cup (\Delta \times \Sigma); \\
P &= \{S \to A(\hat{a}, a) \mid a \in \Sigma\} \cup && (1) \\
&\quad \{A \to A(a, a) \mid a \in \Sigma\} \cup && (2) \\
&\quad \{A \to ((z_0, a), a) \mid a \in \Sigma\} \cup && (3) \\
&\quad \left\{ (\alpha_1, a)(\alpha_2, b) \to (\beta_1, a)(\beta_2, b) \;\middle|\; \begin{array}{l} \alpha_1\alpha_2 \to \beta_1\beta_2 \in P' \\ \text{für } a, b \in \Sigma \end{array} \right\} \cup && (4) \\
&\quad \{(\alpha_1, a) \to (\beta_1, a) \mid \alpha_1 \to \beta_1 \in P' \text{ für } a, b \in \Sigma\} \cup && (5) \\
&\quad \{((z, a), b) \to b \mid z \in F, a \in \Gamma, b \in \Sigma\} \cup && (6) \\
&\quad \{(a, b) \to b \mid a \in \Gamma, b \in \Sigma\}. && (7)
\end{aligned}
$$

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, 5) 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 (6) und (7) 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 (6) der Grammatik G nie zur Anwendung.

LBA versus Typ-1-Grammatik

- Zusammengefasst folgt aus der obigen Idee formal die Äquivalenzenkette:

$$
\begin{aligned}
x \in L(M) \quad &\Longleftrightarrow \\
S \quad &\vdash_G^* \quad ((z_0, a_1), a_1)(a_2, a_2) \cdots (a_{n-1}, a_{n-1})(\hat{a}_n, a_n) \\
&\qquad\qquad\qquad\qquad \text{mit (1), (2) und (3)} \\
&\vdash_G^* \quad (\gamma_1, a_1) \ldots (\gamma_{k-1}, a_{k-1})((z, \gamma_k), a_k)(\gamma_{k+1}, a_{k+1}) \cdots (\gamma_n, a_n) \\
&\qquad\qquad\qquad\qquad \text{mit (4), wobei } z \in F,\ \gamma_i \in \Gamma,\ a_i \in \Sigma \\
&\vdash_G^* \quad a_1 a_2 \cdots a_n = x \\
&\qquad\qquad\qquad\qquad \text{mit (6) und (7)} \\
&\Longleftrightarrow \quad x \in L(G),
\end{aligned}
$$

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, wobei man bei verkürzenden Regeln Symbole nach rechts verschieben muss.
- 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, man muss nur $\square$ berücksichtigen und kann sich die Behandlung von $\hat{a}$ sparen.
- 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)