Hans U. Simon
Annette Ilgen

Bochum, den 05.02.2009

Beispiele zur Vorlesung

Theoretische Informatik

WS 08/09

Vorbemerkung: Hier findet sich eine Sammlung von Beispielen und Motivationen zur Vorlesung Theoretische Informatik.

4 Kontextsensitive und Typ 0 Sprachen

4.1 Deterministische Turingmaschine

Wir wollen eine Turingmaschine M konstruieren, die gerade Binärzahlen durch 2 dividiert. Sie besteht aus folgenden Komponenten. $\Sigma = \{0, 1\}$
$Z = \{z_0, z_1, z_e\}$
$\Gamma = \{0, 1, \square\}$
$z_0 =$ Startzustand
$\square =$ Blank
$E = \{z_e\}$

Die Zustände sind dabei wie folgt zu interpretieren.
z_0: Suche das Bit am weitesten rechts
z_1: Falls es sich um eine 0 handelt lösche diese
z_e: Stoppe

Die Funtion $\delta : Z \times \Gamma \to Z \times \Gamma \times \{L, R, N\}$ wird in Form einer Turing-Tafel angegeben.

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

Betrachten wir die Konfigurationen der Turingmaschine bei Bearbeitung der Eingabe 110.

$$z_0110 \vdash 1z_010 \vdash 11z_00 \vdash 110z_0 \vdash 11z_10 \vdash 11z_e$$

Die Turingmaschine gelangt in einen Endzustand. Weitere Schritte sind nicht definiert. Das Wort wird akzeptiert, da es durch 2 teilbar ist.

Betrachten wir die Konfigurationen der Turingmaschine bei Bearbeitung der Eingabe 111.

$$z_0111 \vdash 1z_011 \vdash 11z_01 \vdash 111z_0 \vdash 11z_11$$

Für den Zustand z_1 ist keine Folgekonfiguration definiert. Die Maschine stoppt und verwirft das Wort, da sie sich nicht in einem Endzustand befindet.

Die Sprache die diese Turingmaschine erkennt ist die Menge aller geraden Binärzahlen.

4.2 Nichtdeterministische Turingmaschine

Wir geben ein Beispiel für eine nichtdeterministische Turingmaschine, die ebenso wie die Turingmaschine aus Beispiel 4.1 gerade Binärzahlen durch 2 teilt. Die Komponenten der NTM sind die gleichen wie im Beispiel 4.1. Die Turing-Tafel erlaubt nun jedoch mehrere mögliche Folgekonfigurationen in einem Schritt.

δ	0	1	$\square$
z_0	$\{(z_0, 0, R), (z_1, \square, R)\}$	$\{(z_0, 1, R)\}$	
z_1			$\{(z_e, \square, N)\}$

Die Zustände sind dabei wie folgt zu interpretieren.
z_0: Gehe nach rechts, wenn dabei eine 0 gefunden wird, darf sie gelöscht werden; Folgezustand ist dann z_1
z_1: Falls die NTM in einem Blank landet hat sie die "richtige" 0 gelöscht
z_e: Stoppe

Die Sprache ist auch hier die Menge aller geraden Binärzahlen, denn nachem die Maschine eine Null gelöscht hat, geht sie in Zustand $z_1$über. Dieser prüft ob das nächste Zeichen ein Blank ist, denn nur dann befand sich die gelöschte Null am Ende der Eingabe und nur dann gelangen wir in einen Endzustand.

4.3 Haltebereich einer Turingmaschine

Betrachte die DTM M aus Beispiel 4.1. Ihr Haltebereich ist

$$H(M) = \Sigma^+$$

Beachte, dass auf dem Band der TM nur Eingaben aus $\{0, 1\}^*$ erlaubt sind und Blanks nur links und rechts davon auftreten. Die TM stoppt auf jeder korrekten Eingabe, mit Ausnahme des leeren Wortes. Bei diesem gelangt sie mit $\delta(z_1, \square) = (z_1, \square, L)$ in eine Endlosschleife.

4.4 Linear beschränkter Automat

4.4.1 Beispiel 1

Die Turingmaschine aus Beispiel 4.1 lässt sich auch mit einem DLBA simulieren. Wir "verdoppeln" das Eingabealphabet durch die markierten Zeichen und erhalten $\Sigma = \{0, 1, \hat{0}, \hat{1}\}$

Ein LBA setzt voraus, dass das Eingabewort am Ende bereits markiert ist. Wir benötigen nur noch zwei Zustände, da wir das Ende der Eingabe nicht mehr suchen müssen.

δ	0	1	$\hat{0}$	$\hat{1}$
z_0	$(z_0, 0, R)$	$(z_0, 1, R)$	$(z_e, \square, N)$	

4.4.2 Beispiel 2

Betrachte die Sprache $L = \{a^n b^n c^n \in \{a, b, c\}^* | n \geq 1\}$, die von folgendem DLBA erkannt wird.

$Z = \{z_0, \hat{z_a}, z_a, \hat{z_b}, z_b\, z_r, z_e\}$
$E = \{z_e\}$
$z_0 =$ Startzustand
$\Gamma = \{a, b, c, \hat{a}, \hat{c}, \#\}$

δ	$\#$	a	b	c	$\hat{a}$	$\hat{c}$
z_0		$(\hat{z_a}, \#, R)$				
$\hat{z_a}$	$\rightarrow$	$(z_a, \hat{a}, R)$	$(\hat{z_b}, \#, R)$			
z_a	$\rightarrow$	$\rightarrow$	$(z_b, \#, R)$			
$\hat{z_b}$	$\rightarrow$					$(z_e, \#, N)$
z_b	$\rightarrow$		$\rightarrow$	$(z_r, \#, L)$		
z_r	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$(\hat{z_a}, \#, R)$	

Dabei stehen die Pfeile $\rightarrow$ bzw. $\leftarrow$ abkürzend für Rechenschritte der Form $\delta(z, \gamma) = (z, \gamma, R)$ bzw $\delta(z, \gamma) = (z, \gamma, L)$ in denen der Zustand und der Eintrag auf dem Band gleich bleiben und nur ein Schritt nach rechts bzw. links gegangen wird.

Arbeitsweise: Der DLBA geht das Eingabewort von links nach rechts durch und löscht dabei je das erste a, b und c in dieser Reihenfolge. Dabei wird stets der aktuell relevante Beginn der Eingabe neu mit einem Dach markiert. Am Ende der Eingabe wird er wieder auf den Anfang der Eingabe zurückgesetzt und beginnt erneut, bis alle Buchstaben gelöscht wurden.

Der DLBA arbeitet dabei in seinen Zuständen wie folgt.
z_0: Im Startzustand z_0 wird das erste a durch ein # gelöscht, andere Buchstaben werden am Anfang der Eingabe nicht akzeptiert.
$\hat{z_a}$: Es wurde bereits ein a gefunden, jedoch muss der Anfang der Eingabe noch mit einem Dach markiert werden. Gesucht wird jedoch bereits das nächste b. Dieses wird durch ein # gelöscht wenn es gefunden wird.
z_a: Es wurde bereits ein a gefunden und der Anfang der Eingabe wurde bereits mit einem Dach markiert. Gesucht wird das nächste b. Dieses wird durch ein # gelöscht wenn es gefunden wird.
$\hat{z_b}$: Es wurde bereits ein a und ein b gefunden und gelöscht, jedoch wurde der Anfang der Eingabe noch nicht mit einem Dach markiert. Wird also ein $\hat{c}$ gefunden, so ist es das letzte vorhandene c und die Maschine geht in einen akzeptierenden Endzustand über.
z_b: Es wurde bereits ein a und ein b gefunden und gelöscht, der Anfang der Eingabe wurde bereits mit einem Dach markiert. Gesucht wird das nächste c. Dieses wird durch ein # gelöscht wenn es gefunden wird. Danach geht die Maschine in den Zurücksetzungszustand

z_r über.
z_r: Der Lesekopf wird zurückgesetzt bis zum mit Dach markierten ersten Buchstaben der Eingabe. Dieses muss ein $\hat{a}$ sein.

4.5 Vom LBA zur kontextsensitiven Grammatik

Wir betrachten nun den DLBA aus Beispiel 4.4.2.

Analog zum Beweis des Satzes von Kuroda erstellen wir eine Grammatik, deren Variablen zwei Komponenten haben. In der ersten Komponente wird die Rechnung des LBA simuliert, während die zweite Komponente die Eingabe des LBA beibehält. Wird in der ersten Komponente ein Endzustand erreicht, so gibt es in der Grammatik Regeln mit deren Hilfe nun die erste Komponente gelöscht wird. Übring bleiben die Eingabe welche aus Zeichen des Terminalalphabetes Σ besteht.

Wir wollen nun eine Ableitung für das Wort $abc \in L.$ in dieser Grammatik angeben. Dazu verwenden wir folgende Anfangsregeln

$$S \to T(\hat{c}, c), \quad T \to T(b, b), \quad T \to ((z_0, a), a)$$

und erhalten

$$S \Rightarrow T(\hat{c}, c) \Rightarrow T(b, b)(\hat{c}, c) \Rightarrow ((z_0, a), a)(b, b)(\hat{c}, c)$$

Dieser Ausdruck repräsentiert nun in der ersten Komponente die Rechnung des LBA. In der zweiten Komponente wird die Eingabe des LBA persistiert. Zu Beginn der Rechnung steht also in der ersten Komponente Startzustand z_0 auf dem ersten Zeichen der Eingabe. In der zweiten Komponente ist eine Kopie der Eingabe notiert.

Als nächstes betrachten wir die Rechenschritte welche der LBA auf abc ausfürt

$$\begin{array}{ll} \delta(z_0, a) = (\hat{z_a}, \#, R) & z_0 ab\hat{c} \vdash \#\hat{z_a} b\hat{c} \\ \delta(\hat{z_a}, b) = (\hat{z_b}, \#, R) & \#\hat{z_a} b\hat{c} \vdash \#\#\hat{z_b}\hat{c} \\ \delta(\hat{z_b}, \hat{c}) = (z_e, \#, N) & \#\#\hat{z_b}\hat{c} \vdash \#\#z_e\# \end{array}$$

Daraus resultieren folgende Regeln in P'

$$\begin{array}{ll} (z_0, a)\gamma \to \#(\hat{z_a}, \gamma) & \forall \gamma \in \Gamma \\ (\hat{z_a}, b)\gamma \to \#(\hat{z_b}, \gamma) & \forall \gamma \in \Gamma \\ (\hat{z_b}, \hat{c}) \to (z_e, \#) & \end{array}$$

Durch hinzufügen der zweiten Komponente, erhalten wir in P die Regeln

$$\begin{array}{ll} ((z_0, a), \sigma_1)(\gamma, \sigma_2) \to (\#, \sigma_1)((\hat{z_a}, \gamma), \sigma_2) & \forall \gamma \in \Gamma, \text{ und } \forall \sigma_1, \sigma_2 \in \Sigma \\ ((\hat{z_a}, b), \sigma_1)(\gamma, \sigma_2) \to (\#, \sigma_1)((\hat{z_b}, \gamma), \sigma_2) & \forall \gamma \in \Gamma, \text{ und } \forall \sigma_1, \sigma_2 \in \Sigma \\ ((\hat{z_b}, \hat{c}), \sigma_1) \to ((z_e, \#), \sigma_1) & \forall \sigma_1 \in \Sigma \end{array}$$

Daraus wählen wir nun für die γ, σ_1 und σ_2 die wir benötigen folgende Regeln aus

$$\begin{array}{l} ((z_0, a), a)(b, b) \to (\#, a)((\hat{z_a}, b), b) \\ ((\hat{z_a}, b), b)(\hat{c}, c) \to (\#, b)((\hat{z_b}, \hat{c}), c) \\ ((\hat{z_b}, \hat{c}), c) \to ((z_e, \#), c) \end{array}$$

und wenden sie auf das Wort $((z_0, a), a)(b, b)(\hat{c}, c)$ an.

$$((z_0, a), a)(b, b)(\hat{c}, c) \Rightarrow (\#, a)((\hat{z_a}, b), b)(\hat{c}, c) \Rightarrow (\#, a)(\#, b)((\hat{z_b}, \hat{c}), c) \Rightarrow (\#, a)(\#, b)((z_e, \#), c)$$

Wir sehen, wie auf der ersten Komponente die Rechnung des LBA ausgefürt wird, der alle Buchstaben durch # löscht und in den Endzustand z_e übergeht. Die Eingabe in der zweiten Komponente bleibt davon unberührt.

In P gibt es nun die Schlussregeln

$$\begin{array}{ll} (\gamma, \sigma) \to \sigma & \forall \gamma \in \Gamma, \sigma \in \Sigma \\ ((z_e, \gamma), \sigma) \to \sigma & \forall \gamma \in \Gamma, \sigma \in \Sigma \end{array}$$

Sie löschen die erste Komponeten in der Satzform und somit verbleiben nur noch Terminalzeichen aus Σ
Wir wenden diese an und erhalten

$$(\#, a)(\#, b)((z_e, \#), c) \Rightarrow^* abc$$

Damit konnte das Wort abc aus der Startvariablen S abgeleitet werden.

4.6 Abschlusseigenschaften

Kontextsensitive Sprachen sind unter den Operationen $\cup, \cap, \cdot, *, \neg$ abgeschlossen. Betrachte die Sprache $L_{ab} := \{w \in \{a, b, c\}^* |\ |w|_a = |w|_b\}$. Sie ist deterministisch Kontextfrei, somit also auch kontextsensitiv.

- Kontextsensitive Sprachen sind abgeschlossen unter Schnittmengenbildung.
  Betrachte die Sprache $L_{abc} := \{w \in \{a, b, c\}^* |\ |w|_a = |w|_b = |w|_c\}$. Sie ist gleich mit $L_{ab} \cap L_{bc} = \{w \in \{a, b, c\}^* |\ |w|_a = |w|_b$ und $|w|_b = |w|_c\}$. L_{abc} ist nicht kontextfrei aber kontextsensitiv.

- Kontextsensitive Sprachen sind abgeschlossen unter Vereinigung.
  Da $L = \{a^n b^n c^n \in \{a, b, c\}^* | n \geq 0\}$ kontextsensitiv ist, ist auch $L_0 = \{a^n b^n c^n \in \{a, b, c\}^* | n \geq 0\} \cup \{b^n a^n c^n \in \{a, b, c\}^* | n \geq 0\}$ kontextsensitiv.

- Kontextsensitive Sprachen sind Abgeschlossen unter Komplementbildung.
  Somit muss auch die Sprache $\overline{L_{abc}} = \{w \in \{a, b, c\}^* |\ |w|_a \neq |w|_b$ oder $|w|_a \neq |w|_c\}$ kontextsensitiv sein.