

Theoretische Informatik I

Einheit 4.2

Modelle für Typ-0 & Typ-1 Sprachen



1. Nichtdeterministische Turingmaschinen
2. Äquivalenz zu Typ-0 Sprachen
3. Linear beschränkte Automaten
und Typ-1 Sprachen

MASCHINENMODELLE VS. GRAMMATIKEN

- **Ableitbarkeit $w \longrightarrow_G z$ ist nichtdeterministisch**
 - In w können verschiedene Teilworte ersetzt werden
 - Auf ein Teilwort können verschiedene Regeln angewandt werden
 - Simulation erfordert nichtdeterministisches Maschinenmodell
- **Maschinenmodelle sind i.a. deterministisch**
 - Nichtdeterministische Modelle sind “unrealistisch” und nur für elegantere Modellierung geeignet
 - Nichtdeterministische Modelle sind evtl. deterministisch simulierbar aber nur mit exponentiellem Aufwand
- **Verwende nichtdeterministische Turingmaschinen**
 - “Simultane” Behandlung vieler alternativer Konfigurationen
 - Zeige Äquivalenz zu deterministischen Turingmaschinen
 - Zeige Äquivalenz zu Typ-0 Grammatiken
 - Zeige Äquivalenz zu Typ-1 Grammatiken für eingeschränktes Modell

NICHTDETERMINISTISCHE TURINGMASCHINEN

- Eine nichtdeterministische **Turingmaschine (NTM)** ist ein 7-Tupel $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ mit
 - Q nichtleere endliche **Zustandsmenge**
 - Σ endliches **Eingabealphabet**
 - $\Gamma \supseteq \Sigma$ endliches **Bandalphabet**
 - $\delta: Q \times \Gamma \rightarrow \mathcal{P}_e(Q \times \Gamma \times \{L, R\})$ endliche **Überföhrungsfunktion**
 - $q_0 \in Q$ **Startzustand**
 - $B \in \Gamma \setminus \Sigma$ **Leersymbol des Bands**
 - $F \subseteq Q$ Menge von **akzeptierenden (End-)Zuständen**
- **Definition von \vdash^* und $L(M)$ analog zu DTM**
 - $(uZ, q, Xv) \vdash (u, p, ZYv)$, falls $(p, Y, L) \in \delta(q, X)$
 - $(u, q, Xv) \vdash (uY, p, v)$, falls $(p, Y, R) \in \delta(q, X)$
 - \vdots
 - $L(M) = \{w \in \Sigma^* \mid \exists p \in F. \exists u, v \in \Gamma^*. (\epsilon, q_0, w) \vdash^* (u, p, v)\}$

JEDE NTM IST DURCH EINE DTM SIMULIERBAR

Verarbeite alle Alternativen sequentiell

- Für $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ definiere Mengen K_i^w
 - Menge der in i Schritten erzeugbaren Konfigurationen bei Eingabe w

$$K_0^w := \{(\epsilon, q_0, w)\}, \quad K_{i+1}^w := \{\kappa' \mid \exists \kappa \in K_i^w. \kappa \vdash \kappa'\}$$
 - Es gilt $w \in L(M) \Leftrightarrow \exists i. \exists p \in F. \exists u, v \in \Gamma^*. (u, p, v) \in K_i^w$

- Beschreibe DTM zur Erzeugung der K_i^w

K_0^w				K_1^w				K_2^w							
q_0	$w_1..w_n$	$\$$	$\#$	$u_1^1 q_1^1 v_1^1$	$\#$	$u_1^2 q_1^2 v_1^2$	$\# \dots \#$	$u_1^{j_1} q_1^{j_1} v_1^{j_1}$	$\$$	$\#$	$u_2^1 q_1^1 v_2^1$	$\# \dots \#$	$u_2^{j_2} q_2^{j_2} v_2^{j_2}$	$\$$	\dots

- Arbeitsband beschreibt alle bisher erzeugten Konfigurationen der NTM
Die aktuell betrachtete Konfiguration κ wird markiert
 - **Lesen**: Extrahiere aus κ das gelesene Symbol X und Zustand q der NTM
 - **Verarbeiten**: Erzeuge aus κ und $\delta(q, X)$ alle Nachfolgekongfigurationen
Lösche Markierung von κ und markiere nächste Konfiguration auf Band
- Jede mögliche Konfiguration der NTM wird von der DTM aufgesucht

SIMULATION VON NTMS IST EXPONENTIELL

- **Größe der DTM linear mit Größe der NTM**
 - Zustandsüberführungstabelle wird durch Unterprogramme codiert
 - Berechnung der Nachfolgekonfigurationen auf einem Hilfsband
- **Rechenzeit wächst exponentiell**
 - Einzelschritte linear in Größe einer NTM Konfiguration simulierbar
 - Bestimmen einer Nachfolgekonfiguration ist “konstant”
 - Schreiben der Nachfolgekonfiguration linear (zwei Arbeitsbänder)

Aber ...

- **Rechenzeit** der NTM ist Länge des kürzesten akzeptierenden Pfades
- Bei k Alternativen pro Schritt muß die Simulation für n Schritte der NTM im schlimmsten Fall bis zu k^n Konfigurationen erzeugen

Typ-0 Grammatiken und Turingmaschinen beschreiben dieselbe Klasse von Sprachen

- **Grammatik \longrightarrow Turingmaschine**

Turingmaschine simuliert Anwendung der Produktionsregeln

- Ableitbare Wörter werden schrittweise auf Hilfsband geschrieben
- Wörter auf dem Hilfsband werden mit der Eingabe verglichen

Maschine akzeptiert, wenn $w \in L(G)$, und terminiert sonst nicht

- **Turingmaschine \longrightarrow Grammatik**

Grammatik simuliert Konfigurationsübergänge der Turingmaschine

- Erzeuge alle möglichen Eingabewörter und Anfangskonfigurationen
- Codiere Konfigurationsübergänge von M als Regeln
- Simuliere Akzeptieren durch Löschen von Nonterminalsymbolen

Grammatik generiert genau alle Wörter, die M akzeptiert

SATZ: $L \in \mathcal{L}_0 \Rightarrow L$ SEMI-ENTSCHEIDBAR

Zu jeder Grammatik $G = (V, T, P, S)$ kann eine NTM M konstruiert werden mit $L(G) = L(M)$

1. **Schreibe die Eingabe w auf Hilfsband 1**
2. **Schreibe das Startsymbol S auf Hilfsband 2**
3. **Simuliere eine Regelanwendung in G**
 - Wähle nichtdeterministisch ein Teilwort u des Wortes auf Band 2
 - Wähle nichtdeterministisch eine Regel der Form $u \rightarrow v$ aus P
 - Verschiebe Symbole, die rechts von u stehen, um $|v| - |u|$ Stellen
 - Ersetze u durch v
4. **Vergleiche w mit dem Wort auf Hilfsband 2**
 - Akzeptiere w , wenn die Worte gleich sind
 - Ansonsten fahre fort mit 3.

KORREKTHEIT DER KONSTRUKTION

- **M simuliert Ableitbarkeit in G**

- Nach i Schritten steht auf Band 2 ein Wort w_i mit $S \xrightarrow{i}_G w_i$
- Wenn M das Wort w nach i Schritten akzeptiert, dann gilt $w = w_i$

- **M akzeptiert $L(G)$**

- Es gilt $w \in L(G) \Leftrightarrow \exists i. S \xrightarrow{i}_G w$
- Wenn M das Wort w nach i Schritten akzeptiert, dann gilt $S \xrightarrow{i}_G w$, also $w \in L(G)$
- Wenn $S \xrightarrow{i}_G w$ gilt, dann kann M in i Schritten das Wort w auf Band 2 erzeugen und akzeptieren, also $w \in L(M)$

- **M terminiert nicht immer**

- Für $w \notin L(G)$ gilt $w \neq w_i$ für alle i

SATZ: L SEMI-ENTSCHEIDBAR $\Rightarrow L \in \mathcal{L}_0$

Simuliere Abarbeitung der Turingmaschine

- **Idee: Generiere alle Konfigurationen von M**
 - Konfigurationen (u, q, v) werden als Wörter uqv codiert
 - Begrenzer $\#$ trennt Eingabe w von Konfigurationen
 - Verarbeitung von w simuliert durch Wörter der Form $w \# uqv \#$
- **Beschreibe Konfigurationsübergänge durch Regeln**
 - Regeln simulieren Vorschriften für Erzeugung von \vdash aus δ
- **Lege w frei, wenn M akzeptiert hat**
 - Entferne Wort nach $\#$, wenn M einen Endzustand erreicht
- **Grammatik erzeugt von M akzeptierte Sprache**
 - $L(G) = \{w \in \Sigma^* \mid \exists p \in F. \exists u, v \in \Gamma^*. (\epsilon, q_0, w) \vdash^* (u, p, v)\} = L(M)$

REGELN DER GRAMMATIK G

- **Erzeugung von Anfangskonfigurationen**

- Regeln zur Erzeugung aller Wörter der Form $w \# q_0 w \#$

- **Simulation der Konfigurationsübergänge**

- Regeln der Form $q X V \mapsto Y p V$ für $V \in \Gamma$, $\delta(q, X) = (p, Y, R)$

- Regeln der Form $q X \# \mapsto Y p B \#$ für $\delta(q, X) = (p, Y, R)$

- Regeln der Form $Z q X \mapsto p Z Y$ für $Z \in \Gamma$, $\delta(q, X) = (p, Y, L)$

- Regeln der Form $\# q X \mapsto \# p B Y$ für $\delta(q, X) = (p, Y, L)$

- **Schlußregeln für Endzustände**

- Regeln der Form $Z q \mapsto q$ für $Z \in \Gamma$, $q \in F$

- Regeln der Form $q Z \mapsto q$ für $Z \in \Gamma$, $q \in F$

- Regeln der Form $\# q \# \mapsto \epsilon$ für $q \in F$

Detailbeweise z.B. in Erk-Priese, Seite 199–201

Welches Modell paßt zu Typ-1 Sprachen?

- **Typ-1 Sprachen werden “expansiv” erzeugt**
 - In jeder Ableitung $S \longrightarrow w_1 \longrightarrow w_2 \dots \longrightarrow w$ eines Wortes $w \in L(G)$ ist keines der w_i länger als w (Ausnahme $w = \epsilon$)
 - Turingmaschine braucht maximal $|w|$ Bandzellen zur Simulation
- **Beschränke NTMs auf linearen Bandverbrauch**
 - Das Arbeitsband ist nur halbseitig unendlich
 - Anfangskonfigurationen haben die Form $(\epsilon, q_0, w\#)$
 - $\#$ ist ein spezielles Bandende-Symbol, das niemals überlaufen oder überschrieben werden darf
- **Formal: linear beschränkter Automat (LBA)**
 - NTM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ mit halbseitig unendlichem Band und ausgezeichnetem Symbol $\# \in \Gamma \setminus (\Sigma \cup \{B\})$ und der Einschränkung $\delta(q, \#) \subseteq \{(p, \#, L) \mid p \in Q\}$ für alle $q \in Q$

LINEAR BESCHRÄNKTER AUTOMAT FÜR $\{0^n 1^n 2^n \mid n \in \mathbb{N}\}$

- 1. Anfangskonfiguration ist $(\epsilon, q_0, w\#)$**
- 2. Wenn das Band leer ist, akzeptiere die Eingabe**
- 3. Ansonsten ersetze die erste 0 durch B**
 - Wenn keine 0 unter dem Kopf steht, halte an ohne zu akzeptieren
- 4. Gehe rechts zur ersten 1; ersetze diese durch B**
 - Vor der 1 dürfen nur Nullen oder Blanks kommen (!)
 - Wenn keine 1 vorkommt, halte an ohne zu akzeptieren
- 5. Gehe rechts zur ersten 2; ersetze diese durch B**
 - Vor der 2 dürfen nur noch Einsen oder Blanks kommen (!)
 - Wenn keine 2 am Ende steht, halte an ohne zu akzeptieren
- 6. Laufe zurück zum Anfang des restlichen Wortes**
 - Fahre fort mit Schritt 2

Optimierung: Schließe Lücken durch Verschieben

- Verfahren funktioniert analog auch für $\{0^n 1^n 2^n 3^n 4^n \mid n \in \mathbb{N}\}$

LBAS SIND MASCHINENMODELL FÜR TYP-1 SPRACHEN

$$\mathcal{L}_1 = \{ L \mid L = L(M) \text{ für einen LBA } M \}$$

- **Beweise für \mathcal{L}_0 können modifiziert werden**

- **Typ-1 Grammatik \longrightarrow LBA**

Turingmaschine simuliert Anwendung der Produktionsregeln

- Ableitbare Wörter werden schrittweise erzeugt und mit w verglichen
- Beschränkung der Simulation auf Regelanwendungen, die Wörter mit maximaler Länge $|w|$ erzeugen

Maschine ist linear beschränkter Automat

- Linkes und rechtes Ende des Bandes wird niemals überschritten
- Lineare Simulation der Hilfsbänder mit größerem Bandalphabet

- **LBA \longrightarrow Typ-1 Grammatik**

- LBA muß bei Eingabe w das Band nicht mehr erweitern
- Simuliere Verarbeitung der Eingabe w mit Wörtern der Form $(w_1, u_1) \dots (w_i, u_i)(w_{i+1}, q)(w_{i+1}, v_1) \dots (w_n, v_j)$ statt $w \# uqv \#$
- Kürzende **Grammatikregeln können jetzt expansiv formuliert werden**