Informatik IV

Theoretische Informatik

Kapitel 4

Deterministisch Kontextfreie Sprachen

Sommersemester 2019

Dozent: Prof. Dr. J. Rothe

Deterministischer Kellerautomat (DPDA)

Definition

Ein Septupel

$$M = (\Sigma, \Gamma, Z, \delta, z_0, \#, F)$$

heißt *deterministischer Kellerautomat* (kurz DPDA), falls gilt:

1. $M' = (\Sigma, \Gamma, Z, \delta, z_0, \#)$ ist ein PDA;
2. $(\forall a \in \Sigma)\,(\forall A \in \Gamma)\,(\forall z \in Z)\,[\|\delta(z, a, A)\| + \|\delta(z, \lambda, A)\| \leq 1]$;
3. $F \subseteq Z$ ist eine ausgezeichnete Teilmenge von Endzuständen (M *akzeptiert per Endzustand*, nicht per leerem Keller).

Deterministischer Kellerautomat: Beispiel

Beispiel: Der PDA M für die Sprache

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

aus einem früheren Beispiel:

$z_0 a\#$	$\to$	$z_0 A\#$	$z_1 \lambda\#$	$\to$	$z_1 \lambda$
$z_0 aA$	$\to$	$z_0 AA$	$z_1 bA$	$\to$	$z_1 \lambda$
$z_0 bA$	$\to$	$z_1 \lambda$			

hat nur deterministische δ-Übergänge, ist aber aufgrund der Akzeptanz mittels leerem Keller nach obiger Definition kein DPDA.

Deterministischer Kellerautomat: Beispiel

Um einen deterministischen Kellerautomaten M_e für die Sprache $L = \{a^m b^m \mid m \geq 1\}$ zu definieren, müssen wir noch einen Endzustand z_e einführen: $M_e = (\{a, b\}, \{A, \#\}, \{z_0, z_1, z_e\}, \delta_e, z_0, \#, \{z_e\})$ mit der Überführungsfunktion δ_e:

$z_0 a \#$	$\to$	$z_0 A \#$	$z_1 \lambda \#$	$\to$	$z_e \lambda$
$z_0 a A$	$\to$	$z_0 A A$	$z_1 b A$	$\to$	$z_1 \lambda$
$z_0 b A$	$\to$	$z_1 \lambda$			

Beispielsweise ist $aabb \in L(M_e) = L$, denn:

$$
\begin{aligned}
(z_0, aabb, \#) &\vdash_{M_e} (z_0, abb, A\#) \vdash_{M_e} (z_0, bb, AA\#) \\
&\vdash_{M_e} (z_1, b, A\#) \quad \vdash_{M_e} (z_1, \lambda, \#) \\
&\vdash_{M_e} (z_e, \lambda, \lambda)
\end{aligned}
$$

Deterministischer Kellerautomat: Beispiel

Beispiel: Der PDA $M' = (\{a, b\}, \{A, B, \#\}, \{z_0, z_1\}, \delta, z_0, \#)$ mit der Überführungsfunktion δ:

$z_0a\#$	$\to$	$z_0A\#$	z_0aA	$\to$	$z_1\lambda$	z_0aA	$\to$	z_0AA	z_0bB	$\to$	$z_1\lambda$
z_0aB	$\to$	z_0AB	$z_0\lambda\#$	$\to$	$z_1\lambda$	$z_0b\#$	$\to$	$z_0B\#$	z_1aA	$\to$	$z_1\lambda$
z_0bA	$\to$	z_0BA	z_1bB	$\to$	$z_1\lambda$	z_0bB	$\to$	z_0BB	$z_1\lambda\#$	$\to$	$z_1\lambda$

mit $L(M') = \{w\ sp(w) \mid w \in \{a, b\}^*\}$ ist kein DPDA, da M' nichtdeterministische δ-Übergänge hat:

- $\delta(z_0, a, A) = \{(z_0, AA), (z_1, \lambda)\}$, also $\|\delta(z_0, a, A)\| = 2 > 1$,
- $\delta(z_0, b, B) = \{(z_0, BB), (z_1, \lambda)\}$, also $\|\delta(z_0, b, B)\| = 2 > 1$, und
- $\delta(z_0, a, \#) = \{(z_0, A\#)\}$ und $\delta(z_0, \lambda, \#) = \{(z_1, \lambda)\}$, also $\|\delta(z_0, a, \#)\| + \|\delta(z_0, \lambda, \#)\| = 2 > 1$.

Sprache eines deterministischen Kellerautomaten

Definition

Die von einem deterministischen Kellerautomaten

$$M = (\Sigma, \Gamma, Z, \delta, z_0, \#, F)$$

akzeptierte *Sprache* ist definiert durch

$$L(M) = \{x \in \Sigma^* \mid (z_0, x, \#) \vdash_M^* (z, \lambda, \gamma) \text{ für ein } z \in F \text{ und } \gamma \in \Gamma^*\}.$$

- Für jedes Eingabewort $w \in \Sigma^*$ ist $(z_0, w, \#)$ die entsprechende *Startkonfiguration* von M.
- Für jedes $\gamma \in \Gamma^*$ und $z \in F$ ist die Konfiguration (z, λ, γ) eine *Endkonfiguration* des DPDA M.

Deterministisch kontextfreie Sprache

Definition

- Eine Sprache A heißt *deterministisch kontextfrei*, falls es einen deterministischen Kellerautomaten M gibt mit

$$A = L(M).$$

- Mit

$$\mathrm{DCF} = \{L(M) \mid M \text{ ist DPDA}\}$$

bezeichnen wir die *Klasse der deterministisch kontextfreien Sprachen*.

Deterministisch kontextfreie Sprachen

Bemerkung:

- Im Gegensatz zu (nichtdeterministischen) Kellerautomaten macht es bei DPDA einen Unterschied, ob man sie mittels Akzeptierung per Endzustand oder mittels Akzeptierung per leerem Keller definiert.
- DCF stimmt genau mit der Klasse der so genannten LR(k)-Sprachen überein, die bei der Syntaxanalyse im Compilerbau eine Rolle spielen.
- Für deterministisch kontextfreie Sprachen ist das Wortproblem in linearer Zeit lösbar.

Deterministisch kontextfreie Sprachen

Theorem

$\mathrm{REG} \subset \mathrm{DCF} \subset \mathrm{CF}$.

Beweis: Die Inklusionen $\mathrm{REG} \subseteq \mathrm{DCF} \subseteq \mathrm{CF}$ sind klar.
Die Ungleichheiten werden nicht bewiesen, sondern nur skizziert:

- Die Sprache $L = \{w \,\$\, sp(w) \mid w \in \{a,b\}^*\}$ ist deterministisch kontextfrei. Andererseits kann man mit dem Pumping-Lemma für reguläre Sprachen zeigen, dass $L \notin \mathrm{REG}$.
- Die Sprache $L' = \{w\, sp(w) \mid w \in \{a,b\}^*\}$ ist jedoch *nicht* deterministisch kontextfrei. Andererseits ist L' in CF.

Daraus folgt $\mathrm{REG} \neq \mathrm{DCF} \neq \mathrm{CF}$, also $\mathrm{REG} \subset \mathrm{DCF} \subset \mathrm{CF}$. ❑

Deterministisch kontextfreie Sprachen

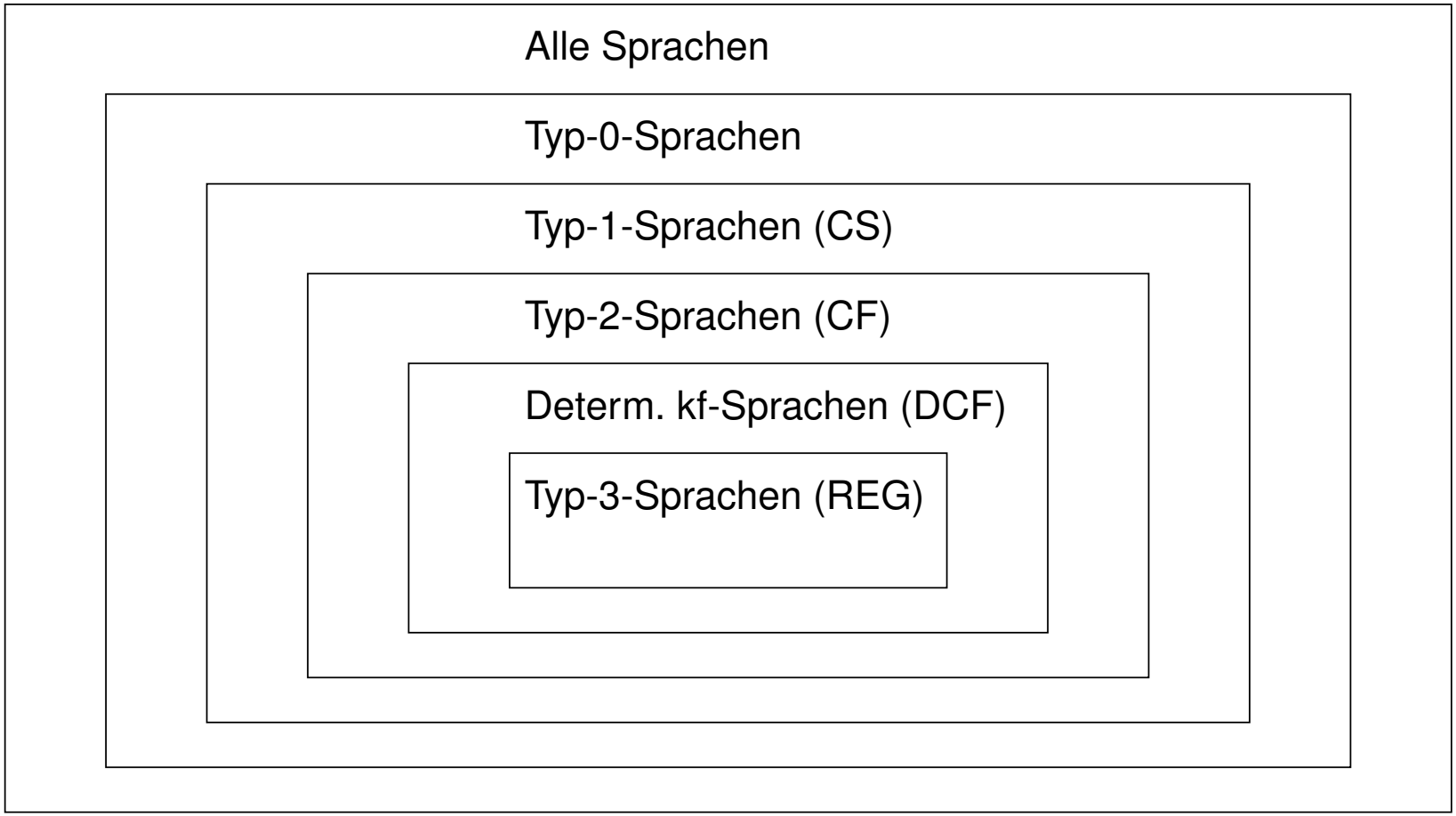

Abbildung: Einordnung von DCF in die Chomsky-Hierarchie

Deterministisch kontextfreie Sprachen

Theorem

DCF *ist abgeschlossen*

- *unter Komplement,*

aber weder

- *unter Schnitt* *(wie CF)*
- *noch unter Vereinigung*
- *noch unter Konkatenation*
- *noch unter Iteration.*

ohne Beweis

Deterministisch kontextfreie Sprachen

Behauptung: Die Sprache

$$\begin{aligned} L &= \{a^n b^n c^m \mid n, m \geq 0\} \cup \{a^n b^m c^m \mid n, m \geq 0\} \\ &= \{a^i b^j c^k \mid i = j \text{ oder } j = k\} \end{aligned}$$

ist kontextfrei und inhärent mehrdeutig. **ohne Beweis**

Ziel der Syntaxanalyse

- Programmiersprachen sind meist kontextfreie Sprachen, deren Wörter die syntaktisch korrekten Programme sind.
- Für eine durch eine kfG G gegebene Programmiersprache $L = L(G)$ und ein gegebenes Programm w soll getestet werden, ob w syntaktisch korrekt ist, d.h., ob gilt: $w \in L(G)$.
- Das ist gerade das Wortproblem.
  - Falls ja: Angabe eines Syntaxbaums der Ableitung $S \vdash_G^* w$.
  - Falls nein: Angabe möglichst vieler Syntaxfehler.

  Wir beschränken uns auf den Fall „ja“.

Aufgaben eines Compilers (Übersetzers)

Übersetzen eines Programms aus einer höheren Programmiersprache in Maschinensprache.

1. Die *lexikalische Analyse* erkennt mittels endlicher Automaten und bestimmter Hashing-Techniken Schlüsselwörter (`while`, `for`, `if`, `then`, etc.) und vereinbarte Namen.
2. Die *Syntaxanalyse* überprüft die syntaktische Korrektheit des Programmtextes.
3. Die *semantische Analyse* überprüft die „semantische“ Korrektheit des Programms.
4. Die *Code-Erzeugung* (Assembler Code, Maschinensprache) und *Optimierung* ermöglichen die Synthese des erfolgreich kompilierten Programms.

Strategien der Syntaxanalyse (Parsing)

- allgemeine Verfahren für bestimmte Teilklassen;
- Tabellenstrategien (z.B. der Algorithmus von Cocke, Younger und Kasami);
- ableitungsorientierte Strategien.

Unter den vielen Strategien für die Syntaxanalyse interessieren wir uns nur für ein spezielles Top-down-Verfahren, das v.l.n.r. vorgeht:

das $\mathrm{LL}(k)$-Parsing.

Grundidee bei ableitungsorientierten Strategien

- Sei $G = (\Sigma, N, S, P)$ eine kfG, $w \in L(G)$ sei ein gegebenes Wort (d.h. Programm), und T sei ein Syntaxbaum für die Ableitung

$$S \vdash_G^* w = xyz.$$

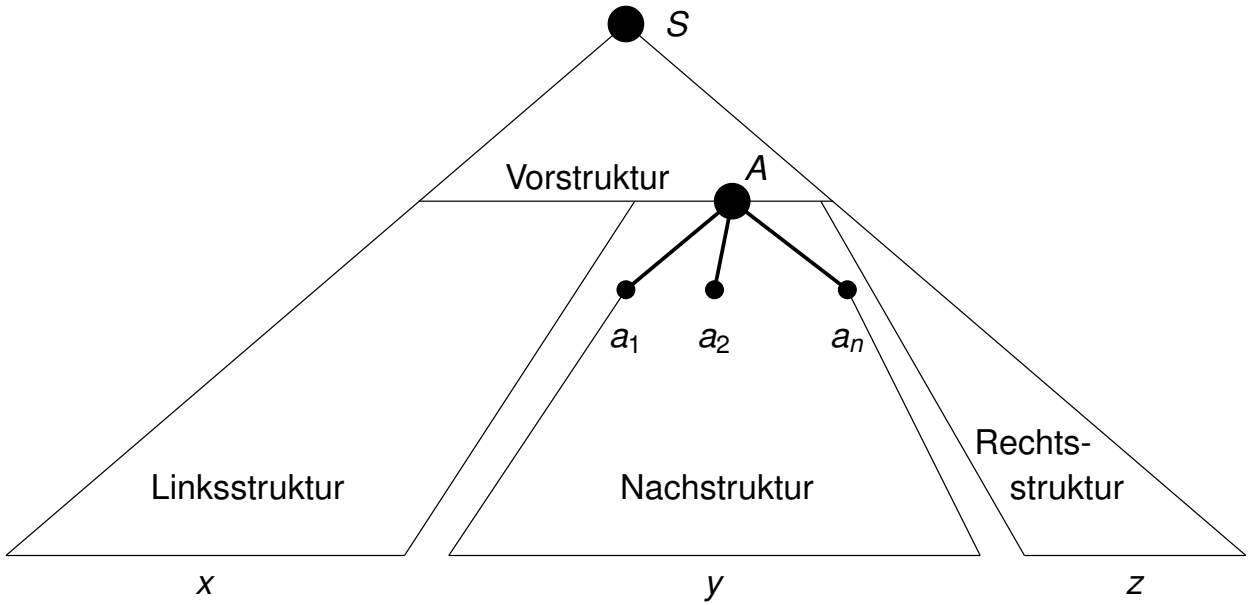

Abbildung: Struktur eines Syntaxbaums beim Parsing

Grundidee bei ableitungsorientierten Strategien

- Betrachte eine Verzweigung in T, die durch Anwendung der Regel

$$A \to \alpha = a_1 a_2 \cdots a_n$$

entsteht, wobei $A \in N$ und jedes $a_i \in \Sigma \cup N$.

- Die Abbildung zeigt die Struktur des Syntaxbaums T beim Parsing.
- Dabei ist $w = xyz$, wobei
  - x der terminale Linkskontext,
  - y der terminale Nachkontext und
  - z der terminale Rechtskontext

von A sind.

Grundidee bei ableitungsorientierten Strategien

- Die wesentlichen Strategien sind:

  Top-down-Analyse:

  Fortschreiten, wenn die Vorstruktur bekannt und die Nachstruktur unbekannt ist, d.h. von oben nach unten.
  Dabei geht man von links nach rechts vor, erweitert also einen schon bekannten Präfix des abzuleitenden Wortes.

  Bottom-up-Analyse:

  Fortschreiten, wenn die Nachstruktur bekannt und die Vorstruktur unbekannt ist, d.h. von unten nach oben.
  Dabei geht man von rechts nach links vor, erweitert also einen schon bekannten Suffix des abzuleitenden Wortes.

Grundidee bei ableitungsorientierten Strategien

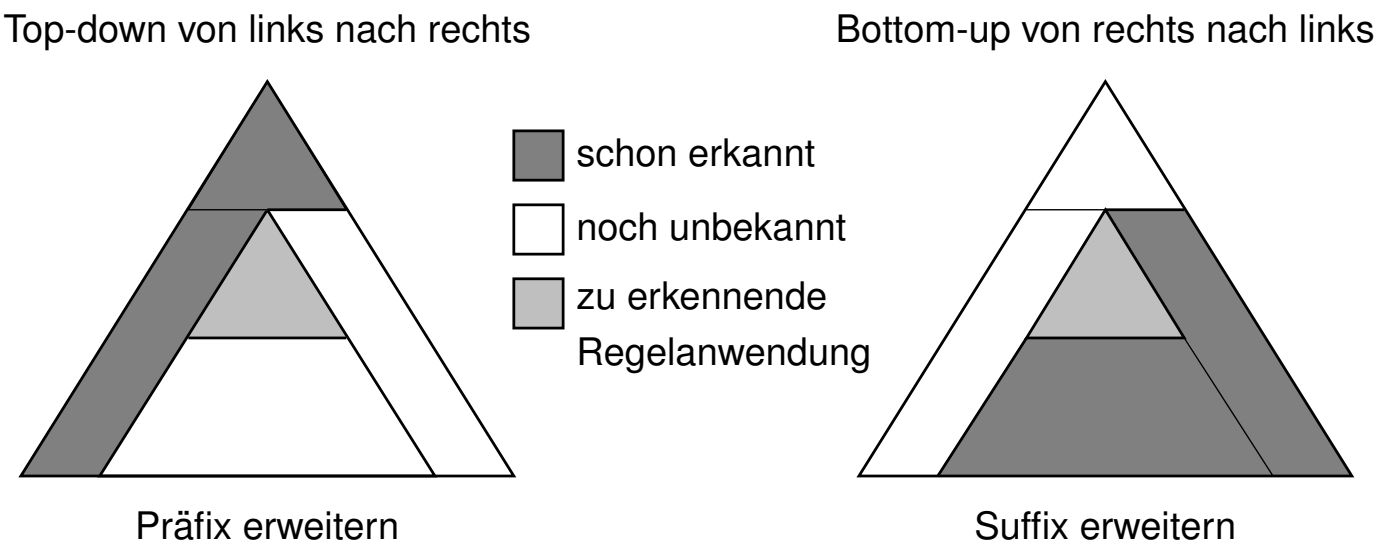

Abbildung: Top-down-Strategie und Bottom-up-Strategie

Die Abbildung veranschaulicht beide Strategien.

Look-ahead: Beispiel

Beispiel:

- Betrachte das Wort $w = abba\$$ und die kfG G mit den folgenden Regeln:

$$S \to C\$, \quad C \to bA \mid aB, \quad A \to a \mid aC, \quad B \to b \mid bC,$$

wobei das Terminalzeichen \$ als eine Endmarke dient.

- Die einzige mögliche Startmöglichkeit für eine Ableitung $S \vdash^*_{G,l} w$ ist die Regel

$$S \to C\$.$$

Look-ahead: Beispiel

Beispiel:

- Danach gibt es zwei Möglichkeiten, C zu ersetzen:

$$C \to bA \text{ und } C \to aB.$$

- Die erste Möglichkeit entfällt aber, da der erste Buchstabe von w ein a ist; also muss $C \to aB$ angewandt werden.

- Es reicht also ein Look-ahead von einem Symbol aus, um die richtige Entscheidung zu treffen.
- Bisher gilt:

$$S \vdash_{G,l} C\$ \vdash_{G,l} aB\$.$$

Look-ahead: Beispiel

Beispiel:

- Nun gibt es zwei Möglichkeiten, B zu ersetzen:

$$B \to b \text{ und } B \to bC.$$

- Aber diesmal reicht ein Look-ahead von einem Symbol nicht mehr aus, da beide Regeln mit b beginnen.

- Wir brauchen also einen Look-ahead von zwei Symbolen.

- Damit sehen wir, dass wieder die erste Möglichkeit entfällt, da $w \neq ab\$$. Es wird also $B \to bC$ angewandt, und wir erhalten:

$$S \vdash_{G,l} C\$ \vdash_{G,l} aB\$ \vdash_{G,l} abC\$.$$

Look-ahead: Beispiel

Beispiel:

- Es gibt zwei Möglichkeiten, C zu ersetzen.
- Da der dritte Buchstabe von w ein b ist (ein Look-ahead von einem Symbol genügt), wird $C \to bA$ angewandt:

$$S \vdash_{G,l} C\$ \vdash_{G,l} aB\$ \vdash_{G,l} abC\$ \vdash_{G,l} abbA\$.$$

- Nun ist nur die Regel $A \to a$ möglich:

$$S \vdash_{G,l} C\$ \vdash_{G,l} aB\$ \vdash_{G,l} abC\$ \vdash_{G,l} abba\$,$$

  und der Syntaxbaum für die Ableitung $S \vdash^*_{G,l} w$ ist gefunden.
- Insgesamt genügte immer ein Look-ahead von zwei Symbolen. G ist ein Beispiel für eine (starke) $\mathrm{LL}(2)$-Grammatik.

Deutschlands längstes Wort hat ausgedient

DPA-Meldung am 03.06.2013, 08:12 Uhr:

- *Das längste Wort der deutschen Sprache verschwindet aus dem Sprachgebrauch.*
  Der mecklenburg-vorpommerische Landtag in Schwerin beschloss vergangene Woche, das 63 Buchstaben lange Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz aus dem Landesrecht zu verbannen.

Deutschlands längstes Wort hat ausgedient

DPA-Meldung am 03.06.2013, 08:12 Uhr:

- *Das längste Wort der deutschen Sprache verschwindet aus dem Sprachgebrauch.*
  Der mecklenburg-vorpommerische Landtag in Schwerin beschloss vergangene Woche, das 63 Buchstaben lange
  Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz
  aus dem Landesrecht zu verbannen.

- *In den Duden schaffte es das Gesetz nie, weil es nicht häufig genug im Sprachgebrauch auftauchte, sagte eine Duden-Sprecherin. Spitzenreiter im Wörterbuch ist derzeit die 36 Buchstaben lange*
  Kraftfahrzeug-Haftpflichtversicherung.