Theoretische Informatik
Kapitel 4 – Deterministisch kontextfreie Sprachen

Sommersemester 2024

Dozentin: Mareike Mutz

im Wechsel mit

Prof. Dr. M. Leuschel

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:

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

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

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:

$z_0 a\# \rightarrow z_0 A\#$	$z_1 \lambda\# \rightarrow z_e \lambda$
$z_0 aA \rightarrow z_0 AA$	$z_1 bA \rightarrow z_1 \lambda$
$z_0 bA \rightarrow z_1 \lambda$	

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

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

Deterministischer Kellerautomat: Beispiel

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

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

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:

- Bei (nichtdeterministischen) PDAs macht es für die Ausdrucksstärke keinen Unterschied, ob man Akzeptierung per Endzustand oder per leerem Keller definiert:
  - leer $\rightarrow$ Endzustand:
    zusätzliches neues Symbol unten auf Stapel ablegen und in Endzustand wechseln wenn dieses Symbol oben auftaucht
  - Endzustand $\rightarrow$ leer:
    bei Endzustand in einen neuen Zustand wechseln der den Stapel leert; zusätzliches Symbol auf dem Stapel verhindert falsches Erkennen außerhalb von Endzuständen
- Im Gegensatz dazu, macht es bei DPDA einen Unterschied, ob man sie mittels Akzeptierung per Endzustand oder mittels Akzeptierung per leerem Keller definiert.

Deterministisch kontextfreie Sprachen

Bemerkung:

- DPDA mittels Akzeptierung per leerem Keller definiert erkennen Sprachen L mit der Präfix Eigenschaft: $\alpha \in L \wedge \alpha\beta \in L \Rightarrow \beta = \lambda$. Man benutzt deshalb Akzeptierung per Endzustand für DPDAs
- 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

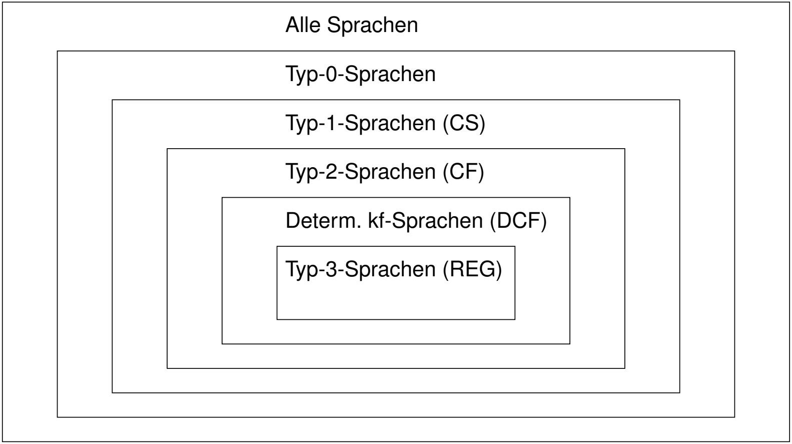

Abbildung: Einordnung von DCF in die Chomsky-Hierarchie

Deterministisch kontextfreie Sprachen

Theorem

DCF *ist abgeschlossen*

- *unter Komplement,* *(im Gegensatz zu CF)*

aber weder

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

ohne Beweis

Deterministisch kontextfreie Sprachen

Gegenbeispiel für Abschluss unter Vereinigung:

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".

Java Syntax

14.9. The `if` Statement

The `if` statement allows conditional execution of a statement or a conditional choice of two statements, executing one or the other but not both.

```
IfThenStatement:
    if ( Expression ) Statement

IfThenElseStatement:
    if ( Expression ) StatementNoShortIf else Statement

IfThenElseStatementNoShortIf:
    if ( Expression ) StatementNoShortIf else StatementNoShortIf
```

The *Expression* must have type `boolean` or `Boolean`, or a compile-time error occurs.

14.9.1. The `if-then` Statement

An `if-then` statement is executed by first evaluating the *Expression*. If the result is of type `Boolean`, it is subject to unboxing conversion (§5.1.8).

If evaluation of the *Expression* or the subsequent unboxing conversion (if any) completes abruptly for some reason, the `if-then` statement completes abruptly for the same reason.

Otherwise, execution continues by making a choice based on the resulting value:

- If the value is `true`, then the contained *Statement* is executed; the `if-then` statement completes normally if and only if execution of the *Statement* completes normally.
- If the value is `false`, no further action is taken and the `if-then` statement completes normally.

14.9.2. The `if-then-else` Statement

An `if-then-else` statement is executed by first evaluating the *Expression*. If the result is of type `Boolean`, it is subject to unboxing conversion (§5.1.8).

If evaluation of the *Expression* or the subsequent unboxing conversion (if any) completes abruptly for some reason, then the `if-then-else` statement completes abruptly for the same reason.

Otherwise, execution continues by making a choice based on the resulting value:

- If the value is `true`, then the first contained *Statement* (the one before the `else` keyword) is executed; the `if-then-else` statement completes normally if and only if execution of that statement completes normally.
- If the value is `false`, then the second contained *Statement* (the one after the `else` keyword) is executed; the `if-then-else` statement completes normally if and only if execution of that statement completes normally.

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.

Compilerphasen

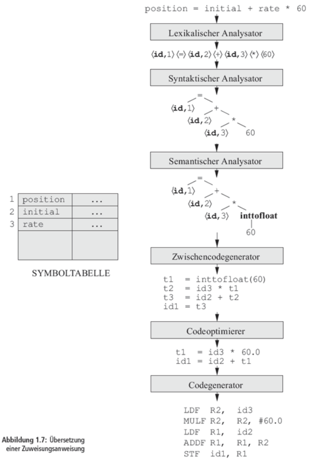

Abbildung 1.7: Übersetzung einer Zuweisungsanweisung

Quelle: Aho, Lam, Sethi, Ullman: Compiler, 2008, Pearson.

Strategien der Syntaxanalyse (Parsing)

- allgemeine Verfahren für bestimmte Teilklassen;
- Tabellenstrategien (z.B. der Algorithmus von Cocke, Younger und Kasami);
- ableitungsorientierte Strategien
  - Top-Down Strategien (zB $\mathrm{LL}(k)$)-Parsing)
  - Bottom-Up Strategien (zB LR(k) oder LALR(k) Parsing

Wiederholung: Parsing: Einfaches Beispiel

$G = (\{a, b\},\ \{S, A\},\ S,\ R)$ mit

$$R = \{S \to AS \mid b,$$
$$A \to a\}$$

Ist $aab \in L(G)$?

Top-down parsing: wir starten mit S und versuchen eine Ableitung von aab zu finden:

$$S \vdash AS \vdash aS \vdash aAS \vdash aaS \vdash aab$$

LL(K) Parsing: basierend auf den nächsten k Symbolen wird entschieden welche Regel verwendet wird.

Bottom-Up parsing: wir starten mit aab und versuchen rückwärts zu S zu kommen:

Top-Down Parsing

Grammatiken müssen oft umgeschrieben werden damit diese für deterministisches Top-Down Parsing geeignet sind.

$$E \to E + E \mid E * E \mid (E) \mid id$$

Nach Links-Faktorisierung und Elimination von Mehrdeutigkeit und Links-Rekursion (siehe Compilerbau):

$$E \to TE'$$

$$E' \to +TE' \mid \lambda$$

$$T \to FT'$$

$$T' \to *FT' \mid \lambda$$

$$F \to (E) \mid id$$

Top-Down Parsing

$$E \to TE'$$

$$E' \to +TE' \mid \lambda$$

$$T \to FT'$$

$$T' \to *FT' \mid \lambda$$

$$F \to (E) \mid id$$

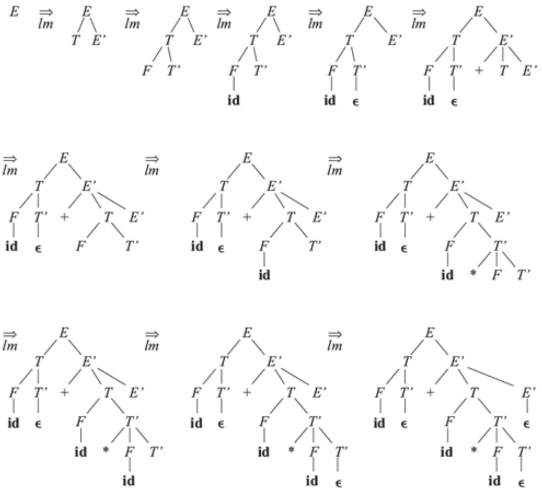

Abbildung 4.12: Top-Down-Parsing für id+id * id

Top-Down Parsing

$$E \to TE' \ , \ E' \to +TE' \mid \lambda$$
$$T \to FT' \ , \ T' \to *FT' \mid \lambda \ , \ F \to (E) \mid id$$

Aus dieser Grammatik kann eine Parsingtabelle für einen LL(1) Parser erstellt werden, der basierend auf dem nächsten Symbol (Lookahead von 1) entscheidet welche Regel angewendet wird:

Nichtterminal	Eingabesymbol					
	id	+	*	(	)	\$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to \text{id}$			$F \to (E)$		

Abbildung 4.17: Parsertabelle M zu Beispiel 4.18

Quelle: Aho, Lam, Sethi, Ullman: Compiler, 2008, Pearson.

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:

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

- Die erste Möglichkeit entfällt aber, da der erste Buchstabe von w ein a ist; also muss $C \rightarrow aB$ 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:

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

- 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 $B \rightarrow bC$ 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 $C \rightarrow bA$ angewandt:

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

- Nun ist nur die Regel $A \rightarrow a$ 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.