Script generated by TTT

Title: Simon: Compilerbau (02.05.2012)

Date: Wed May 02 14:11:53 CEST 2012

Duration: 85:12 min

Pages: 63

Bemerkungen:

- Bei einem Eingabewort der Länge n werden maximal $\mathcal{O}(n)$ Mengen konstruiert
- Ist eine Menge bzw. eine Kante des DFA einmal konstruiert, heben wir sie in einer Hash-Tabelle auf.
- Bevor wir einen neuen Übergang konstruieren, sehen wir erst nach, ob wir diesen nicht schon haben

Zusammenfassend finden wir:

Satz:

Zu jedem regulären Ausdruck e kann ein deterministischer Automat $A = \mathcal{P}(A_e)$ konstruiert werden mit

$$\mathcal{L}(A) = [\![e]\!]$$

Kapitel 5:
Design eines Scanners

Design eines Scanners

Eingabe (vereinfacht): eine Menge von Regeln:

$$\begin{array}{ll} e_1 & \{\, \texttt{action}_1 \,\} \\ e_2 & \{\, \texttt{action}_2 \,\} \\ \multicolumn{2}{c}{\ldots} \\ e_k & \{\, \texttt{action}_k \,\} \end{array}$$

Design eines Scanners

Eingabe (vereinfacht): eine Menge von Regeln:

$$\begin{array}{ll} e_1 & \{\ \texttt{action}_1\ \} \\ e_2 & \{\ \texttt{action}_2\ \} \\ \quad\ldots & \\ e_k & \{\ \texttt{action}_k\ \} \end{array}$$

Ausgabe: ein Programm, das

- ... von der Eingabe ein maximales Präfix w liest, das $e_1 \mid \ldots \mid e_k$ erfüllt;
- ... das minimale i ermittelt, so dass $w \in [\![e_i]\!]$;
- ... für w $\texttt{action}_i$ ausführt.

Implementierung:

Idee:

- Konstruiere den DFA $\mathcal{P}(A_e) = (Q, \Sigma, \delta, \{q_0\}, F)$ zu dem Ausdruck $e = (e_1 \mid \ldots \mid e_k)$;
- Definiere die Mengen:

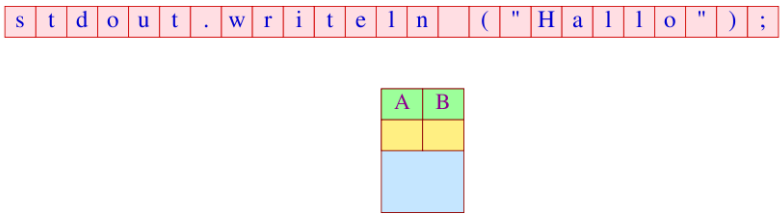
$$\begin{array}{lcl} F_1 & = & \{q \in F \mid q \cap \mathsf{last}[e_1] \neq \emptyset\} \\ F_2 & = & \{q \in (F \backslash F_1) \mid q \cap \mathsf{last}[e_2] \neq \emptyset\} \\ & \ldots & \\ F_k & = & \{q \in (F \backslash (F_1 \cup \ldots \cup F_{k-1})) \mid q \cap \mathsf{last}[e_k] \neq \emptyset\} \end{array}$$

- Für Eingabe w gilt: $\delta^*(q_0, w) \in F_i$ genau dann wenn der Scanner für w $\texttt{action}_i$ ausführen soll

Implementierung:

Idee (Fortsetzung):

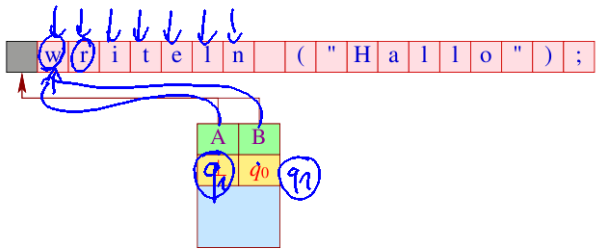
- Der Scanner verwaltet zwei Zeiger $\langle A, B\rangle$ und die zugehörigen Zustände $\langle q_A, q_B\rangle$...
- Der Zeiger A merkt sich die letzte Position in der Eingabe, nach der ein Zustand $q_A \in F$ erreicht wurde;
- Der Zeiger B verfolgt die aktuelle Position.



Implementierung:

Idee (Fortsetzung):

- Der Scanner verwaltet zwei Zeiger $\langle A, B\rangle$ und die zugehörigen Zustände $\langle q_A, q_B\rangle$...
- Der Zeiger A merkt sich die letzte Position in der Eingabe, nach der ein Zustand $q_A \in F$ erreicht wurde;
- Der Zeiger B verfolgt die aktuelle Position.

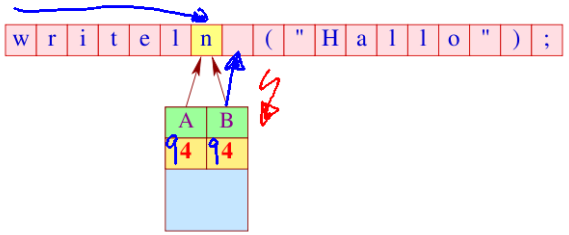


Implementierung:

Idee (Fortsetzung):

- Ist der aktuelle Zustand $q_B = \emptyset$, geben wir Eingabe bis zur Postion A aus und setzen:

$$\begin{array}{llllll} B & := & A; & A & := & \perp; \\ q_B & := & q_0; & q_A & := & \perp \end{array}$$

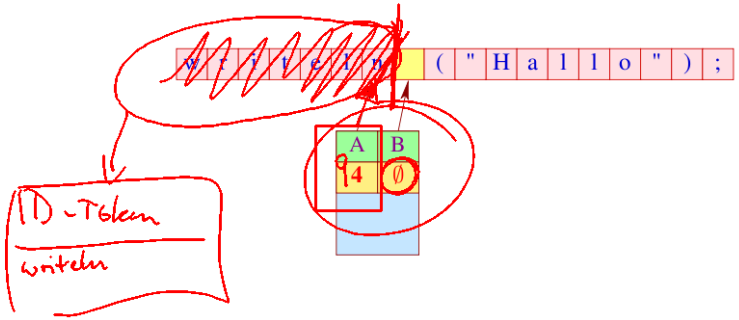


Implementierung:

Idee (Fortsetzung):

- Ist der aktuelle Zustand $q_B = \emptyset$, geben wir Eingabe bis zur Postion A aus und setzen:

$$\begin{array}{llllll} B & := & A; & A & := & \perp; \\ q_B & := & q_0; & q_A & := & \perp \end{array}$$

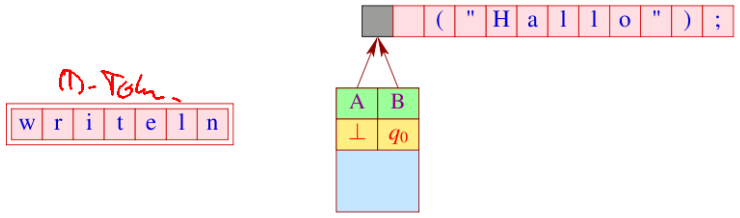


Implementierung:

Idee (Fortsetzung):

- Ist der aktuelle Zustand $q_B = \emptyset$, geben wir Eingabe bis zur Postion A aus und setzen:

$$\begin{array}{llllll} B & := & A; & A & := & \perp; \\ q_B & := & q_0; & q_A & := & \perp \end{array}$$



Erweiterung: Zustände

- Gelegentlich ist es nützlich, verschiedene Scanner-Zustände zu unterscheiden.
- In unterschiedlichen Zuständen sollen verschiedene Tokenklassen erkannt werden können.
- In Abhängigkeit der gelesenen Tokens kann der Scanner-Zustand geändert werden

Beispiel: Kommentare

Innerhalb eines Kommentars werden Identifier, Konstanten, Kommentare, ... nicht erkannt

Eingabe (verallgemeinert): eine Menge von Regeln:

$\langle$state$\rangle$ { e_1 { action$_1$ yybegin(state$_1$); }
e_2 { action$_2$ yybegin(state$_2$); }
...
e_k { action$_k$ yybegin(state$_k$); }
}

- Der Aufruf `yybegin (state`$_i$`);` setzt den Zustand auf state$_i$.
- Der Startzustand ist (z.B. bei JFlex) `YYINITIAL`.

... im Beispiel:

```
⟨YYINITIAL⟩    "/*"    { yybegin(COMMENT); }
⟨COMMENT⟩    { "*/"    { yybegin(YYINITIAL); }
               . | \n    {  }
             }
```

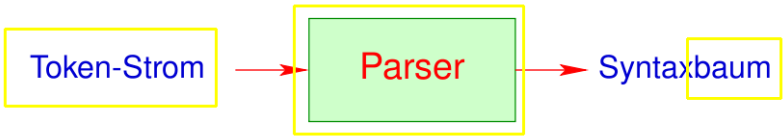

Bemerkungen:

- "`.`" matcht alle Zeichen ungleich "`\n`".
- Für jeden Zustand generieren wir den entsprechenden Scanner.
- Die Methode `yybegin (STATE);` schaltet zwischen den verschiedenen Scannern um.
- Kommentare könnte man auch direkt mithilfe einer geeigneten Token-Klasse implementieren. Deren Beschreibung ist aber ungleich komplizierter
- Scanner-Zustände sind insbesondere nützlich bei der Implementierung von Präprozessoren, die in einen Text eingestreute Spezifikationen expandieren sollen.

Themengebiet:

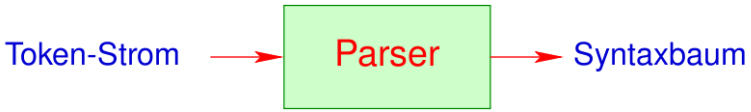
Syntaktische Analyse

Die syntaktische Analyse



- Die syntaktische Analyse versucht, Tokens zu größeren Programmeinheiten zusammen zu fassen.

Die syntaktische Analyse



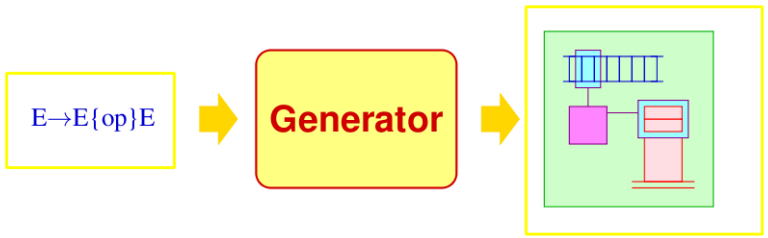
- Die syntaktische Analyse versucht, Tokens zu größeren Programmeinheiten zusammen zu fassen.
- Solche Einheiten können sein:
  - → Ausdrücke;
  - → Statements;
  - → bedingte Verzweigungen;
  - → Schleifen; ...

Diskussion:

Auch Parser werden i.a. nicht von Hand programmiert, sondern aus einer Spezifikation generiert:

Diskussion:

Auch Parser werden i.a. nicht von Hand programmiert, sondern aus einer Spezifikation generiert:



Spezifikation der hierarchischen Struktur: kontextfreie Grammatiken

Generierte Implementierung: Kellerautomaten + X

Ende der Präsentation. Klicken Sie zum Schließen.

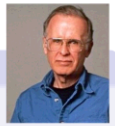
Grundlagen: Kontextfreie Grammatiken

- Programme einer Programmiersprache können unbeschränkt viele Tokens enthalten, aber nur endlich viele Token-Klassen
- Als endliches Terminal-Alphabet T wählen wir darum die Menge der Token-Klassen.
- Die Schachtelung von Programm-Konstrukten lässt sich elegant mit Hilfe von kontextfreien Grammatiken beschreiben ...

Grundlagen: Kontextfreie Grammatiken

- Programme einer Programmiersprache können unbeschränkt viele Tokens enthalten, aber nur endlich viele Token-Klassen
- Als endliches Terminal-Alphabet T wählen wir darum die Menge der Token-Klassen.
- Die Schachtelung von Programm-Konstrukten lässt sich elegant mit Hilfe von kontextfreien Grammatiken beschreiben ...


Noam Chomsky


John Backus

Definition:

Eine kontextfreie Grammatik (CFG) ist ein 4-Tupel $G = (N, T, P, S)$ mit:

- N die Menge der Nichtterminale,
- T die Menge der Terminale,
- P die Menge der Produktionen oder Regeln, und
- $S \in N$ das Startsymbol

Konventionen

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \to \alpha \quad \text{mit} \quad A \in N\,,\ \alpha \in (N \cup T)^*$$

Konventionen

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \to \alpha \quad \text{mit} \quad A \in N\,,\ \alpha \in (N \cup T)^*$$

... im Beispiel:

$$\begin{aligned} S &\to a\,S\,b \\ S &\to \epsilon \end{aligned}$$

Spezifizierte Sprache: $\{a^n b^n \mid n \geq 0\}$

Konventionen

Die Regeln kontextfreier Grammatiken sind von der Form:

$$A \to \alpha \quad \text{mit} \quad A \in N\,, \quad \alpha \in (N \cup T)^*$$

... im Beispiel:

$$\begin{array}{lcl} S & \to & a\,S\,b \\ S & \to & \epsilon \end{array}$$

Spezifizierte Sprache: $\{a^n b^n \mid n \geq 0\}$

Konventionen:

In Beispielen ist die Spezifikation der Nichtterminale und Terminale i.a. implizit:

- Nichtterminale sind: $A, B, C, ..., \langle \text{exp} \rangle, \langle \text{stmt} \rangle, ...;$
- Terminale sind: $a, b, c, ..., \text{int}, \text{name}, ...;$

... weitere Beispiele:

$$\begin{array}{lcl} S & \to & \langle \text{stmt} \rangle \\ \langle \text{stmt} \rangle & \to & \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle; \\ \langle \text{if} \rangle & \to & \text{if (} \langle \text{rexp} \rangle \text{) } \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle \\ \langle \text{while} \rangle & \to & \text{while (} \langle \text{rexp} \rangle \text{) } \langle \text{stmt} \rangle \\ \langle \text{rexp} \rangle & \to & \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid ... \\ \langle \text{lexp} \rangle & \to & \text{name} \mid ... \end{array}$$

... weitere Beispiele:

$$\begin{array}{lcl} S & \to & \langle \text{stmt} \rangle \\ \langle \text{stmt} \rangle & \to & \langle \text{if} \rangle \mid \langle \text{while} \rangle \mid \langle \text{rexp} \rangle; \\ \langle \text{if} \rangle & \to & \text{if (} \langle \text{rexp} \rangle \text{) } \langle \text{stmt} \rangle \text{ else } \langle \text{stmt} \rangle \\ \langle \text{while} \rangle & \to & \text{while (} \langle \text{rexp} \rangle \text{) } \langle \text{stmt} \rangle \\ \langle \text{rexp} \rangle & \to & \text{int} \mid \langle \text{lexp} \rangle \mid \langle \text{lexp} \rangle = \langle \text{rexp} \rangle \mid ... \\ \langle \text{lexp} \rangle & \to & \text{name} \mid ... \end{array}$$

Weitere Konventionen:

- Für jedes Nichtterminal sammeln wir die rechten Regelseiten und listen sie gemeinsam auf
- Die j-te Regel für A können wir durch das Paar (A, j) bezeichnen ($j \geq 0$).

Weitere Grammatiken:

$$\begin{array}{lcl} E & \to & E{+}E \mid E{*}E \mid (\ E\ ) \mid \text{name} \mid \text{int} \end{array}$$

$$\begin{array}{lcl} E & \to & E{+}T \mid T \\ T & \to & T{*}F \mid F \\ F & \to & (\ E\ ) \mid \text{name} \mid \text{int} \end{array}$$

Die beiden Grammatiken beschreiben die gleiche Sprache

Weitere Grammatiken:

E	$\rightarrow$	$E+E$ [0]	$\mid$	$E*E$ [1]	$\mid$	$(\ E\ )$ [2]	$\mid$ name [3]	$\mid$ int [4]

E	$\rightarrow$	$E+T$ [0]	$\mid$	T [1]		
T	$\rightarrow$	$T*F$ [0]	$\mid$	F [1]		
F	$\rightarrow$	$(\ E\ )$ [0]	$\mid$	name [1]	$\mid$	int [2]

Die beiden Grammatiken beschreiben die gleiche Sprache

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \rightarrow \ldots \rightarrow \alpha_m$ heißt Ableitung.

... im Beispiel: $\underline{E}$

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \rightarrow \ldots \rightarrow \alpha_m$ heißt Ableitung.

... im Beispiel: $\underline{E} \rightarrow \boxed{E}+\boxed{T}$

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \rightarrow \ldots \rightarrow \alpha_m$ heißt Ableitung.

... im Beispiel:
$$\begin{aligned} \underline{E} &\rightarrow \underline{E}+T \\ &\rightarrow \underline{T}+T \end{aligned}$$

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \to \ldots \to \alpha_m$ heißt Ableitung.

... im Beispiel:

$$\begin{aligned} \underline{E} &\to \underline{E} + T \\ &\to \underline{T} + T \\ &\to T * \underline{F} + T \end{aligned}$$

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \to \ldots \to \alpha_m$ heißt Ableitung.

... im Beispiel:

$$\begin{aligned} \underline{E} &\to \underline{E} + T \\ &\to \underline{T} + T \\ &\to T * \underline{F} + T \\ &\to \end{aligned}$$

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \to \ldots \to \alpha_m$ heißt Ableitung.

... im Beispiel:

$$\begin{aligned} \underline{E} &\to \underline{E} + T \\ &\to \underline{T} + T \\ &\to T * \underline{F} + T \\ &\to \underline{T} * \text{int} + T \\ &\to \underline{F} * \text{int} + T \\ &\to \text{name} * \text{int} + \underline{T} \\ &\to \text{name} * \text{int} + \underline{F} \\ &\to \end{aligned}$$

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \to \ldots \to \alpha_m$ heißt Ableitung.

... im Beispiel:

$$\begin{aligned} \underline{E} &\to \underline{E} + T \\ &\to \underline{T} + T \\ &\to T * \underline{F} + T \\ &\to \underline{T} * \text{int} + T \\ &\to \underline{F} * \text{int} + T \\ &\to \text{name} * \text{int} + \underline{T} \\ &\to \text{name} * \text{int} + \underline{F} \\ &\to \text{name} * \text{int} + \text{int} \end{aligned}$$

Definition

Eine Ableitung $\to$ ist eine Relation auf Wörtern über $N \cup T$, mit

$$\alpha \to \alpha' \quad \text{gdw.} \quad \alpha = \alpha_1 A \alpha_2 \ \wedge \ \alpha' = \alpha_1 \beta \alpha_2 \text{ für ein } A \to \beta \in P$$

Ableitung

Grammatiken sind Wortersetzungssysteme. Die Regeln geben die möglichen Ersetzungsschritte an. Eine Folge solcher Ersetzungsschritte $\alpha_0 \to \ldots \to \alpha_m$ heißt Ableitung.

... im Beispiel:

$$\begin{aligned} \underline{E} &\to \underline{E} + T \\ &\to \underline{T} + T \\ &\to T * \underline{F} + T \\ &\to \underline{T} * \text{int} + T \\ &\to \underline{F} * \text{int} + T \\ &\to \text{name} * \text{int} + \underline{T} \\ &\to \text{name} * \text{int} + \underline{F} \\ &\to \text{name} * \text{int} + \text{int} \end{aligned}$$

Definition

Eine Ableitung $\to$ ist eine Relation auf Wörtern über $N \cup T$, mit

$\alpha \to \alpha'$ gdw. $\alpha = \alpha_1 A \alpha_2 \ \wedge \ \alpha' = \alpha_1 \beta \alpha_2$ für ein $A \to \beta \in P$

Den reflexiven und transitiven Abschluss von $\to$ schreiben wir: $\to^*$

Ableitung

Bemerkungen:

- Die Relation $\to$ hängt von der Grammatik ab
- In jedem Schritt einer Ableitung können wir:
  - eine Stelle auswählen, wo wir ersetzen wollen, sowie
  - eine Regel, wie wir ersetzen wollen.
- Die von G spezifizierte Sprache ist:

$$\mathcal{L}(G) = \{w \in T^* \mid S \to^* w\}$$

Ableitung

Bemerkungen:

- Die Relation $\to$ hängt von der Grammatik ab
- In jedem Schritt einer Ableitung können wir:
  - eine Stelle auswählen, wo wir ersetzen wollen, sowie
  - eine Regel, wie wir ersetzen wollen.
- Die von G spezifizierte Sprache ist:

$$\mathcal{L}(G) = \{w \in T^* \mid S \to^* w\}$$

Achtung:

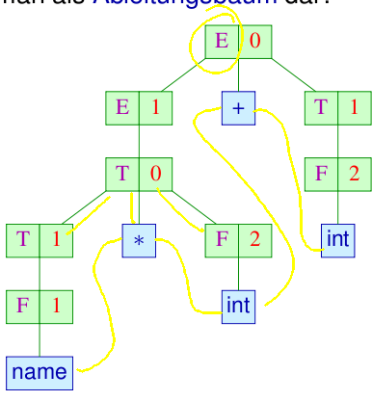
Die Reihenfolge, in der disjunkte Teile abgeleitet werden, ist unerheblich

Ableitungsbaum

Ableitungen eines Symbols stellt man als Ableitungsbaum dar:

... im Beispiel:

$$\begin{aligned} \underline{E} &\to^0 \underline{E} + T \\ &\to^1 \underline{T} + T \\ &\to^0 T * \underline{F} + T \\ &\to^2 \underline{T} * \text{int} + T \\ &\to^1 \underline{F} * \text{int} + T \\ &\to^1 \text{name} * \text{int} + \underline{T} \\ &\to^1 \text{name} * \text{int} + \underline{F} \\ &\to^2 \text{name} * \text{int} + \text{int} \end{aligned}$$



Ein Ableitungsbaum für $A \in N$:

innere Knoten: Regel-Anwendungen

Wurzel: Regel-Anwendung für A

Blätter: Terminale oder ϵ

Die Nachfolger von (B, i) entsprechen der rechten Seite der Regel

Spezielle Ableitungen

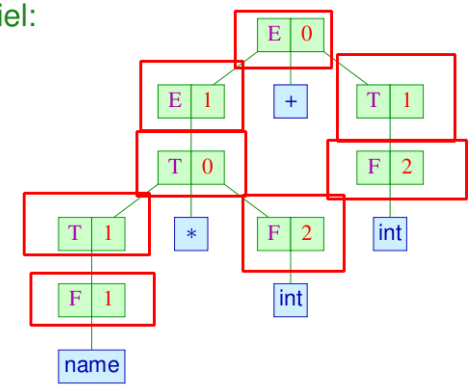
Beachte:

Neben beliebigen Ableitungen betrachtet man solche, bei denen stets das linkeste (bzw. rechteste) Vorkommen eines Nichtterminals ersetzt wird.

- Diese heißen Links- (bzw. Rechts-) Ableitungen und werden durch Index L bzw. R gekennzeichnet.
- Links-(bzw. Rechts-) Ableitungen entsprechen einem links-rechts (bzw. rechts-links) preorder-DFS-Durchlauf durch den Ableitungsbaum
- Reverse Rechts-Ableitungen entsprechen einem links-rechts postorder-DFS-Durchlauf durch den Ableitungsbaum

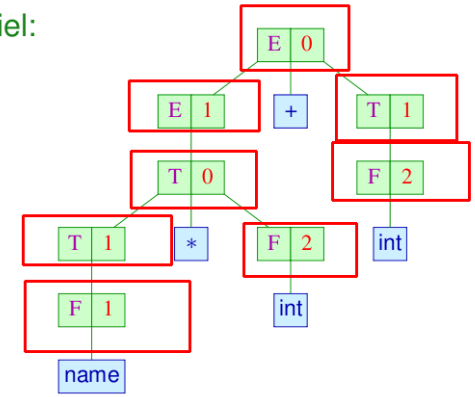
Spezielle Ableitungen

... im Beispiel:



Spezielle Ableitungen

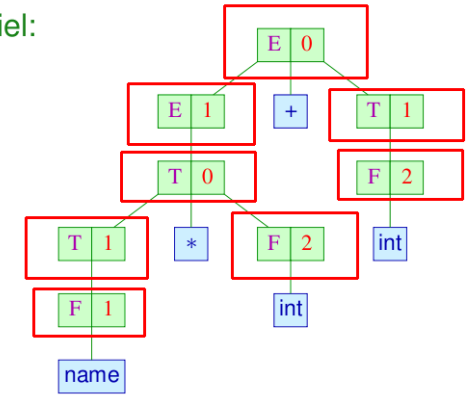
... im Beispiel:



Links-Ableitung: $(E,0)\,(E,1)\,(T,0)\,(T,1)\,(F,1)\,(F,2)\,(T,1)\,(F,2)$

Spezielle Ableitungen

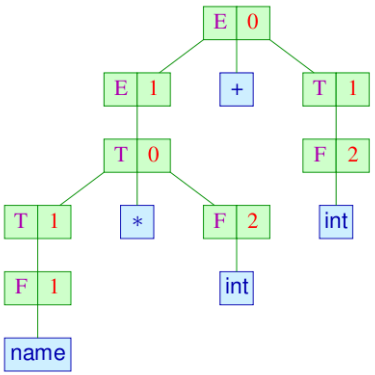
... im Beispiel:



Links-Ableitung: $(E,0)\,(E,1)\,(T,0)\,(T,1)\,(F,1)\,(F,2)\,(T,1)\,(F,2)$
Rechts-Ableitung: $(E,0)\,(T,1)\,(F,2)\,(E,1)\,(T,0)\,(F,2)\,(T,1)\,(F,1)$

Spezielle Ableitungen

... im Beispiel:



Links-Ableitung: $(E,0)\,(E,1)\,(T,0)\,(T,1)\,(F,1)\,(F,2)\,(T,1)\,(F,2)$

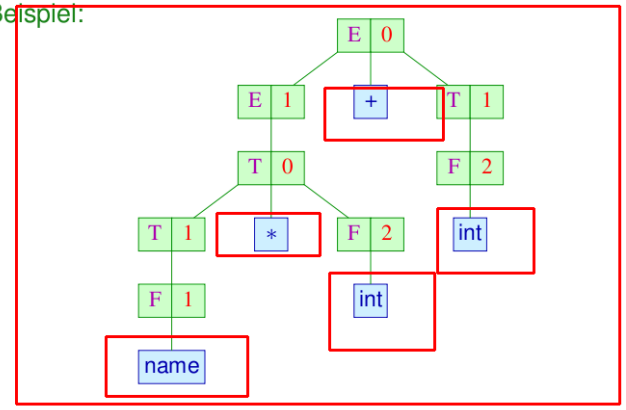
Rechts-Ableitung: $(E,0)\,(T,1)\,(F,2)\,(E,1)\,(T,0)\,(F,2)\,(T,1)\,(F,1)$

Reverse Rechts-Ableitung: $(F,1)\,(T,1)\,(F,2)\,(T,0)\,(E,1)\,(F,2)\,(T,1)\,(E,0)$

Eindeutige Grammatiken

Die Konkatenation der Blätter des Ableitungsbaums t bezeichnen wir auch mit $\text{yield}(t)$.

... im Beispiel:



liefert die Konkatenation: $\text{name} * \text{int} + \text{int}$.

Eindeutige Grammatiken

Definition:

Die Grammatik G heißt eindeutig, falls es zu jedem $w \in T^*$ maximal einen Ableitungsbaum t von S gibt mit $\text{yield}(t) = w$.

... in unserem Beispiel:

$$E \to E{+}E\ ^0 \mid E{*}E\ ^1 \mid (\ E\ )\ ^2 \mid \text{name}\ ^3 \mid \text{int}\ ^4$$

$$\begin{array}{lcl} E & \to & E{+}T\ ^0 \mid T\ ^1 \\ T & \to & T{*}F\ ^0 \mid F\ ^1 \\ F & \to & (\ E\ )\ ^0 \mid \text{name}\ ^1 \mid \text{int}\ ^2 \end{array}$$

Die zweite ist eindeutig, die erste nicht

Fazit:

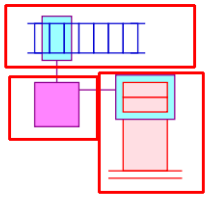
- Ein Ableitungsbaum repräsentiert eine mögliche hierarchische Struktur eines Worts.
- Bei Programmiersprachen sind wir nur an Grammatiken interessiert, bei denen die Struktur stets eindeutig ist
- Ableitungsbäume stehen in eins-zu-eins-Korrespondenz mit Links-Ableitungen wie auch (reversen) Rechts-Ableitungen.
- Links-Ableitungen entsprechen einem Topdown-Aufbau des Ableitungsbaums.
- Reverse Rechts-Ableitungen entsprechen einem Bottom-up-Aufbau des Ableitungsbaums.

Kapitel 2:

Grundlagen: Kellerautomaten

Grundlagen: Kellerautomaten

Durch kontextfreie Grammatiken spezifizierte Sprachen können durch Kellerautomaten (Pushdown Automata) akzeptiert werden:



Der Keller wird z.B. benötigt, um korrekte Klammerung zu überprüfen

Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

Konventionen:

- Wir unterscheiden nicht zwischen Kellersymbolen und Zuständen
- Das rechteste / oberste Kellersymbol repräsentiert den Zustand
- Jeder Übergang liest / modifiziert einen oberen Abschnitt des Kellers

... im Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$(0,\ aaabbb) \vdash (11,\ aabbb)$$

... im Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{aligned}(0,\ aaabbb) &\vdash (11,\ aabbb)\\ &\vdash (111,\ abbb)\end{aligned}$$

... im Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{aligned}(0,\ aaabbb) &\vdash (11,\ aabbb)\\ &\vdash (111,\ abbb)\\ &\vdash (1111,\ bbb)\end{aligned}$$

... im Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{aligned}(0,\ aaabbb) &\vdash (11,\ aabbb)\\ &\vdash (111,\ abbb)\\ &\vdash (1111,\ bbb)\\ &\vdash (112,\ bb)\end{aligned}$$

... im Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{aligned}(0,\ aaabbb) &\vdash (11,\ aabbb)\\ &\vdash (111,\ abbb)\\ &\vdash (1111,\ bbb)\\ &\vdash (112,\ bb)\\ &\vdash (12,\ b)\end{aligned}$$

... im Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

$$\begin{aligned}(0,\ aaabbb) &\vdash (11,\ aabbb)\\ &\vdash (111,\ abbb)\\ &\vdash (1111,\ bbb)\\ &\vdash (112,\ bb)\\ &\vdash (12,\ b)\end{aligned}$$

... im Beispiel:

Zustände: $0, 1, 2$
Anfangszustand: 0
Endzustände: $0, 2$

0	a	11
1	a	11
11	b	2
12	b	2

Ein Berechnungsschritt wird durch die Relation $\vdash \subseteq (Q^* \times T^*)^2$ beschrieben, wobei

$$(\alpha\gamma, xw) \vdash (\alpha\gamma', w) \quad \text{für} \quad (\gamma, x, \gamma') \in \delta$$

Bemerkungen:

- Die Relation $\vdash$ hängt natürlich vom Kellerautomaten M ab
- Die reflexive und transitive Hülle von $\vdash$ bezeichnen wir mit $\vdash^*$
- Dann ist die von M akzeptierte Sprache:

$$\mathcal{L}(M) = \{w \in T^* \mid \exists f \in F : (q_0, w) \vdash^* (f, \epsilon)\}$$

Ein Berechnungsschritt wird durch die Relation $\vdash \subseteq (Q^* \times T^*)^2$ beschrieben, wobei

$$(\alpha\gamma, xw) \vdash (\alpha\gamma', w) \quad \text{für} \quad (\gamma, x, \gamma') \in \delta$$

Bemerkungen:

- Die Relation $\vdash$ hängt natürlich vom Kellerautomaten M ab
- Die reflexive und transitive Hülle von $\vdash$ bezeichnen wir mit $\vdash^*$
- Dann ist die von M akzeptierte Sprache:

$$\mathcal{L}(M) = \{w \in T^* \mid \exists f \in F : (q_0, w) \vdash^* (f, \epsilon)\}$$

Wir akzeptieren also mit Endzustand und leerem Keller

Deterministischer Kellerautomat

Definition:

Der Kellerautomat M heißt deterministisch, falls jede Konfiguration maximal eine Nachfolge-Konfiguration hat.

Das ist genau dann der Fall wenn für verschiedene Übergänge (γ_1, x, γ_2), $(\gamma_1', x', \gamma_2') \in \delta$ gilt:
Ist γ_1 ein Suffix von γ_1', dann muss $x \neq x' \ \wedge \ x \neq \epsilon \neq x'$ sein.

... im Beispiel:

0	a	11
1	a	11
11	b	2
12	b	2

... ist das natürlich der Fall

Kellerautomaten


M. Schützenberger


Antony Öttinger

Satz:

Zu jeder kontextfreien Grammatik $G = (N, T, P, S)$ kann ein Kellerautomat M konstruiert werden mit $\mathcal{L}(G) = \mathcal{L}(M)$.

Der Satz ist für uns so wichtig, dass wir zwei Konstruktionen für Automaten angeben, motiviert durch die beiden speziellen Ableitungsmöglichkeiten:

- M_G^L zur Bildung von Linksableitungen
- M_G^R zur Bildung von reversen Rechtsableitungen

Kapitel 3:

Top-down Parsing