Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Sprachtechnologie und Compiler WS 2008/2009 — http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting — snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Matthias Braun — braun@ipd.info.uni-karlsruhe.de

Übungsblatt 7 | Ausgabe: 7.12.2008 | Besprechung: 10.12.2008

Aufgabe 1: *Attributierte Grammatiken*

Gegeben folgender Auszug aus der Grammatik einer Java-ähnlichen Sprache:

$$
\begin{aligned}
\mathit{CompilationUnit} &\rightarrow \mathit{Class}\ \mathit{CompilationUnit} \mid \epsilon \\
\mathit{Class} &\rightarrow \textbf{class id}\ \{\ \mathit{ClassMembers}\ \} \\
\mathit{ClassMember} &\rightarrow \mathit{Field} \mid \mathit{Method} \\
\mathit{Field} &\rightarrow \mathit{Type}\ \textbf{id}\ ; \\
\mathit{Method} &\rightarrow \mathit{Type}\ \textbf{id}\ \mathit{CompoundStatement} \\
\mathit{CompoundStatement} &\rightarrow \{\ \mathit{StatementList}\ \} \\
\mathit{StatementList} &\rightarrow \mathit{Statement}\ \mathit{StatementList} \mid \epsilon \\
\mathit{Statement} &\rightarrow \mathit{CompoundStatement} \mid ; \\
\mathit{Type} &\rightarrow \textbf{void}
\end{aligned}
$$

Weiter seien Funktionen mit folgenden Signaturen zum Aufbau eines AST gegeben:

$$
\begin{aligned}
\text{newCompilationUnit} &: \mathcal{P}(\mathit{Class}) \rightarrow \mathit{CompilationUnit} \\
\text{newClass} &: \mathit{Symbol} \times \mathcal{P}(\mathit{ClassMember}) \rightarrow \mathit{Class} \\
\mathit{newField} &: \mathit{Symbol} \times \mathit{Type} \rightarrow \mathit{ClassMember} \\
\mathit{newMethod} &: \mathit{Symbol} \times \mathit{Type} \times \mathit{Statement} \rightarrow \mathit{ClassMember} \\
\mathit{newCompoundStatement} &: \mathcal{P}(\mathit{Statement}) \rightarrow \mathit{Statement} \\
\mathit{emptyStatement} &: \mathit{Statement} \\
\mathit{voidType} &: \mathit{Type}
\end{aligned}
$$

Das Terminal **id** besitzt ein vordefiniertes Attribut $\mathit{symbol} : \mathit{Symbol}$.

Stellen Sie Attributierungsregeln auf, die zu einem Programm einen passendes AST erzeugen.

Aufgabe 2: *Auswertungsreihenfolgen*

2.1 *Eigenschaften*

Geben Sie attributierte Grammatiken die die folgenden Bedingungen erfüllen. Geben Sie zusätzlich zur verdeutlichung einen beliebigen Parsebaum zur verdeutlichung an.

Geben Sie eine AG an, die:

- Nur synthetisierte Attribute enthält.

- Nur ererbte Attribute enthält.
- Ein Attrib enthält das sowohl synthetisiert als auch ererebt wird.
- Stets mit einem Baumdurchlauf berechenbar ist.
- Mehr als einen Baumdurchlauf für ihre Berechnung benötigen.
- Nur für manche Bäume berechenbar ist. Es sollte sowohl berechenbare als auch nicht berechenbar Bäume geben, in denen alle Grammatikproduktionen vorkommen.

2.2 *Bedingungen*

Welche Bedingungen müssen gelten damit eine attributierte Grammatik während des LL parsens berechnet werden kann?

Aufgabe 3: *Auswertung Boolescher Ausdrücke*

Boolesche Ausdrücke können oft mit bedingten Sprüngen ausgewertet werden. Dabei muss jeweils die Adresse spezifiziert werden, an der die Berechnung nach dem Sprung fortgesetzt wird. Gegeben sei folgende Syntax Boolescher Ausdrücke:

conditional_clause	::=	`if` *boolean_expr* `then` *stmt_list* `else` *stmt_list* `end`
boolean_expr	::=	*boolean_expr* *boolean_op* *boolean_expr*
boolean_expr	::=	`not` *boolean_expr*
boolean_op	::=	`and` | `or`

Die Codeerzeugung soll aus dem AST erzeugen. Dafür müssen die Knoten mit Sprungmarken versehen werden, die mit bedingten Sprüngen angesprungen werden können. Die Funktion new_label erzeugt eine neue Sprungmarke. Das Attribut *location* gibt die Sprungmarke eines Knotens an. Die Attribute *jump_true* und *jump_false* die Sprungziele bei positiver bzw. negativer Auswertung eines Ausdrucks.

Geben Sie eine attributierte Grammatik an, die die Auswertung Boolescher Ausdrücke spezifiziert.