Prof. Dr. R. Loogen, M. Dieterle
Fachbereich Mathematik und Informatik
Hans-Meerwein-Straße
D-35032 Marburg

15. November 2011

Philipps Universität Marburg

Übungen zu „Grundlagen des Compilerbau“, Winter 2011/12

Nr. 5, Abgabe der Aufgaben: 22. November 2011 vor der Vorlesung

---

Aufgaben

5.1 LL(2)-Grammatik

6 Punkte

Gegeben sei die Grammatik $G = (\{S, H, P\}, \{\texttt{\$}, \texttt{id}, \texttt{:}\}, P, S)$ mit den Produktionen:

$$
\begin{array}{lllll}
\text{P:} & S & \to & H\ \texttt{\$} \\
 & H & \to & P \\
 & & | & P\ H \\
 & P & \to & \texttt{id}\ \texttt{:} \\
 & & | & P\ \texttt{id}
\end{array}
$$

(a) Linksfaktorisieren Sie G und beseitigen Sie die Linksrekursion. / 1

(b) Konstruieren Sie die LL(1)-Analysetabelle zu der resultierenden Grammatik. Benennen Sie den vorhandenen Konflikt. / 3

(c) Zeigen Sie, dass der auftretende Konflikt durch einen Lookahead von zwei aufgelöst werden kann. / 2

Bitte wenden!

5.2 Top-Down-Parser für eine While-Sprache

6 Punkte

Die Grammatik in Abb. 1 beschreibt eine While-Sprache ähnlich wie auf Blatt 3 und eignet sich für Top-Down-Parsing. Die Sprache ist allerdings nur partiell definiert, ab dem Level der Ausdrücke wurde die Sprache "flachgeklopft". Ausdrücke werden als Terminalsymbole (Token) behandelt.

Aufgabe: Programmieren Sie mit den Parser-Kombinatoren der Vorlesung einen *Recursive-Descent*-Parser für diese While-Sprache, der mit einem Lookahead von 1 deterministisch arbeitet. Ihr Parser soll die Ausgabe des Scanners[1] (`[Token]`) einlesen und einen abstrakten Syntaxbaum für die Eingabe aufbauen bzw. bei Fehlern eine Meldung mit dem falschen Symbol für den *ersten* Fehler ausgeben.

Auf der Webseite der Vorlesung finden Sie eine Haskell-Datei `ProgTypePart.hs` mit Datentypdefinitionen für einen abstrakten Syntaxbaum der While-Sprache, zwei Testeingaben sowie ein passendes Scannermodul.

Grammatik			la_1**-Mengen**
program	→	**program** Id '{' *dec stmts* '}'	
dec	→	**var** *decls*	
decls	→	*varlist* ':' *type A*	
type	→	**int**	{**int**}
	|	**bool**	{**bool**}
A	→	';' *decls*	{';'}
	|	ε	{Id, **print**, **if**, **while**, '{'}
varlist	→	Id *B*	
B	→	',' Id *B*	{','}
	|	ε	{':'}
stmt	→	Id ':=' Expr	{Id}
	|	**print** Expr	{**print**}
	|	**if** Expr **then** '{'*stmts*'}' *maybeElse*	{**if**}
	|	**while** Expr **do** *stmt*	{**while**}
	|	'{' *stmts* '}'	{'{'}
maybeElse	→	**else** '{' *stmts* '}'	{**else**}
	|	ε	{';', '}'}
stmts	→	*stmt C*	
C	→	';' *stmts*	{';'}
	|	ε	{'}'}

Bezeichner (Id) seien wie auf Blatt 3 beschrieben.

Abbildung 1: Grammatik mit la_1-Mengen für eine While-Sprache

[1] eine modifizierte Version des Scanners aus Aufgabe 3.2