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

22. November 2011

Philipps Universität Marburg

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

**Nr. 6, Abgabe der Aufgaben: 29. November 2011 vor der Vorlesung**

---

## Aufgaben

### 6.1 Top-Down-Parser für eine While-Sprache II

6 Punkte

Die Grammatik in Abb. 1 erweitert die While-Sprache von Blatt 5 um Ausdrücke und eignet sich für Top-Down-Parsing.

**Aufgabe:** Programmieren Sie mit Parsec einen *Recursive-Descent*-Parser für die erweiterte While-Sprache. Benutzen Sie nicht die vordefinierte Parsergenerierungsfunktion `buildExpressionParser`. Auf der Webseite der Vorlesung finden Sie eine Umsetzung von Aufgabe 5.3 für Parsec, diese können Sie anpassen und erweitern. Des weiteren sind die Haskell-Datei `ProgType.hs` mit Datentypdefinitionen, zwei Testeingaben sowie ein passendes Scannermodul auf der VL-Seite bereitgestellt. Ihr Parser soll die Ausgabe des Scanners[1] (`[WhileToken]`) einlesen und einen abstrakten Syntaxbaum für die Eingabe aufbauen.

| | | |
|---|---|---|
| *program* | → | **program** Id '{' *dec stmts* '}' |
| *dec* | → | **var** *decls* |
| *decls* | → | *varlist* ':' *type A* |
| *type* | → | **int** |
| | \| | **bool** |
| *A* | → | ';' *decls* |
| | \| | $\varepsilon$ |
| *varlist* | → | Id *B* |
| *B* | → | ',' Id *B* |
| | \| | $\varepsilon$ |
| *stmt* | → | Id ':=' *expr* |
| | \| | **print** *expr* |
| | \| | **if** *expr* **then** '{'*stmts*'}' *maybeElse* |
| | \| | **while** *expr* **do** *stmt* |
| | \| | '{' *stmts* '}' |
| *maybeElse* | → | **else** '{' *stmts* '}' |
| | \| | $\varepsilon$ |
| *stmts* | → | *stmt C* |
| *C* | → | ';' *stmts* |
| | \| | $\varepsilon$ |

| | | |
|---|---|---|
| *expr* | → | *T expr'* |
| *expr'* | → | '+' *T expr'* |
| | \| | '-' *T expr'* |
| | \| | '\|' *T expr'* |
| | \| | $\varepsilon$ |
| *T* | → | *F T'* |
| *T'* | → | '*' *F T'* |
| | \| | '/' *F T'* |
| | \| | '&' *F T'* |
| | \| | $\varepsilon$ |
| *F* | → | '{' *expr* RelOp *expr* '}' |
| | \| | '(' *expr* ')' |
| | \| | Id |
| | \| | BoolVal |
| | \| | NumVal |

Bezeichner (Id), Werte (*Val) und Operatoren aus *RelOp* seien wie auf Blatt 3 beschrieben. Ausgehend von den Symbolklassen von Blatt 3 ist die Sprache um die booleschen Operatoren '&' und '|' erweitert, wobei die Operatoren aus *ArithOp* nun jeweils durch eigene Symbolklassen repräsentiert werden.

Abbildung 1: Grammatik für eine While-Sprache

[1] eine modifizierte Version des Scanners aus Aufgabe 3.2

Bitte wenden!

## 6.2 **Parserkombinatoren** 3 Punkte

(a) Definieren Sie mit den Definitionen der Vorlesung einen Parserkombinator / 1

```
alternating :: Parser tok a → Parser tok b → Parser tok [(a,b)]
```

um eine alternierende Folge zweier verschiedener Elemente zu erkennen, in der auf ein Element erster Art immer ein Element zweiter Art folgt. Die Parameter des Kombinators sind Parser für die Elemente, der Parser liefert die erkannte Sequenz als Liste von Elementpaaren.

(b) Definieren Sie mit `alternating` einen Parser, der eine durch einzelne Leerzeichen (`sat isBlanc`) getrennte und mit einem Leerzeichen beendete Liste von Initialisierungen *Name*=*Wert* erkennt, wobei *Name:* [`a`−`z`]+, *Wert:* [`0`−`9`]+. / 2

## 6.3 **Bottom-Up-Parsing** 3 Punkte

Gegeben sei die folgende, startseparierte Grammatik $G$. Zeigen Sie, dass die Grammatik LR(0) ist.

$$
\begin{array}{lll}
G: & S' \to & S \\
 & S \to & (a) \mid () \mid (T) \\
 & T \to & T, S \mid S
\end{array}
$$