Prof. Dr. Arnd Poetzsch-Heffter
Dr.-Ing. Jan Schäfer

Technische Universität Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Übungsblatt 4: Übersetzer und sprachverarbeitende Werkzeuge (SS 2011)

Hand Out: 11. Mai 2011
Hand In: 18. Mai 2011

Aufgabe 1 $LL(k)$-Grammatiken

a) Geben Sie zu den folgenden regulären Ausdrücken die kontexfreien Grammatiken, welche die gleichen Sprachen beschreiben, an:

   1. $((xy^*x) \mid (yx^*y))?$
   2. $((0 \mid 1)^+\text{“.”}(0 \mid 1)^*) \mid ((0 \mid 1)^*\text{“.”}(0 \mid 1)^+)$

b) Geben Sie zu den regulären Ausdrücken aus der ersten Teilaufgabe jeweils eine LL(1)-Grammatik an. Lösen Sie das Problem, indem Sie in den von Ihnen definierten kf-Grammatiken die Mehrdeutigkeiten und Linksrekursionen entfernen und, falls nötig, die Grammatiken linksfaktorisieren (*left factoring*). Eine Definition der Linksfaktorisierung finden Sie im Buch “Modern Compiler Implementation in Java”, Andrew W. Appel, Seite 55.

Aufgabe 2 Parser für MiniJava

In den nächsten Übungen werden Sie einen Compiler für eine einfache Java-Teilsprache, `MiniJava`, entwickeln. Unten ist die erste Fassung ihrer Grammatik angegeben. Sie orientiert sich an der Java Language Specification, 2. Edition.

Ihre Aufgabe diese Woche, ist es mit Hilfe von JavaCup und JFlex eine Kombination aus Scanner und Parser zu entwickeln. Verwenden Sie dazu die `MiniJava.flex` Datei von der Webseite. Ihr Parser soll direkt die angegebene Grammatik umsetzen. Schreiben Sie ebenfalls ein Hauptprogramm, dass eine als Parameter übergebene Datei mit dem Parser analysiert und im Fehlerfall ausgibt, in welcher Zeile und Spalte, sowie bei welchem Token, der Fehler aufgetreten ist.

Hinweise zur Kooperation von JFlex und JavaCup:

- Die Datei `java_cup.jar` ist zum Ausführen und Kompilieren eines erzeugten Parsers im Classpath nötig. Es enthält u.a. eine Klasse `java_cup.runtime.Symbol`, die als Tokenrepräsentation oder zumindest als Superklasse einer eigenen Klasse verwendet werden muss.

  In den Aktionen des Lexer muss jeweils ein Objekt des Typs `java_cup.runtime.Symbol` erzeugt und zurückgeliefert werden.

- JavaCup erzeugt, nebem dem Parser, auch noch eine Klasse mit einer Konstante pro definiertem Nichtterminaltyp, diese heißt standardmäßig `sym`.

- JFlex unterstützt die Anbindung an einen JavaCup-Parser direkt mit der Direktive `%cup`. Lesen Sie dazu im Handbuch von JFlex nach.

  Ein Lexer, der mit `%cup` erzeugt wurde kann direkt von einem JavaCup-erzeugten Parser verwendet werden, dazu muss er nur dem Konstruktor des Parsers übergeben werden.

MiniJava Grammatik

Hinweis: Die Grammatik enthält einen Shift/Reduce-Konflikt (`if-then-else`-Problem). Der Konflikt muss nicht aufgelöst werden. Verwenden Sie die JavaCUP-Option `-expect 1` um den Konflikt zu ignorieren.

```
Literal ::= INTEGER_LITERAL | BooleanLiteral;
BooleanLiteral ::= TRUE | FALSE;
Infixop ::= EQEQ | GTEQ | PLUS | MINUS | MULT | DIV;
PrefixOp ::= PLUS | MINUS;
Modifiers ::= | STATIC;
Type ::= INT | BOOLEAN;

Expression ::= Expression1 ExpressionRest;
ExpressionRest ::=
   | Infixop Expression1 ExpressionRest;
Expression1 ::= PrefixOp Expression1
   | Primary;
Primary ::= ParExpression
   | Literal
   | IDENTIFIER IdentifierRest;
IdentifierRest ::=
   | Arguments
   | DOT IDENTIFIER IdentifierRest;
Arguments ::= LPAR RPAR
   | LPAR Expressions RPAR;
Expressions ::= Expression COMMA Expressions
   | Expression;
ParExpression ::= LPAR Expression RPAR;

Statement ::= Block
   | IF ParExpression Statement
   | IF ParExpression Statement ELSE Statement
   | WHILE ParExpression Statement
   | RETURN Expression SEMI
   | RETURN SEMI
   | SEMI
   | ExpressionStatement;
Block ::= LBRACK RBRACK
   | LBRACK BlockStatements RBRACK;
BlockStatements ::= BlockStatement BlockStatements
   | BlockStatement;
BlockStatement ::= LocalVariableDeclarationStatement
   | Statement;
LocalVariableDeclarationStatement ::= Type VariableDeclarators;
ExpressionStatement ::= Expression SEMI;

VariableDeclarators ::= VariableDeclarator COMMA VariableDeclarator
   | VariableDeclarator;
VariableDeclarator ::= IDENTIFIER VariableDeclaratorRest;
VariableDeclaratorRest ::= EQ ExpressionStatement;

CompilationUnit ::= TypeDeclarationList;
TypeDeclarationList ::= TypeDeclaration TypeDeclarationList
   | TypeDeclaration;
TypeDeclaration ::= ClassDeclaration;
ClassDeclaration ::= CLASS IDENTIFIER ClassBody;
ClassBody ::= LBRACK ClassBodyDeclarations RBRACK
   | LBRACK RBRACK;
ClassBodyDeclarations ::= ClassBodyDeclaration ClassBodyDeclarations
   | ClassBodyDeclaration;
ClassBodyDeclaration ::= SEMI
   | Modifiers MemberDecl;
```

```
MemberDecl ::= MethodOrFieldDecl
   | VOID IDENTIFIER MethodDeclaratorRest;
MethodOrFieldDecl ::= Type IDENTIFIER MethodOrFieldRest;
MethodOrFieldRest ::= VariableDeclaratorRest
   | SEMI
   | MethodDeclaratorRest;
MethodDeclaratorRest ::= LPAR FormalParameters RPAR Block
   | LPAR RPAR Block;
FormalParameters ::= FormalParameter COMMA FormalParameters
   | FormalParameter;
FormalParameter ::= Type IDENTIFIER;
```