Faculty of Computer Science
Free University of Bozen-Bolzano
Alessandro Artale

Compilers Exam – 10.July.2008

STUDENT NAME:
STUDENT NUMBER:
STUDENT SIGNATURE:

This Exam will constitute the 70% of the overall course assessment.

1 Exercise: Grammar Rewriting

Consider the following ambiguous grammar for Arithmetic expressions (where id stands for the terminal "identifier"):

$$E \to E + E \mid E - E \mid E * E \mid E \div E \mid \mathsf{id}$$

1. Explain why the grammar is ambiguous with an example sentence.
2. Rewrite the grammar eliminating ambiguity by considering the Arithmetic operators as left-associative and the following precedence: $"+, -" \; < \; "*, \div"$.
3. Show that the example you provided in point 1 is no more ambiguous.
4. What information is transmitted by the lexical analyzer to the parser when it recognizes an identifier?

2 Exercise. Top-Down Parsing

Consider the following context-free grammar (with num terminal symbol):

$$\begin{array}{lcl} T & \to & -TT \mid +TT \mid (T) \mid R \\ R & \to & *RR \mid \mathsf{num} \end{array}$$

Show the following:

1. The value of the functions `FIRST` and `FOLLOW` for all the non terminal symbols.
2. The parsing table for the LL(1) Top Down Parser recognizing the above grammar.
3. The stack and the moves of the LL(1) parser on input: $+ -$ `5` $*$ `2 3 4`.

3 Exercise: Bottom-Up Parsing

Given the following Grammar for Statements with productions r1.–r6.

$$
\begin{array}{lrcl}
\text{r1.} & SList & \rightarrow & SList \ ; \ S \\
\text{r2.} & SList & \rightarrow & S \\
\text{r3.} & S & \rightarrow & \mathsf{id} \ = \ E \\
\text{r4.} & E & \rightarrow & E \ + \ \mathsf{id} \\
\text{r5.} & E & \rightarrow & (E) \\
\text{r6.} & E & \rightarrow & \mathtt{id}
\end{array}
$$

where:
$V_N = \{SList, S, E\}$, $V_T = \{\mathsf{id}, =, ;, +, (,)\}$ and $SList$ is the scope.

Show the following:

1. The canonical SLR collection
2. The transition diagram describing the automaton which recognizes handles at the top of the stack
3. The parsing table for the SLR parser
4. The stack and the moves of the SLR parser on input: $\mathsf{id}_1 = \mathsf{id}_2 + (\mathsf{id}_3)$
5. The first item I_0, and the item $I_1 = \text{goto}(I_0, SList)$ built following the LR(1) technique.

4 Exercise: Semantic Analysis

Given the following grammar together with the semantic rules:

PRODUCTION	SEMANTIC RULES
$Prog \to S$	$S.next := newlabel; Prog.code := S.code \parallel gen(S.next\ ':')$
$S \to S_1\ ;\ S_2$	$S_1.next := newlabel; S_2.next := S.next;$
	$S.code := S_1.code \parallel gen(S_1.next\ ':') \parallel S_2.code$
$S \to \mathsf{while}\ Test\ \mathsf{do}\ S_1$	$Test.begin := newlabel;\ Test.true := newlabel;$
	$Test.false := S.next;\ S_1.next := Test.begin;$
	$S.code := gen(Test.begin\ ':') \parallel Test.code \parallel gen(Test.true\ ':') \parallel$
	$S_1.code \parallel gen('\mathsf{goto}'\ Test.begin)$
$S \to \mathsf{id} := E$	$S.code := E.code \parallel gen(\mathsf{id}.place\ ':='\ E.place)$
$Test \to \mathsf{id}_1\ \mathsf{relop}\ \mathsf{id}_2$	$Test.code := gen('\mathsf{if}'\ \mathsf{id}_1.place\ \mathsf{relop}.op\ \mathsf{id}_2.place\ '\mathsf{goto}'\ Test.true) \parallel$
	$gen('\mathsf{goto}'\ Test.false)$
$E \to E_1 + \mathsf{id}$	$E.place := newtemp;$
	$E.code := E_1.code \parallel gen(E.place\ ':='\ E_1.place\ '+'\ \mathsf{id}.place)$
$E \to \mathsf{id}$	$E.place := \mathsf{id}.place; E.code :='\ '$

where:

- The function *newlabel* generates new symbolic labels.
- The function *newtemp* generates new variables names.
- The function *gen* generates strings such that everything in quotes is generated literally while the rest is evaluated.
- The attribute *code* produces the three-address code.
- The attribute $\mathsf{id}.place$ represents the name of the variable associated to the token id.
- The attribute $\mathsf{relop}.op$ represents the comparison operators (i.e., $<, <=, =, <>, >, >=$).
- The symbol $\parallel$ means string concatenation.

Given the input:

```
while x < y do
      z := a + b
```

Show the following:

1. The annotated parse tree (without the *code* attribute) for the input together with the values of the attributes;
2. The three-address code produced by the semantic actions for the given input.
3. The semantic rules for the production: $S \to \mathsf{if}\ Test\ \mathsf{then}\ S_1$.