Free University of Bozen-Bolzano

Faculty of Computer Science

Bachelor in Computer Science and Engineering

Alessandro Artale

Formal Languages and Compilers – A.Y. 2015/16 – 12.Sept.2016

Compiler Part

Time: 1^h50 minutes

STUDENT NAME:
STUDENT NUMBER:
STUDENT SIGNATURE:

This is a closed book exam. The use of Pencils is not allowed! Write clearly, in the sense of logic, language and legibility. The clarity of your explanations affects your grade. Write your name and ID on every solution sheet.

1 Exercise: Lexical Analyser [7 Points]

Lexeme, Token and Attribute. [3 Points]

1. Make an example where you show what is a *lexeme* what is a *token* and what is an *attribute*. [1 Point]

2. During what phase of the compilation we need the information attached to the Token? and in what phase is needed the information attached to an Attribute? [2 Points]

Conflicts in lexical analysis. [4 Points]

3. During the lexical analysis there are 2 kinds of conflicts:

   Case 1. The same Lexeme is recognized by two different RE's.

   Case 2. A given RE can recognize portion of a Lexeme.

   For the Case 2, describe the 2 different techniques that can solve it. [4 Points]

2 Exercise. Top-Down Parsing [8 Points]

Given the following grammar with terminals $VT = \{\mathsf{num}, +, *, (,), ,\}$, where num stands for the terminal "number'.

$$
\begin{array}{rcl}
\texttt{E} & \rightarrow & \texttt{LE RE} \\
\texttt{RE} & \rightarrow & * \texttt{ E} \mid \epsilon \\
\texttt{LE} & \rightarrow & \textbf{num} \mid (\ + \texttt{ LS }) \\
\texttt{LS} & \rightarrow & \textbf{num LSR} \\
\texttt{LSR} & \rightarrow & , \texttt{ LS} \mid \epsilon
\end{array}
$$

1. Show the value of the function `FIRST` for all the non terminal symbols. [1 Point]

2. Show the value of the function `FOLLOW` for all the non terminal symbols. [1 Point]

3. Show the parsing table for the LL(1) Top Down Parser recognizing the grammar. [3 Points]

4. Show how the stack of the LL(1) parser evolves and the resulting parse tree for the input: "$3 * (+ 6, 9)$". [3 Points]

3 Exercise: Bottom-Up Parsing [14 Points]

Consider the following grammar with terminals $VT = \{*, \texttt{COS}, \texttt{id}\}$:

$$\texttt{E} \rightarrow \texttt{COS E} \mid \texttt{E*E} \mid \texttt{id}$$

1. Show that the above Grammar is not SLR by showing a state with a conflict. In particular, describe: [4 Points]
   - The kind of conflict;
   - The default decision taken by `YACC` when dealing with such a conflict;
   - A technique to eliminate the conflict.

Consider now the following grammar with terminals $VT = \{\mathsf{array}, \mathsf{of}, \mathsf{int}, \mathsf{id}, \mathsf{num}, ,, ;\}$

$$\begin{array}{rcl} \texttt{SL} & \rightarrow & \texttt{S ; SL} \mid \texttt{S} \\ \texttt{S} & \rightarrow & \texttt{T VL} \\ \texttt{T} & \rightarrow & \textbf{array num of } \texttt{T} \mid \textbf{int} \\ \texttt{VL} & \rightarrow & \textbf{id} \texttt{ , VL} \mid \textbf{id} \end{array}$$

Show the following:

2. The canonical SLR collection. [4 Points]
3. The transition diagram describing the automaton which recognizes handles at the top of the stack. [2 Points]
4. The parsing table for the SLR parser. Show in full the table for the states I_0 and I_1 while for all the other states show just the reduce actions. [2 Points]
5. The stack and the moves of the SLR parser on input: "array 10 of int x". [2 Points]

4 Exercise: Semantic Analysis [4 Points]

1. Complete the following syntax directed definition:

Production	Semantic Rules
Decl → T : VL	**?**
T → **int**	T.type = **?**
T → **real**	T.type = **?**
T → **vect** [**num**] **of** T_1	T.type = vect(num.val, **?**)
VL → VL_1, **id**	VL_1.type = **?**
VL → **id**	*addtype*(id.ptr, **?**)

Where *addtype* is a function that adds to the symbol table entry id.ptr its type. [2 Points]

Given the following syntax directed definition:

Production	Semantic Rules
$Prog \rightarrow S$	$S.next := newlabel; Prog.code := S.code \parallel gen(S.next\ ':')$
$S \rightarrow S_1\ ;\ S_2$	$S_1.next := newlabel; S_2.next := S.next;$
	$S.code := S_1.code \parallel gen(S_1.next\ ':') \parallel S_2.code$
$S \rightarrow$ if $Test$ then $\{S_1\}$	$Test.true := newlabel;\ Test.false := S.next;\ S_1.next := S.next;$
	$S.code := Test.code \parallel gen(Test.true\ ':') \parallel S_1.code$
...	...

Where:

- The function *newlabel* generates new symbolic labels: $l_1, l_2, \ldots$
- The function *gen* generates strings such that everything in quotes is generated literally while the rest is evaluated.
- The symbol $\parallel$ means string concatenation.

Show the following:

2. Discuss the notion of L-Attributed Definitions by giving the definition for such notion and individuate the inherited attributes of the production: $S \rightarrow$ if $Test$ then $\{S_1\}$. [2 Points]