University of Puerto Rico
Mayagüez Campus
College of Engineering
Department of Electrical and Computer Engineering

ICOM4029 – Compilers
Professor: Bienvenido Vélez
Technical Assistant: René D. Badía

Laboratory 6 – Syntactic Analysis (Continued)

I. Error Recovery

A good syntactic analyzer should be able to recover from syntax errors in the code and continue with its analysis as far as possible. With CUP, this is achieved by the use of the special *error* terminal.

1. Preparation

The files needed for this lab can be obtained by doing:

```
mkdir lab6
cd lab6
cp ~icom4029/labfiles/lab6/* .
```

2. Example 1 – Simple Expression Evaluator (example1.cup)

```
// CUP specification for a simple expression evaluator
// (w/ error recovery)
import java_cup.runtime.*;

action code {:
    int curr_lineno() {
      return (((ExampleLexer)parser.getScanner()).get_curr_lineno());
    }
:}

parser code {:
    public void syntax_error(Symbol cur_token) {
        int lineno = action_obj.curr_lineno();
        System.out.print("line " + lineno +
                         ": parse error at or near ");
        Utilities.printToken(cur_token);
    }

    public void unrecovered_syntax_error(Symbol cur_token) {
    }
:}

/* Terminals (tokens returned by the scanner). */
terminal                SEMI, PLUS, MINUS, MULT, DIV;
terminal                LPAREN, RPAREN;
terminal AbstractSymbol INT_CONST;
terminal                ERROR;
terminal                errP;

/* Non-terminals */
non terminal            expr_list, expr_part;
non terminal Integer    expr;
```

```
/* Precedences */
precedence left errP;
precedence left PLUS, MINUS;
precedence left MULT, DIV;

/* The grammar */
expr_list ::= expr_part expr_list | expr_part
            ;

expr_part ::= expr:e SEMI
            {: System.out.println("= " + e); :}
            | error expr:e SEMI
            {: System.out.println("= " + e); :}
            %prec errP
            ;

expr      ::= expr:e1 PLUS expr:e2
            {: RESULT = new Integer(e1.intValue() + e2.intValue()); :}
            | expr:e1 MINUS expr:e2
            {: RESULT = new Integer(e1.intValue() - e2.intValue()); :}
            | expr:e1 MULT expr:e2
            {: RESULT = new Integer(e1.intValue() * e2.intValue()); :}
            | expr:e1 DIV expr:e2
            {: RESULT = new Integer(e1.intValue() / e2.intValue()); :}
            | INT_CONST:i
            {: RESULT = new Integer(i.getString()); :}
            | LPAREN expr:e RPAREN
            {: RESULT = e; :}
            ;
```

3. Compiling and Running

To create the parser using CUP (all in one line):

```
java -classpath /home/courses/icom4029/icom4029/cool/lib:. java_cup.Main -
parser parser1 -symbols TokenConstants -expect 100 -nopositions <
example1.cup
```

To compile the java files (all in one line):

```
javac -classpath /home/courses/icom4029/icom4029/cool/lib:. parser1.java
TokenConstants.java
```

To run the parser:

```
./parser1 input.txt
```

The last output should be:

```
= 3
line 2: parse error at or near '-'
= 6
= 2
```

Which are the results of evaluating:

```
1 + 2;
1 - - 2 * 3;
6/(1+2);
```

II. LL(1) Parsing Table

1. LL(1) CFG:

```
L -> S L | ε
S -> E;
E -> A B
A -> ( E ) | int X
B -> + E | - E | ε
X -> * A | / A | ε
```

Note: L is different from the previous lab and therefore, from the example 1 grammar, because it was left-recursive. It should have been left-factored like this from the start:
Original:
```
L -> S L | S
```
Left-factored:
```
L -> S T
T -> L | ε
```

2. Compute the "First" Sets

```
First( ( ) = {(}                    First( L ) = {int, (}
First( ) ) = {)}                    First( S ) = {int, (}
First(int) = {int}                  First( E ) = {int, (}
First( + ) = {+}                    First( A ) = {int, (}
First( - ) = {-}                    First( B ) = {+, -, ε}
First( * ) = {*}                    First( X ) = {*, /, ε}
First( / ) = {/}                    First( ; ) = {;}
```

3. Compute the "Follow" Sets

Add $ to Follow(T) if T is the start non-terminal or T can end the grammar.

```
Follow( ( ) = {int, (}              Follow( L ) = {$}
Follow( ) ) = {+, -, )}             Follow( S ) = {int, (, $}
Follow(int) = {*, /, +, -, )}       Follow( E ) = {;, )}
Follow( + ) = {int, (}              Follow( A ) = {+, -, ), ;}
Follow( - ) = {int, (}              Follow( B ) = {;, )}
Follow( * ) = {int, (}              Follow( X ) = {+, -, ), ;}
Follow( / ) = {int, (}              Follow( ; ) = {int, (, $}
```

4. Construct the Parsing Table

Non-Terminals	Terminals								
	(	)	int	+	-	*	/	;	$
L	S L		S L						ε
S	E ;		E ;						
E	A B		A B						
A	(E)		int X						
B		ε		+ E	- E			ε	
X		ε		ε	E	* A	/ A	ε	

III. PA3 Discussion