Compiladores e Intérpretes
Informe de la Tercera Entrega

Diego Marcovecchio (LU: 83815)   Leonardo Molas (LU: 82498)

16 de Septiembre de 2010

Índice general

Introducción

Esta entrega consiste en el desarrollo del **Analizador Sintáctico** que formará parte del Compilado de Mini-Pascal. Para esto, a partir de un código fuente, se lee la sucesión de lexemas, con el Analizador Léxico LexAn, de la entrega anterior, y se verifica que esta sucesión sea generada por la gramática previamente entregada.

Se documentará también la modificación de la gramática, de manera tal que pase a ser LL(1).

1.2. Corrección de la gramática

Antes de comenzar con la modificación, se corrigieron los errores marcados por la cátedra.

```
<program> ::= <program heading> <block>.

<program heading> ::= program <identifier>;

<block> ::= <constant definition part><type definition part><variable declaration
      part><procedure and function declaration part><statement part>

<constant definition part> ::= <empty> | const <constant definition>{;<constant
      definition>};

<constant definition> ::= <identifier>=<constant>

<identifier> ::= <letter>{<letter or digit>}

<letter or digit> ::= <letter> | <digit>

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q |
      R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j |
      k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<constant> ::= <unsigned number> | <sign><unsigned number> | <constant
      identifier> | <sign><constant identifier> | <char>

<unsigned number> ::= <unsigned integer>

<unsigned integer> ::= <digit sequence>

<digit sequence> ::= <digit>{<digit>}

<sign> ::= + | -

<constant identifier> ::= <identifier>
```

```
<type definition part> ::= <empty> | type <type definition>{;<type definition>};

<type definition> ::= <identifier>=<type>

<type> ::= <simple type> | <structured type>

<simple type> ::= <subrange type> | <type identifier>

<subrange type> ::= <constant>..<constant>

<type identifier> ::= <identifier>

<structured type> ::= <unpacked structured type>

<unpacked structured type> ::= <array type>

<array type> ::= array[<index type>] of <component type>

<index type> ::= <simple type>

<component type> ::= <simple type>

<variable definition part> : := <empty> | var<variable declaration>{;<variable
     declaration>};

<variable declaration> ::= <identifier>{,<identifier>} : <type>

<procedure and function declaration part> ::= {<procedure or function declaration
     part>;}

<procedure or function declaration part> ::= <procedure declaration> | <function
     declaration>

<procedure declaration> ::= <procedure heading><block>

<procedure heading> ::= procedure <identifier>; | procedure <identifier>(<formal
     parameter section>{;<formal parameter section>});

<formal parameter section> ::= <parameter group> | var <parameter group>

<parameter group> ::= <identifier>{,<identifier>}:<type identifier>

<function declaration> ::= <function heading><block>

<function heading> ::= function<identifier>:<result type>; | <function
     identifier>(<formal parameter section>{;<formal parameter
     section>}):<result type>;

<result type> ::= <type identifier>

<statement part> ::= <compound statement>

<compound statement> ::= begin <statement>{;<statement>} end

<statement> ::= <unlabelled statement>

<unlabelled statement> ::= <simple statement> | <structured statement>

<simple statement> ::= <assignment statement> | <procedure statement> | <empty
     statement>
```

```
<assignment statement> ::= <variable>:=<expression> | <function
      identifier>:=<expression>

<variable> ::= <entire variable> | <component variable>

<entire variable> ::= <variable identifier>

<variable identifier> ::= <identifier>

<component variable> ::= <indexed variable>

<indexed variable> ::= <array variable>[<expression>]

<array variable> ::= <entire variable>

<expression> ::= <simple expression> | <simple expression><relational
      operator><simple expression>

<simple expression> ::= <term> | <simple expression><adding operator><term> |
      <sign><term>

<term>::= <factor> | <term><multiplying operator><factor>

<factor> ::= <variable> | <unsigned constant> | <function designator> |
      (<expression>) | not <factor> | <char>

<char> ::= ’<letter>’ | ’<digit>’

<unsigned constant> ::= <unsigned number> | <constant identifier>

<function designator> ::= <function identifier> | <function identifier>(<actual
      parameter>{,<actual parameter>})

<function identifier> ::= <identifier>

<actual parameter> ::= <expression> | <variable>

<multiplying operator> ::= * | div | and

<adding operator> ::= + | - | or

<relational operator> ::= = | <> | < | <= | >= | >

<procedure statement> ::= <procedure identifier> | <procedure identifier>(<actual
      parameter>{,<actual parameter>})

<procedure identifier> ::= <identifier>

<empty statement> ::= <empty>

<structured statement> ::= <compound statement> | <conditional statement> |
      <repetitive statement>

<conditional statement> ::= <if statement>

<if statement> ::= if <expression> then <statement> | if <expression> then
      <statement> else <statement>

<repetitive statement> ::= <while statement>

<while statement> ::= while <expression> do <statement>

<special symbol> ::= + | - | * | = | <> | < | > | <= | >= | ( | ) | [ | ] | { | }
      | := | . | , | ; | : | div | or | and | not | if | then | else | while | do
      | begin | end | const | var | type | array | function | procedure | program
```

1.3. Introducción de los tokens en la gramática

Como siguiente paso en la adaptación de la gramática, se reemplazaron los terminales por los tokens que devuelve **LexAn**. Para esto, se adoptó la convención de dejar los no terminales en minúscula, mientras que los tokens (terminales) se encuentran en MAYÚSCULA.

Capítulo 1

Gramática

1.1. Tokens

En la tabla 1.1 se encuentran todos los tokens con sus respectivos lexemas, como fue presentada en la entrega anterior, con sus debidas modificaciones.

Token	**Expresión Regular**
Identifier	`[a-zA-Z][a-zA-Z0-9]*`
Number	`[0-9]+`
Char	`'[a-zA-Z0-9]'`
RelOp	`<|>|<=|>=`
Arith_Op	`+|-|*`
Un_LogOp	`not`
Bin_LogOp	`or|and`
Equal	`=`
Type_Declaration	`:`
Assignment	`:=`
Comma	`,`
Semicolon	`;`
End_Program	`.`
Subrange_Separator	`..`
EOF	
Open_Parenthesis	`(`
Close_Parenthesis	`)`
Open_Bracket	`[`
Close_Bracket	`]`
Program	`program`
Type	`type`
Const	`const`
Var	`var`
Function	`function`
Procedure	`procedure`
Array	`array`
Of	`of`
Begin	`begin`
End	`end`
While	`while`
Do	`do`
If	`if`
Then	`then`
Else	`else`

Cuadro 1.1: Tokens