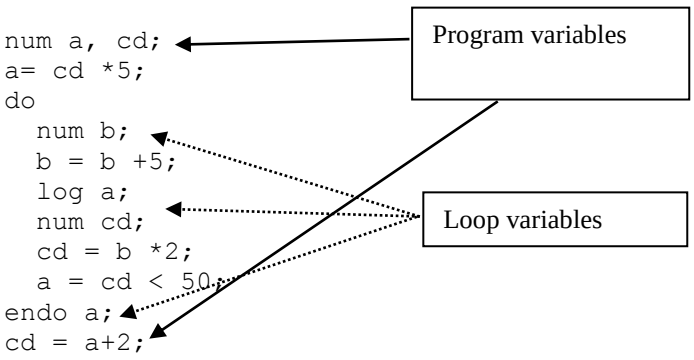
UNITS 7 AND 8: SEMANTIC ANALYSIS and ERROR HANDLING

Build a compiler for the following programming language. There are three types of statements: declaration, arithmetic / logical expression and loop and two types of data: numeric and logical. The sentences are described as follows:

- Declaration
  - **`type name_variable [, name_variable]* ;`**
  - where type can be **`num`** or **`log`** and **`name_variable`** is a char string with a máximum number of 8 characters
- Arithmetic/Logic expressions
  - **`name_variable = expression_arithmetic ;`**
  - **`name_variable = expression_logic;`**
  - Arithmetic expressions can contain variables of type **`num`** and numbers with the operators: +, -, *, /
  - The logical expression relates variables of type num with numbers through logical operators: <,>, =, #. The possible results of the evaluation of the logical expression is V or F. The variable to which it is assigned must be type log.
- Loop
  - **`do [sentence]+ endo [expression_logic | variable_logic] ;`**
  - where sentences can be declaration, expression or loop.

The sentences of the loop are executed at least once, and the loop is repeated while the logical expression in endo takes the value V. The variables are local to the loop where they are declared, if variables are used in the logical expression of the endo then there must be been declared inside the loop. The variables declared in the main body of the program are considered local to it.

Example of a program without errors:



David Griol Barres, Antonio Berlanga de Jesús, Jesús García Herrero, Juan Manuel Alonso Weber

It is required:

1. Define the G grammar that would generate valid sentences of this programming language.

2. Can the grammar G of the first exercise be used to perform an LL(1) analysis? If not, modify it so that it can. Generate the Table LL(1).

3. Construct the Table LR(1) that recognizes sentences of the language generated by the modified grammar of section 2, showing the states and the transitions between them.

4. Specify the lexical and semantic verifications for this language.

5. Describe the semantic routines (in pseudocode) necessary to perform the semantic verifications of the previous section associated with the LR(1) analyzer. If additional data structures are required, explain their usefulness.

Solution

A grammar that generates the given language can be defined as follows:

G = {**S**, **A**, **B**, **D**, **E**, **V**, **X**}, {**S}** , { ; , type, var, do, endo, =, o, p, “,” }

(1) **S**::= **A** ; **S**
(2) **S**::= **A** ;
(3) **A**::= **D**
(4) **A**::= **B**
(5) **A**::= **E**
(6) **D**::= type **V**
(7) **V**::= var
(8) **V**::= var , **V**
(9) **B**::= do **S** endo **X**
(10) **E**::= var = **X**
(11) **X**::= o
(12) **X**::= o p **X**

The token "type" represents the strings "num" and "log", the token "var" to the set of characters that identifies a variable, "o" is an operand (variable or number) and "p" is an operator, either of arithmetic or logical type.

After left-factoring, the modified G’ is:

G’ = {**S**, **S’**, **A**, **B**, **D**, **E**, **V, V’**, **X, X’**}, {**S}** , { ; , type, var, do, endo, =, o, p, “,” }

(1) **S**::= **A** ; **S’**
(2) **S’**::= **S**
(3) **S’**::= λ
(4) **A**::= **D**
(5) **A**::= **B**
(6) **A**::= **E**
(7) **D**::= type **V**
(8) **V**::= var **V’**
(9) **V’**::= , **V**
(10) **V’**::= λ
(11) **B**::= do **S** endo **X**
(12) **E**::= var = **X**
(13) **X**::= o **X’**
(14) **X’**::= p **X**
(15) **X’**::= λ

2

Σ_N	First				Follow	
S	type	do	var		$	endo
S'	type	do	var	λ	$	endo
A	type	do	var		;	
B	do				;	
D	type				;	
E	var				;	
V	var				;	
V'	λ	,			;	
X	o				;	
X'	λ	p			;	

Table LL(1)		Σ_T									
		$	**;**	**type**	**var**	**do**	**endo**	**=**	**o**	**P**	**,**
Σ_N	**S**			1	1	1					
	S'	3		2	2	2	3				
	A			4	6	5					
	B					11					
	D			7							
	E				12						
	V				8						
	V'		10								9
	X								13		
	X'		15							14	

David Griol Barres, Antonio Berlanga de Jesús, Jesús García Herrero, Juan Manuel Alonso Weber

3

State 0	Action	Go To
S''::= · S		[0,S]=1
S::= · A ; S'		[0,A]=2
A::= · D		[0,D]=3
A::= · B		[0,B]=4
A::= · E		[0,E]=5
D::= · type V	[0,type]=D6	
B::= · do S endo X	[0,do]=D7	
E::= · var = X	[0,var]=D8	
State 1	Action	Go To
S'::= S ·	[1,$]=ACP	
State 2	Action	Go To
S::= A · ; S'	[2,";"]=D9	
State 3	Action	Go To
A::=D ·	[3,";"]=R4	
State 4	Action	Go To
A::=B ·	[4,";"]=R5	
State 5	Action	Go To
A::=E ·	[5,";"]=R6	
State 6	Action	Go To
D::= type · V		[6,V]=10
V::= · var V'	[6,var]=D11	
State 7	Action	Go To
B::= do · S endo X		[7,S]=D12
S::= · A ; S'		[7,A]=2
A::= · D		[7,D]=3
A::= · B		[7,B]=4
A::= · E		[7,E]=5
D::= · type V	[7,type]=D6	
B::= · do S endo X	[7,do]=D7	
E::= · var = X	[7,var]=D8	
State 8	Action	Go To
E::= var · = X	[8,"="]=D13	
State 9	Action	Go To
S::= A ; · S'		[9,S']=14
S'::= · S		[9,S']=15
S'::= λ	[9,$]=R3	
	[9,endo]=R3	
S::= · A ; S'		[9,A]=2
A::= · D		[9,D]=3
A::= · B		[9,B]=4
A::= · E		[9,E]=5
D::= · type V	[9,type]=D6	
B::= · do S endo X	[9,do]=D7	
E::= · var = X	[9,var]=D8	
State 10	Action	Go To
D::= type V ·	[10,";"]=R7	
State 11	Action	Go To
V::= var · V'		[11,V']=16
V'::= · , V	[11,","]=D17	
V'::= λ	[11,";"]=R10	

State 12	Action	Go To
B::= do S · endo X	[12,endo]=D18	
State 13	Action	Go To
E::= var = · X		[13,X]=19
X::= · o X'	[13,o]=D21	
State 14	Action	Go To
S::= A ; S' ·	[14,$]=R1	
	[14,endo]=R1	
State 15	Action	Go To
S'::= S ·	[15,$]=R2	
	[15,endo]=R2	
State 16	Action	Go To
V::= var V' ·	[16,";"]=R8	
State 17	Action	Go To
V'::= , · V		[17,V]=22
V::= · var V'	[17,var]=D11	
State 18	Action	Go To
B::= do S endo · X		[18,X]=23
X::= · o X'	[18,o]=21	
State 19	Action	Go To
E::= var = X ·	[19,";"]=R12	
State 21	Action	Go To
X::= o · X'		[21,X']=24
X'::= · p X	[21,p]=D25	
X'::= λ	[21,";"]=R15	
State 22	Action	Go To
V'::= , V ·	[22,";"]=R9	
State 23	Action	Go To
B::= do S endo X ·	[23,";"]=R11	
State 24	Action	Go To
X::= o X' ·	[24,";"]=R13	
State 25	Action	Go To
X'::= p · X		[25,X]=26
X::= · o X'	[25,o]=D21	
State 26	Action	Go To
X'::= p X ·	[26,";"]=R14	

4

The lexical checks for this language are related to the set of characters that are used as identifiers of variables, they must be characters up to a maximum of 8. The following expression represents the names of valid variables:

$$\text{name\_variable} = \bigcup_{i=1}^{8} [a - zA - Z]^i$$

Semantic checks for this language should take into account:

- The scope of declaration of the variables,
- The type of data.

For the control of types, two situations can happen:

- There is only one operand in the expression so that the type of the expression (variable or constant) must match the type of the variable on which the value is assigned.
- There is more than one operator, therefore, there must be at least one operator. In this case, it is a necessary condition (but not sufficient) for a logical expression to be correct that all its operators are logical. The same reasoning applies to the case of arithmetic expressions. It is the operators who determine the type of the expression. The assignment on a variable must take into account the type agreement.

For this domain, the semantic control is done through tables of symbols that collect this attribute. There are two possibilities, create a table of symbols for each scope or label the fields and incorporate that information into a single table of symbols. Whenever it is necessary to evaluate an expression (loop or assignment) it should be verified that the variables involved have been declared in the field in which they are used.

5

(1) **S**::= **A** ; **S'**
(2) **S'**::= **S**
(3) **S'**::= λ
(4) **A**::= **D**
(5) **A**::= **B**
(6) **A**::= **E**
(7) **D**::= type **V**
(8) **V**::= var **V'**
(9) **V'**::= , **V**
(10) **V'**::= λ
(11) **B**::= do **S** endo **X**
(12) **E**::= var = **X**
(13) **X**::= o **X'**
(14) **X'**::= p **X**
(15) **X'**::= λ

The semantic routines to control the scope and type must go in the productions where the expressions are evaluated and the variables declared. In the declaration (7) the lexeme attribute of the token "type" contains the type of the variables that are to be declared and that must be added to the Table of symbols. In this solution a Table of symbols will be used, labeling the scope of the variables. Each time the "do" token is moved, a new scope label must be generated that will be applied to the variables declared in the block. When it is reduced (11) then all the variables in the loop are removed from the Symbol Table.

B::= do {generate new label} **S** endo **X** {liberate variables}
D::= type **V** {}