

**UNITS 7 AND 8: SEMANTIC ANALYSIS and ERROR HANDLING**

We want to develop an analyzer that can verify that the data packages that circulate through an information channel have the appropriate structure.

A data package consists of two types of blocks, the blocks labeled with the letter "a" and those labeled with the letter "b". Valid data packages have the following structure:

- They start with a block "a"
- They can have any number of blocks greater than or equal to 1.
- They cannot have more than two consecutive blocks with the same label.

Example:

a abbabaab aabaa abaaaab ababaabb baab ...

Error package (three consecutive "a" blocks)

Error package (it starts with a "b" block)

1. Define the grammar G of the analyzer.
2. Construct an LL(1) syntax parsing table for the analyzer.
3. Construct an SLR(1) parsing table for the analyzer.
4. Show the derivation trees that are induced in the LL(1) and SLR(1) analysis for the blocks: "aabaa" and "abaaaab"
5. Perform syntax error recovery for the SLR(1) parser.

1.-

The problem of generating the grammar that will be used to perform the parsing can be approached from two points of view:

- Make a grammar that accepts only the valid words of the language. It has the advantage of not requiring any additional control, but the drawback of making recovery of the error more difficult and giving information about it.
- Construct a more general grammar that accepts valid and erroneous sentences, later by means of actions in the production rules, the syntactic analysis must be performed. It has the advantage of facilitating the recovery of errors and allows giving more information about the reason for the error.

Both approaches are valid, in this solution the second one will be developed.

A general grammar, the symbol E is the axiom, which accepts valid information packets (also accepts erroneous) is:

$$E ::= S d E \mid S$$

$$S ::= a S \mid b S \mid a \mid b$$

Factoring:

$$E ::= S E'$$

$$E' ::= d, E \mid \lambda$$

$$S ::= a R \mid b T$$

$$R ::= S \mid \lambda$$

$$T ::= S \mid \lambda$$

The production rules for R and T are the same and can be simplified:

$$E ::= S E'$$

$$E' ::= d E \mid \lambda$$

$$S ::= a R \mid b R$$

$$R ::= S \mid \lambda$$

The inclusion of the token "d," allows to separate the packages, it is the token that represents the blank spaces.

2.-

$\Sigma_N$	First			Follow	
E	a	b		\$	
E'	d	$\lambda$		\$	
R	a	b	$\lambda$	d	\$
S	a	b		d	\$

Table LL(1)		$\Sigma_T$			
		\$	a	b	d
$\Sigma_N$	E	E ::= S E'		E ::= S E'	
	E'	E' ::= $\lambda$		E' ::= d E	
	R	R ::= $\lambda$	R ::= S	R ::= S	R ::= $\lambda$
	S	S ::= a R		S ::= b R	



3.-

1.  $E' ::= E$
2.  $E' ::= \lambda$
3.  $E' ::= d E$
4.  $E ::= S E'$
5.  $R ::= \lambda$
6.  $R ::= S$
7.  $S ::= a R$
8.  $S ::= b R$

State 0	Action	Go To
$E' ::= \cdot E$		[0,E]=1
$E ::= \cdot S E'$		[0,S]=2
$S ::= \cdot a R$	[0,a]=D3	
$S ::= \cdot b R$	[0,b]=D4	
State 1	Action	Go To
$E' ::= E \cdot$	[1,\$]=ACP	
State 2	Action	Go To
$E ::= S \cdot E'$		[2,E']=5
$E' ::= \lambda$	[2,\$]=R2	
$E' ::= \cdot d E$	[2,d]=D6	
State 3	Action	Go To
$S ::= a \cdot R$		[3,R]=7
$R ::= \cdot \lambda$	[3,d]=R5	
	[3,\$]=R5	
$R ::= \cdot S$		[3,S]=8
$S ::= \cdot a R$	[3,a]=D3	
$S ::= \cdot b R$	[3,b]=D4	
State 4	Action	Go To
$S ::= b \cdot R$		[4,R]=9
$R ::= \cdot \lambda$	[4,d]=R5	
	[4,\$]=R5	
$R ::= \cdot S$		[4,S]=8
$S ::= \cdot a R$	[4,a]=D3	

$S ::= \cdot b R$	[4,b]=D4	
State 5	Action	Go To
$E ::= S E' \cdot$	[5,\$]=R4	
State 6	Action	Go To
$E' ::= d \cdot E$		[6,E]=10
$E ::= \cdot S E'$		[6,S]=2
$S ::= \cdot a R$	[6,a]=D3	
$S ::= \cdot b R$	[6,b]=D4	
State 7	Action	Go To
$S ::= a R \cdot$	[7,d]=R7	
	[7,\$]=R7	
State 8	Action	Go To
$R ::= S \cdot$	[8,d]=R6	
	[8,\$]=R6	
State 9	Action	Go To
$S ::= b R \cdot$	[9,d]=R8	
	[9,\$]=R8	
State 10	Action	Go To
$E' ::= d E \cdot$	[10,\$]=R3	

Table SLR(1)		Shift				Go To			
		\$	a	b	d	E'	E	R	S
States	0		D3	D4			1		2
	1	ACP							
	2	R2			D6	5			
	3	R5	D3	D4	R5			7	8
	4	R5	D3	D4	R5			9	8
	5	R4							
	6		D3	D4			10		2
	7	R7			R7				
	8	R6			R6				
	9	R8			R8				
	10	R3							

4.-

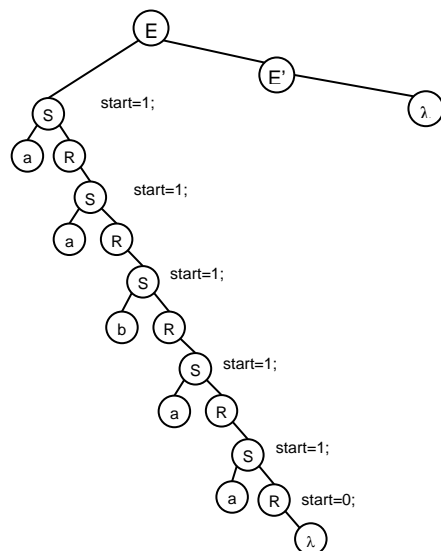
5.-

Sections 4 and 5 are going to be done together, the error recovery routines will be shown and then the derivation tree will be drawn for the blocks "aaba" and "abaaab", the latter is not a valid input for having three consecutive "a"

In order to incorporate the recovery (and detection) of errors into the downward analysis LL(1), the following data structures and actions associated with the productions are going to be used:

- One counter for "a" and another for blocks "b", "account\_a" and "account\_b" initialized to zero.
- A flag indicating the start of the analysis, "start", initialized to zero
- $S ::= a R$  {
  - start = 1;
  - account\_b = 0;
  - account\_a = account\_a + 1;
  - if account\_a > 2 then
    - ERROR("more than two consecutive blocks")
  - if\_end
- $S ::= b R$  {
  - if start=0 then
    - ERROR("start with block b")
  - if\_end
  - start = 1;
  - account\_a = 0;
  - account\_b = account\_b + 1;
  - if account\_b > 2 then
    - ERROR("more than two consecutive b blocks")
  - if\_end
- $R ::= \lambda$  { (\* end of word \*)
  - start = 0;
  - account\_a = 0;
  - account\_b = 0;

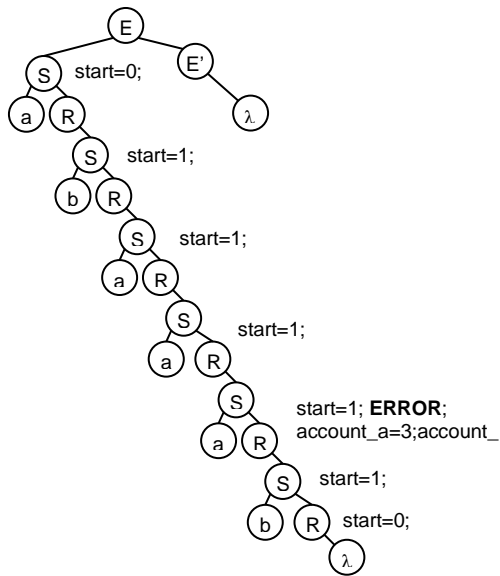
When an error occurs, as the control is done with the Action and not with the production rule, the token is read in the input and the analysis continues.



The analysis tree is:

The value that the variables take after applying the corresponding production rule has been represented. It can be observed that there has been no call to the error routine.





For the second word, when applying for the fifth time the production rule  $S ::= aR$ , the ERROR routine is called, the analysis continues without interruption.

For the SLR(1) analysis:

It can be observed in the SLR(1) analysis table that when rule 7 and 8 are reduced is

because the corresponding token "a" or "b" has been read in the input, this will be the moment to count the number of blocks "a" and "b". In addition, a variable "last" will be used to indicate which type of block "a" or "b" has been the last reduced.

The input  $[7, \$]=R7$  and  $[7, d]=R7$  is associated to the following action:

```
{
if last = "b" then
    account_a = 0;
if_end
last = "a";
account_a = account_a + 1;
if account_a > 2 then
    ERROR("more than two consecutive blocks")
if_end
}
```

The input  $[9, \$]=R8$  and  $[9, d]=R8$  is associated to the following action:

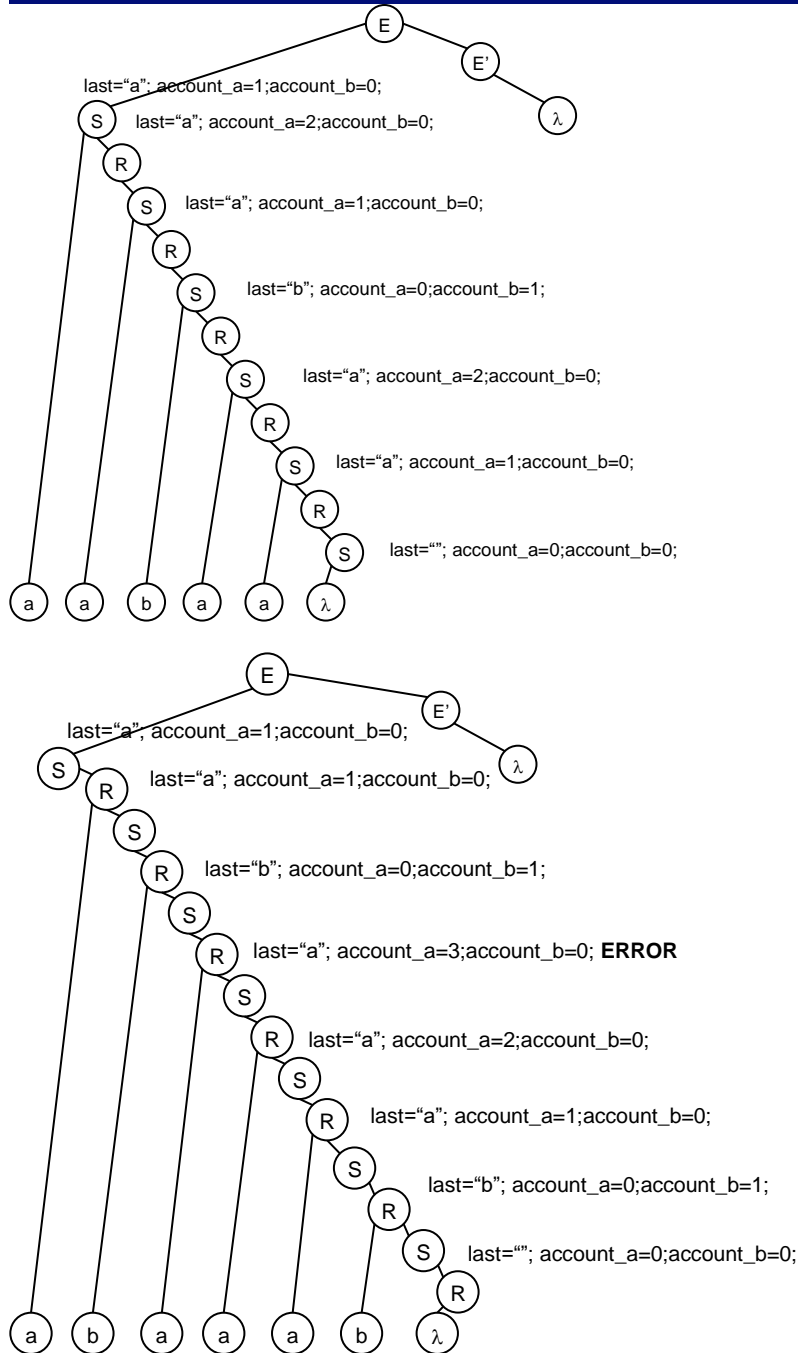
```
{
if last = "a" then
    account_b = 0;
if_end
last = "b";
account_b = account_b + 1;
if account_b > 2 then
    ERROR("more than two b consecutive blocks")
if_end
}
```

Finally in  $[1, \$]=ACP$  we have to control if the block started with "a" or "b" using the following rule:

```
{
if last = "b" then
    ERROR("Start with block b")
if_end
}
```

The rule  $R ::= \lambda$  can be used to initialize the variables.

The analysis trees are:



If the proposed grammar only accepts the specified language, then the actions will have to complete the "gaps" of the analysis table, along with the actions to read the following token.