Miguel Estevez

2017-0200

4.5 a) Show the actions of an LL(1) parser that uses Table 4.4 (page 163) to recognize the following arithmetic expressions:

a-) $3 + 4 * 5 - 6$

Action	Entrada	Stack
	$3 + 4 * 5 - 6\$$	$exp
$exp \rightarrow term\ exp^{'}$	$3 + 4 * 5 - 6\$$	$\$exp^{'}\ term$
$term \rightarrow factor\ term^{'}$	$3 + 4 * 5 - 6\$$	$\$exp^{'}\ term^{'}\ factor$
$factor \rightarrow num$	$3 + 4 * 5 - 6\$$	$\$exp^{'}\ term^{'}\ num$
$expend\ terminal$	$+4 * 5 - 6\$$	$\$exp^{'}\ term^{'}$

Action	Entrada	Stack
$term^{'} \rightarrow \epsilon$	$+4 * 5 - 6\$$	$\$exp^{'}$
$exp \rightarrow opsuma\ term\ \exp^{'}$	$+4 * 5 - 6\$$	$\$exp^{'} term\ opsuma$
$opsuma \rightarrow +$	$+4 * 5 - 6\$$	$\$exp^{'} term +$
expend terminal	$4 * 5 - 6\$$	$\$exp^{'} term$
$term \rightarrow factor\ term^{'}$	$4 * 5 - 6\$$	$\$exp^{'} term^{'}\ factor$
$factor \rightarrow num$	$4 * 5 - 6\$$	$\$exp^{'} term^{'}\ num$
expend terminal	$*5 - 6\$$	$\$exp^{'} term^{'}$
$term^{'} \rightarrow opmult\ factor\ term^{'}$	$*5 - 6\$$	$\$exp^{'} term^{'} factor\ opmult$
$opmult \rightarrow *$	$*5 - 6\$$	$\$exp^{'} term^{'} factor *$
expend terminal	$5 - 6\$$	$\$exp^{'} term^{'} factor$
$factor \rightarrow num$	$5 - 6\$$	$\$exp^{'} term^{'} num$
expend terminal	$-6\$$	$\$exp^{'} term^{'}$
$term^{'} \rightarrow \epsilon$	$-6\$$	$\$exp^{'}$
$exp^{'} \rightarrow opsuma\ term\ exp^{'}$	$-6\$$	$\$exp^{'} term\ opsuma$
$opsuma \rightarrow -$	$-6\$$	$\$exp^{'} term -$
expend terminal	$\$$	$\$exp^{'} term$
$term \rightarrow factor\ term^{'}$	$\$$	$\$exp^{'} term^{'} factor$
$factor \rightarrow num$	$\$$	$\$exp^{'} term^{'} num$
expend terminal	$\$$	$\$exp^{'} term'$
$term^{'} \rightarrow \epsilon$	$\$$	$\$exp^{'}$
$exp^{'} \rightarrow \epsilon$	$\$$	$\$$

b-)$3 * (4 - 5 + 6)$

Action	Entrada	Stack
	$3 * (4 - 5 + 6)\$$	$\$exp$
$exp \rightarrow term\ exp^{'}$	$3 * (4 - 5 + 6)$	$\$exp^{'}\ term$
$term \rightarrow factor\ term^{'}$	$3 * (4 - 5 + 6)$	$\$exp^{'}\ term^{'}\ factor$
$factor \rightarrow num$	$3 * (4 - 5 + 6)$	$\$exp^{'}\ term^{'}\ num$
Expend Terminal	$3 * (4 - 5 + 6)$	$\$exp^{'}\ term^{'}$
$term^{'} \rightarrow opmult\ factor\ term^{'}$	$*(4 - 5 + 6)$	$\$exp^{'}\ term^{'}\ factor\ opmult$
$opmult \rightarrow *$	$*(4 - 5 + 6)$	$\$exp^{'}\ term^{'}\ factor\ *$
Expend Terminal	$*(4 - 5 + 6)$	$\$exp^{'}\ term^{'}\ factor$
$factor \rightarrow (exp)$	$(4 - 5 + 6)\$$	$\$exp^{'}\ term^{'})exp($
Expend Terminal	$4 - 5 + 6)\$$	$\$exp^{'}\ term^{'})exp$
$exp \rightarrow term\ exp^{'}$	$4 - 5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}\ term$
$term \rightarrow factor\ term^{'}$	$4 - 5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}\ factor$
$factor \rightarrow num$	$4 - 5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}\ num$
Expend Terminal	$-5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}$
$term^{'} \rightarrow \epsilon$	$-5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}$
$exp^{'} \rightarrow opsuma\ term\ exp^{'}$	$-5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}\ term\ opsuma$
$opsuma \rightarrow -$	$-5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}\ term\ +$
Expend Terminal	$5 + 6)\$$	$\$exp^{'}\ term^{'})exp^{'}\ term$
$term \rightarrow factor\ term^{'}$	5+6)$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}\ factor$
$factor \rightarrow num$	5+6)$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}\ num$
Expend Terminal	+6)$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}$
$term^{'} \rightarrow \epsilon$	+6)$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}$

Action	Entrada	Stack
$exp^{'} \rightarrow opsuma\ term\ exp^{'}$	+6)$	$\$exp^{'}\ term^{'})exp^{'}$
$opsuma \rightarrow +$	+6)$	$\$exp^{'}\ term^{'})exp^{'}\ term\ opsuma$
Expend Terminal	6)$	$\$exp^{'}\ term^{'})exp^{'}\ term+$
$term \rightarrow factor\ term^{'}$	6)$	$\$exp^{'}\ term^{'})exp^{'}\ term$
$factor \rightarrow num$	6)$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}\ factor$
Expend Terminal	)$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}\ num$
$term^{'} \rightarrow \epsilon$	)$	$\$exp^{'}\ term^{'})exp^{'}\ term^{'}$
$exp^{'} \rightarrow \epsilon$	)$	$\$exp^{'}\ term^{'})exp^{'}$
Expend Terminal	$	$\$exp^{'}\ term^{'})$
$term^{'} \rightarrow \epsilon$	$	$\$exp^{'}$
$exp^{'} \rightarrow \epsilon$	$	$

c-) $3-(4+5*6)

Action	Entrada	Stack
	3-(4+5*6)$	$exp
$exp \rightarrow term\ exp^{'}$	3-(4+5*6)$	$\$exp^{'}\ term$
$term \rightarrow factor\ term^{'}$	3-(4+5*6)$	$\$exp^{'}\ term^{'}\ factor$
$factor \rightarrow num$	3-(4+5*6)$	$\$exp^{'}\ term^{'}\ num$
Expend Terminal	-(4+5*6)$	$\$exp^{'}\ term^{'}$
$term^{'} \rightarrow \epsilon$	-(4+5*6)$	$\$exp^{'}$
$exp^{'} \rightarrow opsuma\ term\ exp^{'}$	-(4+5*6)$	$\$exp^{'}\ term\ opsuma$
$opsuma \rightarrow -$	-(4+5*6)$	$\$exp^{'}\ term\ +$
Expend Terminal	(4+5*6)$	$\$exp^{'}\ term$
$term \rightarrow factor\ term^{'}$	(4+5*6)$	$\$exp^{'}\ term^{'}\ factor$
$factor \rightarrow (exp)$	(4+5*6)$	$\$exp^{'}\ term^{'})exp($
Expend Terminal	4+5*6)$	$\$exp^{'}\ term^{'})exp$

Action	Entrada	Stack
$exp \rightarrow term\ exp'$	4+5*6)$	$\$exp'\ term'\ )exp'\ term$
$term \rightarrow factor\ term'$	4+5*6)$	$\$exp'\ term'\ )exp'\ term'\ factor$
$factor \rightarrow num$	4+5*6)$	$\$exp'\ term'\ )exp'\ term'\ num$
Expend Terminal	+5*6)$	$\$exp'\ term'\ )exp'\ term'$
$term' \rightarrow \epsilon$	+5*6)$	$\$exp'\ term'\ )exp'$
$exp' \rightarrow opsuma\ term\ exp'$	+5*6)$	$\$exp'\ term'\ )exp'\ term\ opsuma$
$opsuma \rightarrow +$	+5*6)$	$\$exp'\ term'\ )exp'\ term+$
Expend Terminal	5*6)$	$\$exp'\ term'\ )exp'\ term$
$term \rightarrow factor\ term'$	5*6)$	$\$exp'\ term'\ )exp'\ term'\ factor$
$factor \rightarrow num$	5*6)$	$\$exp'\ term'\ )exp'\ term'\ num$
Expend Terminal	*6)$	$\$exp'\ term'\ )exp'\ term'$
$term' \rightarrow opmult\ factor\ term'$	*6)$	$\$exp'\ term'\ )exp'\ term'\ factor\ opmult$
$opmult \rightarrow *$	*6)$	$\$exp'\ term'\ )exp'\ term'\ factor*$
Expend Terminal	6)$	$\$exp'\ term'\ )exp'\ term'\ factor$
$factor \rightarrow num$	6)$	$\$exp'\ term'\ )exp'\ term'\ num$
Expend Terminal	)$	$\$exp'\ term'\ )exp'\ term'$
$term' \rightarrow \epsilon$	)$	$\$exp'\ term'\ )exp'$
$exp' \rightarrow \epsilon$	)$	$\$exp'\ term'\ )$
Expend terminal	$	$\$exp'\ term'$
$term' \rightarrow \epsilon$	$	$\$exp'$
$exp' \rightarrow \epsilon$	$	$

4.6 Show the actions of an LL(1) parser that uses table Section 4.2.2, page 155, to recognize the following arithmetic expressions:

a-) (())()

Action	**Entrada**	**Stack**
	(())()$	$S
$S \rightarrow (S)S$	(())()$	$\$S)S($
Expend terminal	))()$	$\$S)S$
$S \rightarrow (S)S$	))()$	$\$S)S)S($
Expend terminal	))()$	$\$S)S)S$
$S \rightarrow \epsilon$	))()$	$\$S)S)$
Expend Terminal	)()$	$\$S)S$
$S \rightarrow \epsilon$	)()$	$\$S)$
Expend terminal	()$	$\$S$
$S \rightarrow (S)S$	()$	$\$S)S($
Expend Terminal	)$	$\$S)S$
$S \rightarrow \epsilon$	)$	$\$S)$
Expend Terminal	$	$\$S$
$S \rightarrow \epsilon$	$	$

b-) (()())

Action	Entrada	Stack
	(()())$	$S
$S \rightarrow (S)S$	(()())$	$\$S)S($
Expend terminal	()())$	$\$S)S$
$S \rightarrow (S)S$	()())$	$\$S)S)S($
Expend Terminal	())$	$\$S)S)S$
$S \rightarrow \epsilon$	())$	$\$S)S)$
Expend Terminal	())$	$\$S)S$
$S \rightarrow (S)S$	())$	$\$S)S)S($
Expend Terminal	))$	$\$S)S)S$
$S \rightarrow \epsilon$	))$	$\$S)S)$
Expend Terminal	)$	$S)S
$S \rightarrow \epsilon$	)$	$S)
Expend Terminal	$	$S
$S \rightarrow \epsilon$	$	$

c-) $()(())$

Action	Entrada	Stack
	)(())$	$S
$S \rightarrow (S)S$	)(())$	$\$S)S($
Expend terminal	)(())$	$\$S)S$
$S \rightarrow \epsilon$	)(())$	$\$S)$
Expend terminal	(())$	$\$S$
$S \rightarrow (S)S$	(())$	$\$S)S($
Expend terminal	())$	$\$S)S$
$S \rightarrow (S)S$	())$	$\$S)S)S($
Expend Terminal	))$	$\$S)S)S$
$S \rightarrow \epsilon$	))$	$\$S)S)$
Expend Terminal	)$	$\$S)S$
$S \rightarrow \epsilon$	)$	$\$S)$
Expend Terminal	$	$\$S$
$S \rightarrow \epsilon$	$	$

4.11 An LL(1) parsing table such as that of Table 4.4 (page 163) generally has many blank entries representing errors. In many cases, all the blank entries in a row can be replaced by a single default entry, thus decreasing the size of the table considerably. Potential default entries occur in nonterminal rows when a nonterminal has a single production choice or when it has an production. Apply these ideas to Table 4.4. What are the drawbacks of such a scheme, if any?

M[N,T]	(	numero	)	+	-	*	$
exp	$exp \rightarrow term\ exp'$	$exp \rightarrow term\ exp'$	ERROR	ERROR	ERROR	ERROR	ERROR
exp'	ERROR	ERROR	$exp' \rightarrow \epsilon$	$exp' \rightarrow opsuma\ term\ exp'$	$exp' \rightarrow opsuma\ term\ exp'$	ERROR	$exp' \rightarrow \epsilon$
opsuma	ERROR	ERROR	ERROR	$opsuma \rightarrow +$	$opsuma \rightarrow -$	ERROR	ERROR
term	$term \rightarrow factor\ term'$	$term \rightarrow factor\ term'$	ERROR	ERROR	ERROR	ERROR	ERROR
$term'$	ERROR	ERROR	$term' \rightarrow \epsilon$	$term' \rightarrow \epsilon$	$term' \rightarrow \epsilon$	$term' \rightarrow opmult\ factor\ term'$	$term' \rightarrow \epsilon$
opmult	ERROR	ERROR	ERROR	ERROR	ERROR	$opmult \rightarrow *$	ERROR
factor	$factor \rightarrow (exp)$	$factor \rightarrow numero$	ERROR	ERROR	ERROR	ERROR	ERROR

Dado la tabla de transición, una desventaja clara que podría tener es que el compilador no contemplaria alguna forma de distinguir entre un error u otro ya que desde que se encuentre el primer error este saldría.

4.12

a. Can a LL(1) grammar be ambiguous? Why or why not?

No es posible que una gramática LL(1) pueda ser ambigua ya que la tabla de transiciones para la gramática LL(1) obliga que exista una sola transición para cada celda de esta forzando a que la gramática no pueda tener ambigüedad.

b. Can an ambiguous grammar be LL(1)? Why or Why not?

No, por el hecho de que una celda en la tabla de transiciones no aceptara 2 posibles caminos para una misma celda.

c. Must an unambiguous grammar be LL(1)? Why or why not?

No necesariamente ya que existen algunas gramáticas las cuales salen del ambito LL(1) por su ambigüedad y requieren un look ahead mayor o un proceso de desambiguacion para tratar su ambigüedad.

4.13 Show that a left-recursive grammar cannot be LL(1).

Teniendo la siguiente gramática

$$S \rightarrow Sa|B$$
$$B \rightarrow \epsilon$$

Dado que las producciones de B son un subconjunto de las de Sa, esto contradice la regla de LL(1) la cual dice que si existen varias producciones para un mismo No terminal estas debe ser desjuntas $PROD(B) \cap PROD(SA) = \emptyset$ lo cual no se cumple en este caso. Que a su vez se puede traducir a tener mas de una posible producción en una celda de la tabla de transiciones.

4.21 Given the grammar $A \rightarrow aAa|\epsilon$

a. Show that this grammar is not LL(1).

Esta gramática no es LL(1) ya que con un solo look ahead no da basto para poder quitar la ambigüedad que hay en esta gramática. También cabe destacar que esta gramática no ha sido concebida para que pueda ser parseada por un LL(1), ya que la mirrilla no sirve de mucho para esta gramática. No te ayuda mucho para quitar la ambigüedad que presenta esta gramática, para

decir que si paro o sigo produciendo mas.

b. An attempt to write a recursive-descent parser for this grammar is represented by the following pseudocode.

```
procedure A ;
begin
    if token = a then
        getToken;
        A;
        if token = a then getToken ;
        else error;
    else if token <> $ then error;
end A ;
```

Show that this procedure will not work correctly.

Por ejemplo si se le pasa un string $aaaa$ que deberia ser aceptado segun la gramatica. Esas cuatros a seran consumidas entre las primeras 3 instrucciones del parser:

if token = a then
    getToken;
    A;

Entonces cuando sea llamada por una 5ta vez la funcion ya no habra mas a que consumir y pasara por la funcion sin hacer nada. Luego la 4ta llamada tendra que hacer el $if\ token\ = a$ lo cual no habra ninguna a y esta retornara error. Lo que pasa es que esta gramatica trata de representar que a cada *a* le corresponde otra *a* reciproca. Entonces lo que esta haciendo la funcion es consumiendo todas las *a* que hay sin dejar a las *a* reciprocas.

C. A backtracking recursive-descent parser for this language can be written but requires the use of an *unGetToken* procedure that takes a token as a parameter and returns that token to the front of the stream of input tokens. It also requires that the procedure. A be written as a Boolean function that returns success or failure, so that when A calls itself, it can test for success before consuming another token, and so that, if the A —> a A a choice fails, the code can go on to try the A —» e alternative. Rewrite the pseudocode of part (b) according to this recipe, and trace its operation on the string aaaa$.

```
procedure A ;
begin
    if token = a then
        getToken;
        if(A) return success;
        if token = a then
            getToken ;
            if token = $ then return success
        else unGetToken(a);
     return failure
end A ;
```