PRÁCTICA II

Saul Andersson D'Angelo Rojas Coila
Departamento de Ciencia de la Computación
UCSP

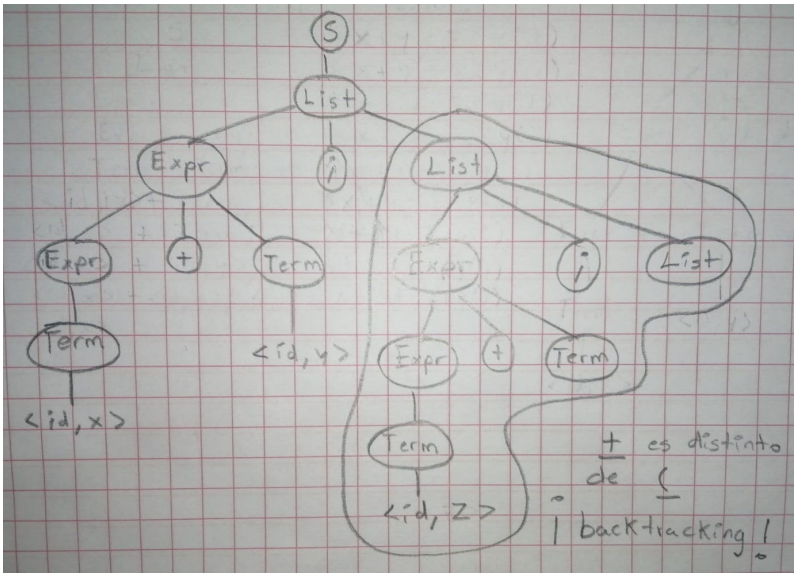
Resolución de los ejercicios de la práctica.

1. Resolución

   Ejercicio i): Dar una derivación más a la izquierda para la cadena “x + y ; z (y ( ))” y arma el árbol de parseo.

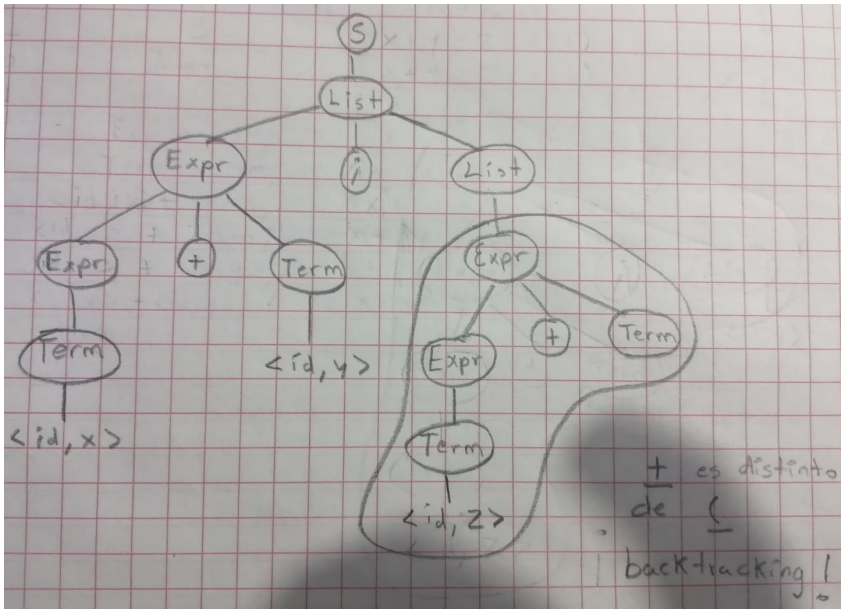
   Primera Tabla

Regla	Forma Sentencial	Entrada
-	S	↑x + y ; z (y ( ))
0	List	↑x + y ; z (y ( ))
1	Expr ; List	↑x + y ; z (y ( ))
3	Expr + Term ; List	↑x + y ; z (y ( ))
4	Term + Term ; List	↑x + y ; z (y ( ))
5	<id,x> + Term ; List	↑x + y ; z (y ( ))
→	<id,x> + Term ; List	x↑ + y ; z (y ( ))
→	<id,x> + Term ; List	x +↑ y ; z (y ( ))
5	<id,x> + <id,y> ; List	x +↑ y ; z (y ( ))
→	<id,x> + <id,y> ; List	x + y↑ ; z (y ( ))
→	<id,x> + <id,y> ; List	x + y ;↑ z (y ( ))
1	<id,x> + <id,y> ; Expr ; List	x + y ;↑ z (y ( ))
3	<id,x> + <id,y> ; Expr + Term ; List	x + y ;↑ z (y ( ))
4	<id,x> + <id,y> ; Term + Term ; List	x + y ;↑ z (y ( ))
5	<id,x> + <id,y> ; <id,x> + Term ; List	x + y ;↑ z (y ( ))
→	<id,x> + <id,y> ; <id,x> + Term ; List	x + y ; z↑ (y ( )) [backtrack!]



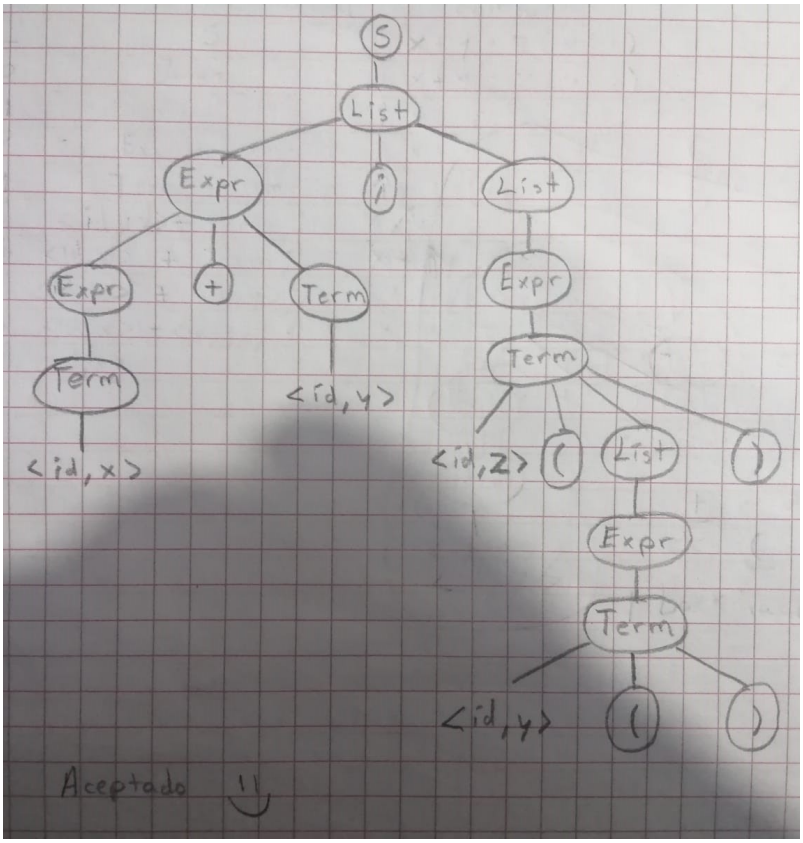
Segunda Tabla

Regla	Forma Sentencial	Entrada
-	S	↑x + y ; z (y ( ))
0	List	↑x + y ; z (y ( ))
1	Expr ; List	↑x + y ; z (y ( ))
3	Expr + Term ; List	↑x + y ; z (y ( ))
4	Term + Term ; List	↑x + y ; z (y ( ))
5	<id,x> + Term ; List	↑x + y ; z (y ( ))
→	<id,x> + Term ; List	x↑ + y ; z (y ( ))
→	<id,x> + Term ; List	x +↑ y ; z (y ( ))
5	<id,x> + <id,y> ; List	x +↑ y ; z (y ( ))
→	<id,x> + <id,y> ; List	x + y↑ ; z (y ( ))
→	<id,x> + <id,y> ; List	x + y ;↑ z (y ( ))
2	<id,x> + <id,y> ; Expr	x + y ;↑ z (y ( ))
3	<id,x> + <id,y> ; Expr + Term	x + y ;↑ z (y ( ))
4	<id,x> + <id,y> ; Term + Term	x + y ;↑ z (y ( ))
5	<id,x> + <id,y> ; <id,x> + Term	x + y ;↑ z (y ( ))
→	<id,x> + <id,y> ; <id,x> + Term ; List	x + y ; z↑ (y ( )) [backtrack!]



Tercera Tabla

Regla	Forma Sentencial	Entrada
-	S	↑x + y ; z (y ( ))
0	List	↑x + y ; z (y ( ))
1	Expr ; List	↑x + y ; z (y ( ))
3	Expr + Term ; List	↑x + y ; z (y ( ))
4	Term + Term ; List	↑x + y ; z (y ( ))
5	<id,x> + Term ; List	↑x + y ; z (y ( ))
→	<id,x> + Term ; List	x↑ + y ; z (y ( ))
→	<id,x> + Term ; List	x +↑ y ; z (y ( ))
5	<id,x> + <id,y> ; List	x +↑ y ; z (y ( ))
→	<id,x> + <id,y> ; List	x + y↑ ; z (y ( ))
→	<id,x> + <id,y> ; List	x + y ;↑ z (y ( ))
2	<id,x> + <id,y> ; Expr	x + y ;↑ z (y ( ))
4	<id,x> + <id,y> ; Term	x + y ;↑ z (y ( ))
Evito incluir la producción 5 y 6 de la gramática, porque ambas dan backtracking, y no quiero extender muchas tablas		
7	<id,x> + <id,y> ; <id,z> (List)	x + y ;↑ z (y ( ))
→	<id,x> + <id,y> ; <id,z> (List)	x + y ; z↑ (y ( ))
→	<id,x> + <id,y> ; <id,z> (List)	x + y ; z (↑ y ( ))
Evito incluir la producción 1 de la gramática, porque me dará un backtracking		
2	<id,x> + <id,y> ; <id,z> (Expr)	x + y ; z (↑ y ( ))
4	<id,x> + <id,y> ; <id,z> (Term)	x + y ; z (↑ y ( ))
Me voy de frente a la producción 6, sin pasar por la 5, para acabar el algoritmo		
6	<id,x> + <id,y> ; <id,z> (<id,y> ( ))	x + y ; z (↑ y ( ))
→	<id,x> + <id,y> ; <id,z> (<id,y> ( ))	x + y ; z (y↑ ( ))
→	<id,x> + <id,y> ; <id,z> (<id,y> ( ))	x + y ; z (y (↑))
→	<id,x> + <id,y> ; <id,z> (<id,y> ( ))	x + y ; z (y ( )↑)
→	<id,x> + <id,y> ; <id,z> (<id,y> ( ))	x + y ; z (y ( ))↑ [Fin, ¡aceptado!]



Ejercicio ii): Transforma la gramática para que se pueda utilizar para construir un parser predictivo topdown con un símbolo de lookahead.
Dada la gramática

S → List
List → Expr ; List
| Expr
Expr → Expr + Term
| Term
Term → id
| id ( )
| id (List)

Primero eliminamos la recursión a la izquierda: la única producción que posee la recursión a la izquierda es Expr → Expr + Term | Term, así que aplicamos el algoritmo y obtenemos la siguiente gramática.

S → List
List → Expr ; List
| Expr
Expr → Term $Expr_1$
$Expr_1$ → + Term $Expr_1$
| ε
Term → id
| id ( )
| id (List)

Segundo debemos realizar la factorización por la izquierda, para convertirla a LL(1).

S → List
List → Expr List'
List' → ; List
| ε
Expr → Term $Expr_1$
$Expr_1$ → + Term $Expr_1$
| ε
Term → id Term'
Term' → ( )
| (List)
| ε

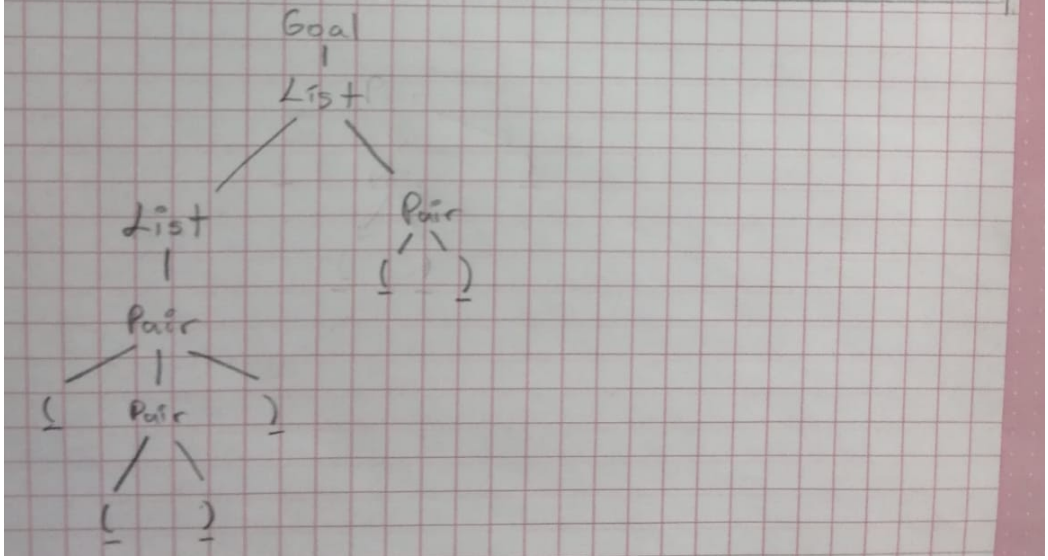
Aún la producción Term' contiene prefijos similares, así que continuamos factorizando.

S → List
List → Expr List'
List' → ; List
| ε
Expr → Term $Expr_1$
$Expr_1$ → + Term $Expr_1$
| ε
Term → id Term'
Term' → (G'
| ε
G' →)
| List)

Ahora la gramática es LL(1).

2. Resolución: (( )) ( )

Iteration	State	Word	Stack	Handle	Action
initial	–	(	$0	– none –	–
1	0	(	$0	– none –	shift 3
2	3	(	$0 (3	– none –	shift 6
3	6	)	$0 (3 (6	– none –	shift 10
4	10	)	$0 (3 (6) 10	( )	reduce 5
5	5	)	$0 (3 Pair 5	– none –	shift 8
6	8	(	$0 (3 Pair 5) 8	(Pair)	reduce 4
7	2	(	$0 Pair 2	Pair	reduce 3
8	1	(	$0 List 1	– none –	shift 3
9	1	)	$0 List 1 (3	– none –	shift 7
10	7	$	$0 List 1 (3) 7	( )	reduce 5
11	4	$	$0 List 1 Pair 4	List Pair	reduce 2
12	1	$	$0 List 1	– none –	accept



Nota: en la iteración 9, la variable state no tiene el valor 1, sino 3.