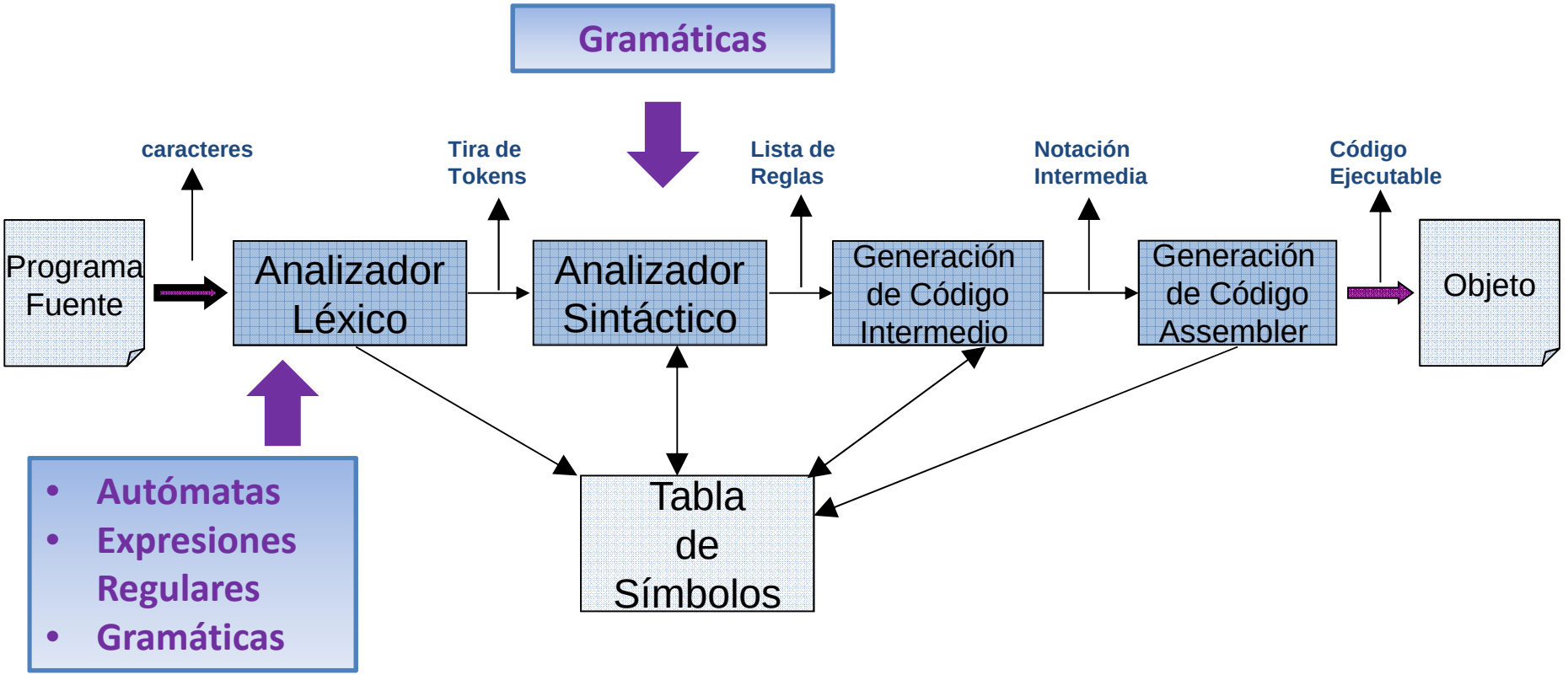
Lenguajes y Compiladores

Analizador Sintáctico

Parsing Ascendente

UNLaM

Etapas del Compilador



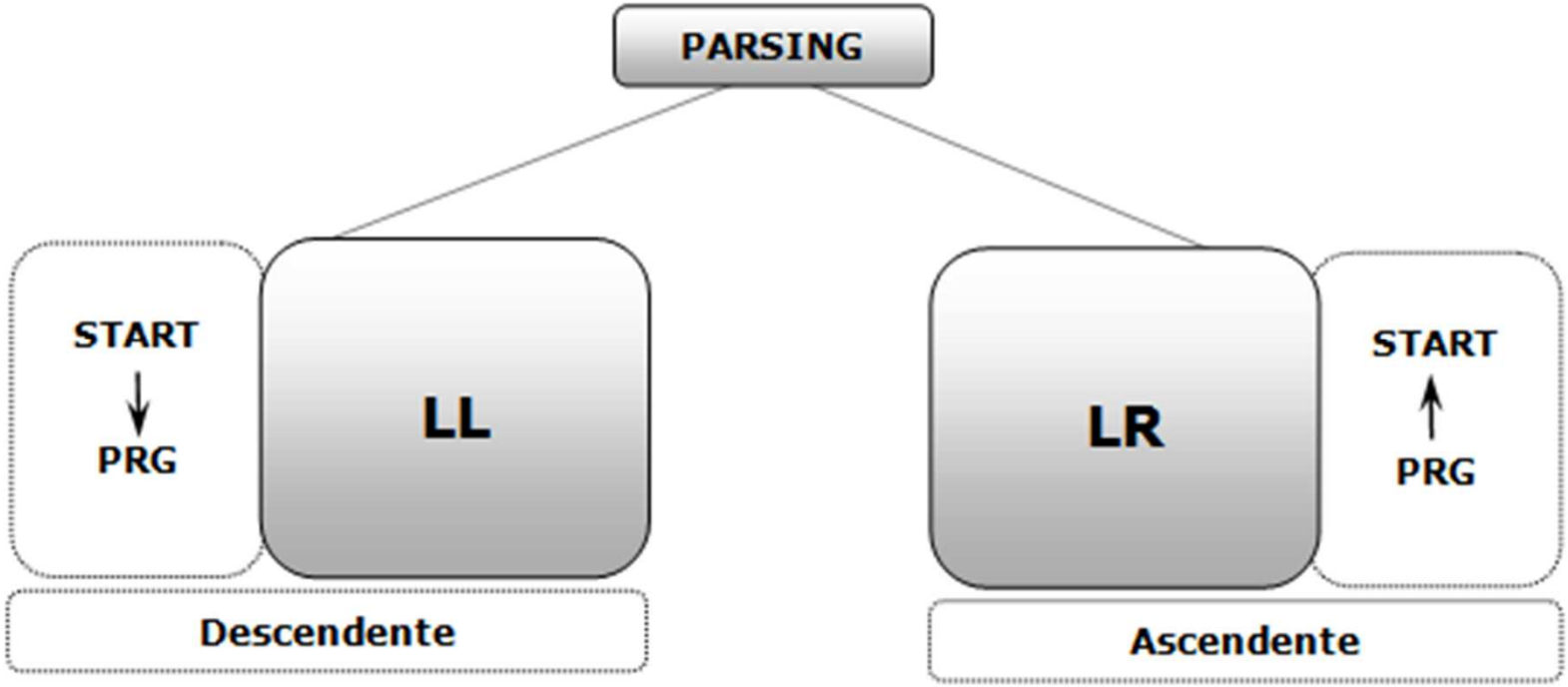
Análisis sintáctico

- Toma el programa a analizar y aplicando la gramática devuelve una lista de reglas en un orden determinado generando un Arbol de parsing

- Este árbol es abstracto, no se crea sino que se expresa a través de la lista de reglas aplicadas.

Análisis sintáctico

Parsing



Análisis sintáctico

Parsing ascendente

Análisis sintáctico

Parsing ascendente

- También denominado “Bottom Up”
- *Análisis Sintáctico* en el que se divide el programa de usuario hasta llegar al símbolo distinguido (start).
- Se dice que este proceso es LR
- “Left” porque lee el programa de usuario de izquierda a derecha.
- “Right” porque para aplicar las reglas busca asociar con el lado derecho de la definición de cada regla.

Análisis sintáctico

Parsing ascendente

SLR ó LR(0)

LR (1)

LALR (look ahead left to right)

START (HIPÓTESIS)

PRG

Análisis sintáctico

Parsing ascendente

- Es indiferente que sean recursivas a derecha o a izquierda.
- No debe ser una gramática ambigua
- Se utiliza desde el nacimiento del lenguaje C.

Análisis sintáctico

Implementación del método SLR

ESTADOS	TERMINALES	NO TERMINALES

Análisis sintáctico

Parsing ascendente

0. **<ASIG'> → <ASIG>**
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

- **La gramática se encuentra "aumentada"**
- **Se genera un nuevo símbolo distinguido**
- **Se establece una única regla con este no terminal**

Análisis sintáctico Parsing ascendente

GRADO DE AVANCE DE UNA REGLA EN EL RECONOCIMIENTO

- Será determinado por un punto (.) en algún lugar de la regla
- Lo que lo que está delante del punto se considera ya analizado
- Lo que está después del punto estará pendiente de análisis

Análisis sintáctico Parsing ascendente

GRADO DE AVANCE DE UNA REGLA EN EL RECONOCIMIENTO

<EXPRESION> → <EXPRESION> + <TERMINO>

Al comienzo, se debe analizar toda la regla:
<EXPRESION> → **.** <EXPRESION> + <TERMINO>

Si <EXPRESION> es reconocido:
<EXPRESION> → <EXPRESION> **.** + <TERMINO>

Cuando toda la regla fue analizada
<EXPRESION> → <EXPRESION> + <TERMINO> **.**

Análisis sintáctico Parsing ascendente

- La tabla se construye con un grupo de estados que forman un autómata finito
- El primer estado coloca el (.) delante del símbolo distinguido ya que se necesita analizar toda la gramática.
- Luego toda vez que un punto queda delante de un símbolo no terminal, se expanden las reglas encabezadas por el mismo.

Análisis sintáctico Parsing ascendente

ESTADO 0

<ASIG'> → . <ASIG>

<ASIG> →. id := <EXPRESION>

0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

Se debe definir cuáles elementos válidos son los que pueden llegar a aparecer a continuación del punto.

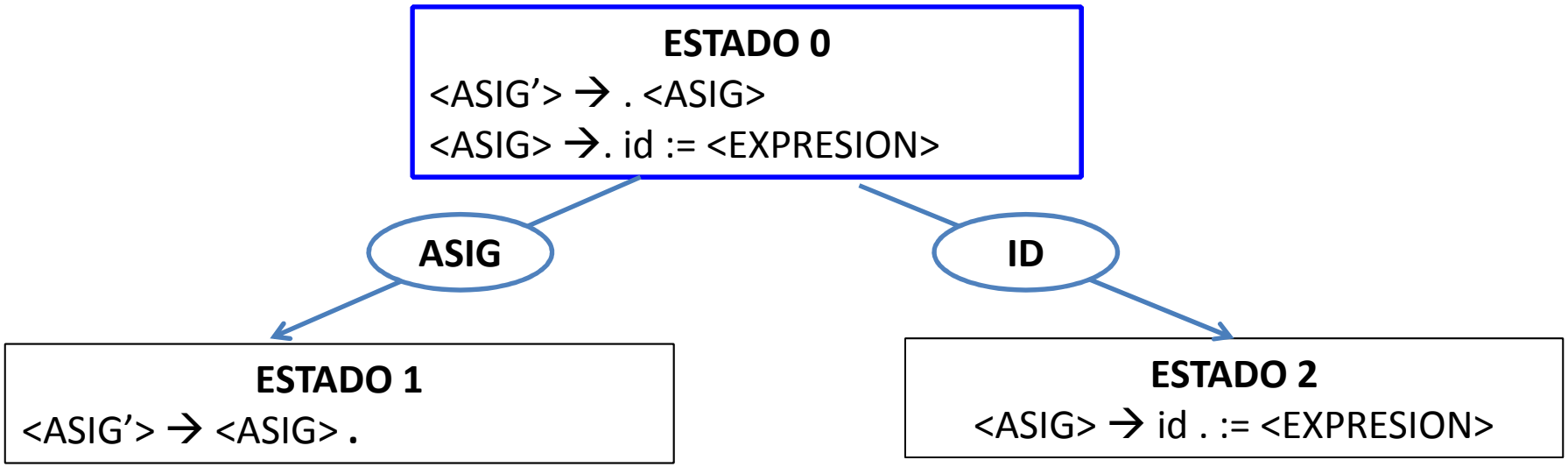
Como el punto está delante de un No Terminal, debemos aplicar una regla para resolverlo, a medida que hacemos esto generamos otro estado.

Análisis sintáctico Parsing ascendente

ESTADO 0
<ASIG'> → . <ASIG>
<ASIG> →. id := <EXPRESION>

0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

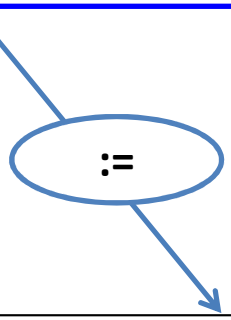
¿Que ocurre cuando estando en el estado 0 llega un <ASIG > o un *id*?



Análisis sintáctico Parsing ascendente

ESTADO 2

<ASIG> → id . := <EXPRESION>



ESTADO 3

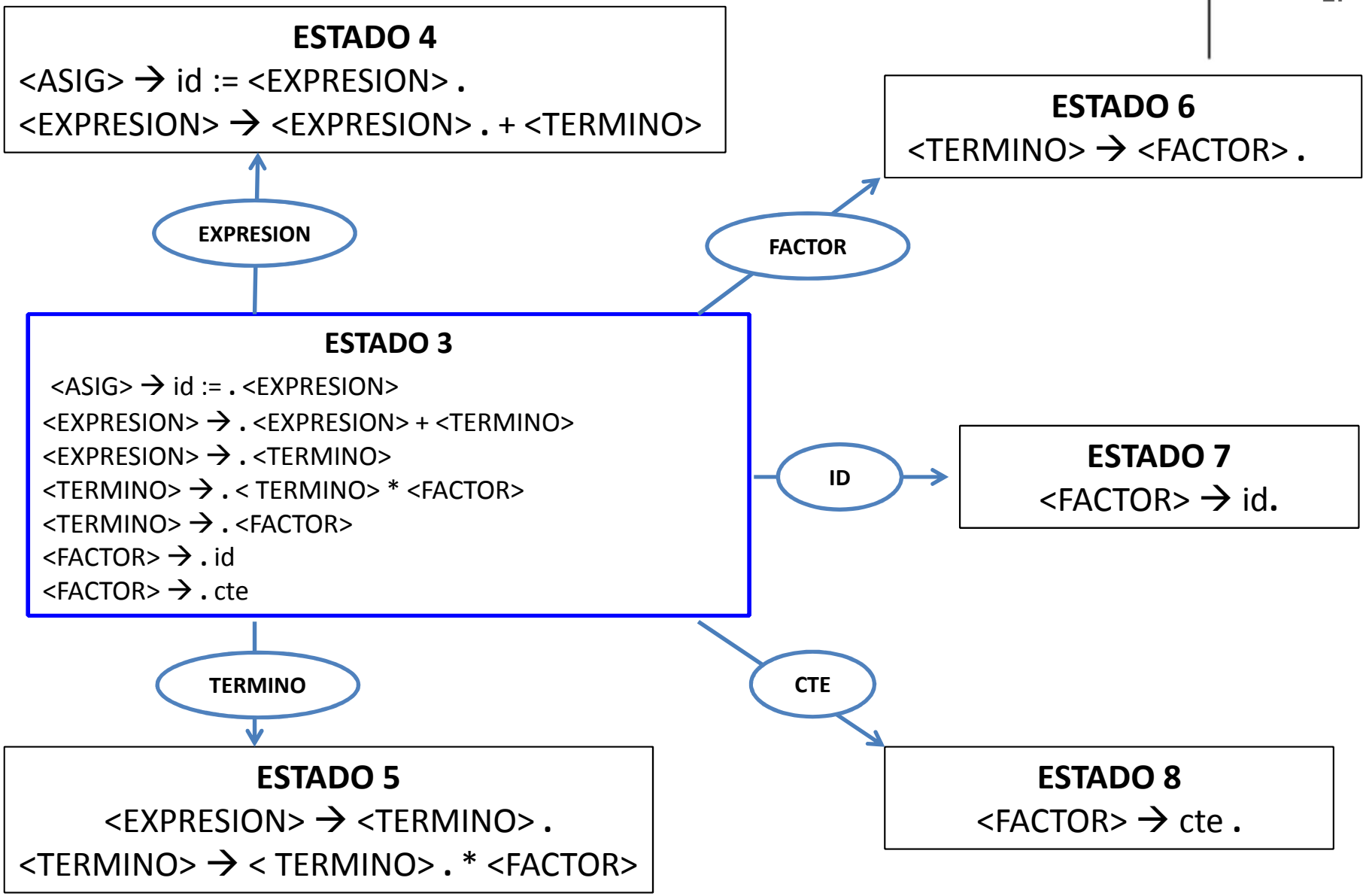
<ASIG> → id := **.** <EXPRESION>

<EXPRESION> → **.** <EXPRESION> + <TERMINO>
<EXPRESION> → **.** <TERMINO>
<TERMINO> → **.** < TERMINO> * <FACTOR>
<TERMINO> → **.** <FACTOR>
<FACTOR> → **.** id
<FACTOR> → **.** cte

0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

- **Aparece un No Terminal delante del punto, <EXPRESION>**
- **Se desarrollan las reglas que definen a este No Terminal**
- **Además se incluyen las reglas 4 y 5 ya que una <EXPRESION> es un <TERMINO>**
- **Ídem para las reglas 6 y 7**

Análisis sintáctico Parsing ascendente

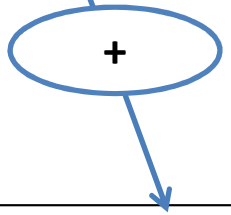


Análisis sintáctico Parsing ascendente

ESTADO 4

<ASIG> → id := <EXPRESION> .

<EXPRESION> → <EXPRESION> . + <TERMINO>



ESTADO 9

<EXPRESION> → <EXPRESION> + . <TERMINO>

<TERMINO> → . < TERMINO> * <FACTOR>

<TERMINO> → . <FACTOR>

<FACTOR> → . id

<FACTOR> → . cte

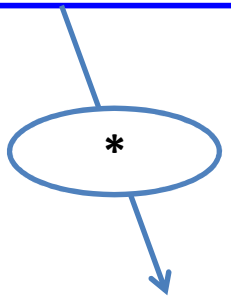
0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

Análisis sintáctico Parsing ascendente

0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

ESTADO 5

<EXPRESION> → <TERMINO> .
<TERMINO> → < TERMINO> . * <FACTOR>



ESTADO 10

<TERMINO> → < TERMINO> * . <FACTOR>
<FACTOR> → . id
<FACTOR> → . cte

ESTADO 6

<TERMINO> → <FACTOR> .

ESTADO 7

<FACTOR> → id.

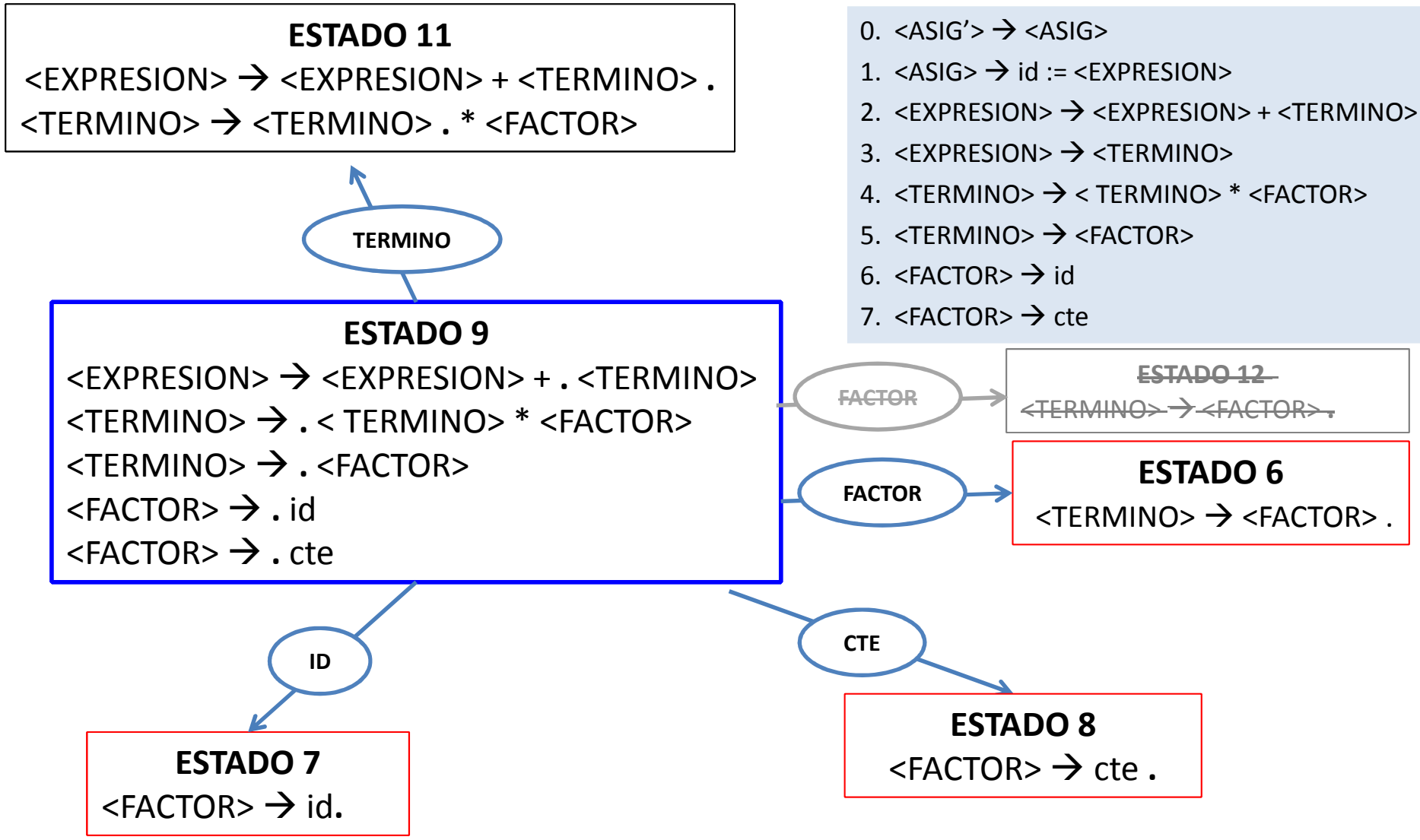
ESTADO 8

<FACTOR> → cte .

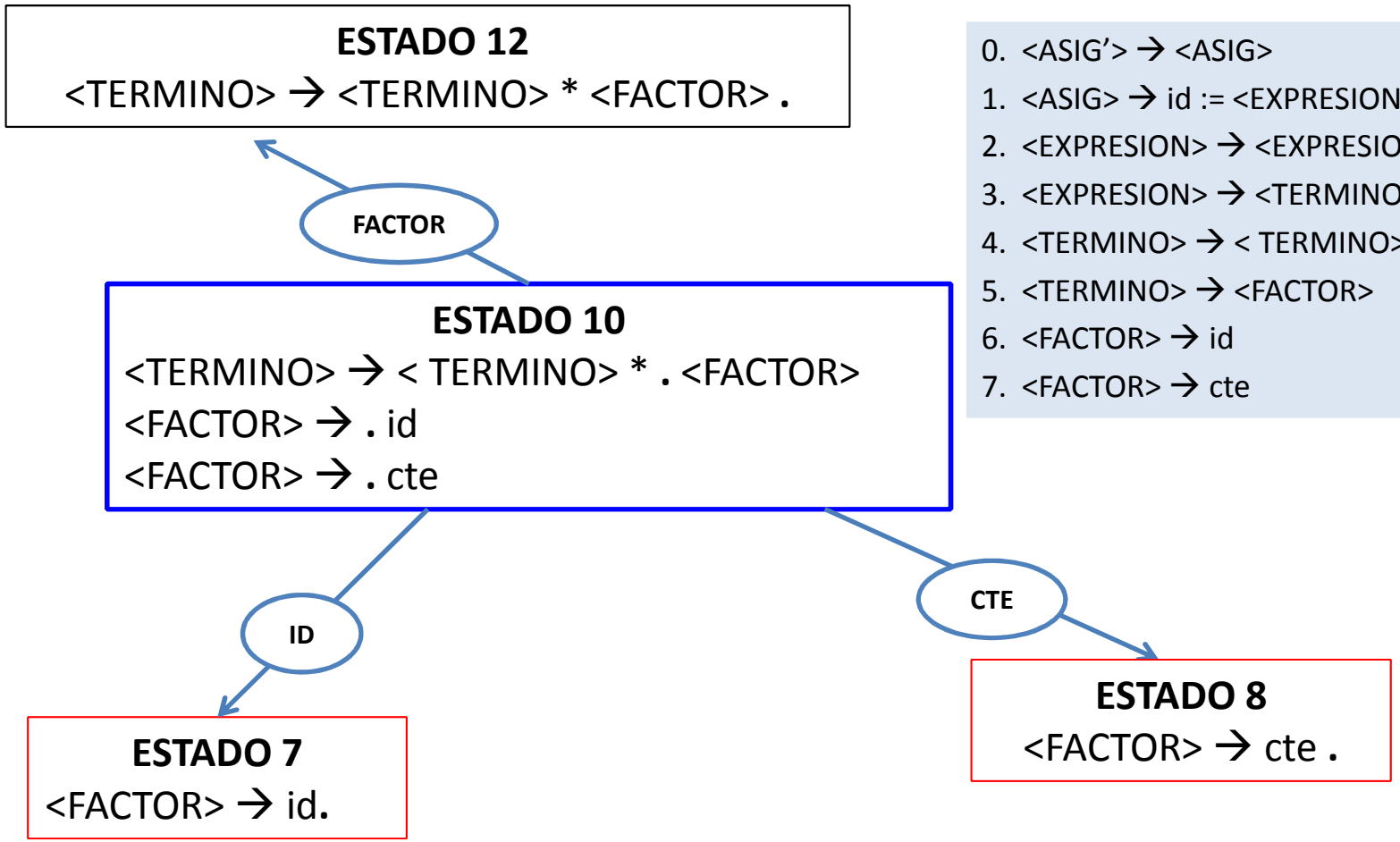
Notar: Estos estados no generan nuevos estados

Análisis sintáctico Parsing ascendente

0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte



Análisis sintáctico Parsing ascendente



0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

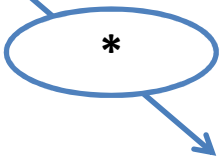
Análisis sintáctico Parsing ascendente

0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

ESTADO 11

<EXPRESION> → <EXPRESION> + <TERMINO> **.**

<TERMINO> → <TERMINO> **.** * <FACTOR>



ESTADO 10

<TERMINO> → < TERMINO> * **.** <FACTOR>

<FACTOR> → **.** id

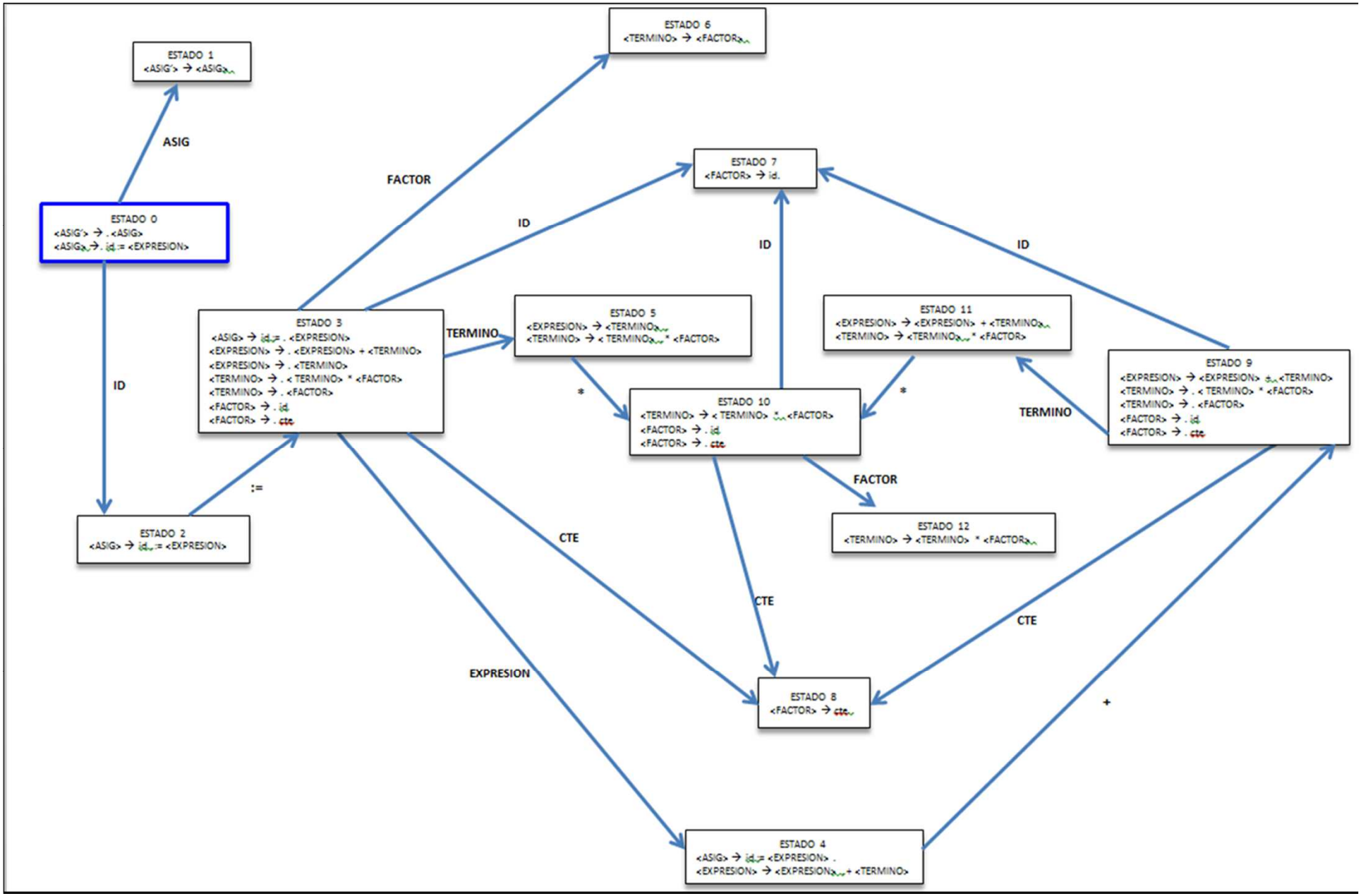
<FACTOR> → **.** cte

ESTADO 12

<TERMINO> → <TERMINO> * <FACTOR> **.**

No existen estados abiertos.
Se da por finalizado el algoritmo de generación de estados.

Análisis sintáctico Parsing ascendente



Análisis sintáctico Parsing ascendente

Para facilitar el armado de la tabla se reemplazará la notación BNF por notación de gramáticas

0. <ASIG'> → <ASIG>
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <EXPRESION> + <TERMINO>
3. <EXPRESION> → <TERMINO>
4. <TERMINO> → < TERMINO> * <FACTOR>
5. <TERMINO> → <FACTOR>
6. <FACTOR> → id
7. <FACTOR> → cte

0. A'→ A
1. A→ id := E
2. E→ E+ T
3. E→ T
4. T → T* F
5. T → F
6. F→ id
7. F→ cte

Análisis sintáctico Parsing ascendente

ESTADOS ACCIONES GOTO

	id	:=	+	*	cte	$	A	E	T	F
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										
10										
11										
12										

Análisis sintáctico Parsing ascendente

GOTO

- Se detectan aquellos estados en los que se llega cuando recibe un No Terminal

ESTADO 1 (0, A)
ESTADO 4 (3, E)
ESTADO 5 (3, T)
ESTADO 6 (3, F)
ESTADO 11 (9, T)
ESTADO 6 (9, F)
ESTADO 12 (10, F)

Análisis sintáctico Parsing ascendente

ESTADO 1 (0, ASIG)
ESTADO 4 (3, EXPRESION)
ESTADO 5 (3, TERMINO)
ESTADO 6 (3, FACTOR)
ESTADO 11 (9, TERMINO)
ESTADO 6 (9, FACTOR)
ESTADO 12 (10, FACTOR)

	id	:=	+	*	cte	$	A	E	T	F
0							1			
1										
2										
3								4	5	6
4										
5										
6										
7										
8										
9									11	6
10										12
11										
12										

Análisis sintáctico Parsing ascendente

ACCIONES

DESPLAZAR: se consume token siguiente que forma parte de la lectura de la entrada del usuario de izquierda a derecha.

⬇

(SHIFT o DESPLAZAR a NUMERO DE ESTADO)

REDUCIR: se reconoce el lado derecho de una regla y se define como el elemento No Terminal del lado izquierdo.

⬇

(SUBIR UN NIVEL EN EL ÁRBOL)

Análisis sintáctico Parsing ascendente

DESPLAZAR

- Se detectan aquellos estados en los que se llega cuando recibe un Terminal

ESTADO 2 (0, ID)
ESTADO 3 (2, :=)
ESTADO 7 (3, ID)
ESTADO 8 (3, CTE)
ESTADO 9 (4, +)
ESTADO 10 (5, *)
ESTADO 7 (9, ID)
ESTADO 8 (9, CTE)
ESTADO 7 (10, ID)
ESTADO 8 (10, CTE)
ESTADO 10 (11, *)

Análisis sintáctico Parsing ascendente

ESTADO 2 (0, id)
ESTADO 3 (2, :=)
ESTADO 7 (3, id)
ESTADO 8 (3, cte)
ESTADO 9 (4, +)
ESTADO 10 (5, *)
ESTADO 7 (9, id)
ESTADO 8 (9, cte)
ESTADO 7 (10, id)
ESTADO 8 (10, cte)
ESTADO 10 (11, *)

	id	:=	+	*	cte	$	A	E	T	F
0	D2						1			
1										
2		D3								
3	D7				D8			4	5	6
4			D9							
5				D10						
6										
7										
8										
9	D7				D8				11	6
10	D7				D8					12
11				D10						
12										

Análisis sintáctico Parsing ascendente

REDUCIR

Se obtienen aquellos estados que tienen al menos una regla con punto al final de todo, es decir, cuando han completado el avance.

ESTADO 1 (0, ASIG)	<A'> → <A> .
ESTADO 4 (3, EXPRESION)	<A> → id := <E> .
ESTADO 5 (3, TERMINO)	<E> → <T> .
ESTADO 6 (3, FACTOR)	<T> → <F> .
ESTADO 7 (3, id)	<F> → id .
ESTADO 8 (3, cte)	<F> → cte .
ESTADO 11 (9, TERMINO)	<E> → <E> + <T> .
ESTADO 6 (9, FACTOR)	<T> → <F> .
ESTADO 7 (9, id)	<F> → id .
ESTADO 8 (9, cte)	<F> → cte .
ESTADO 12 (10, FACTOR)	<T> → <T> * <F> .
ESTADO 7 (10, id)	<F> → id .
ESTADO 8 (10, cte)	<F> → cte .

Análisis sintáctico Parsing ascendente

0. A'→ A
1. A→ id := E
2. E→ E+ T
3. E→ T
4. T → T* F
5. T → F
6. F→ id
7. F→ cte

REDUCIR

Para completar la tabla, se deberá llenar cada casillero
[NUMERO ESTADO, Sgt(No terminal)] (1)

Sgt(A') = { $ }
Sgt(A) = Sgt(A') = { $ }
Sgt(E) = Sgt(A) U { + } = { $, + }
Sgt(T) = Sgt(E) U { * } = { $, +, * }
Sgt(F) = Sgt(T) = { $, +, * }

(1) Ver anexo primeros y siguientes

Análisis sintáctico Parsing ascendente

Sgt(A') = { $ }
Sgt(A) = { $ }
Sgt(E) = { $, + }
Sgt(T) = { $, +, * }
Sgt(F) = { $, +, * }

0. A'→ A
1. A→ id := E
2. E→ E+ T
3. E→ T
4. T → T* F
5. T → F
6. F→ id
7. F→ cte

	id	:=	+	*	cte	$	A	E	T	F
0	D2						1			
1						COMPILACION OK				
2		D3								
3	D7				D8			4	5	6
4			D9			R1 : A→ id := E				
5			R3 : E → T	D10		R3 : E → T				
6			R5 : T → F	R5 : T → F		R5 : T → F				
7			R6 : F → id	R6 : F→ id		R6 : F→ id				
8			R7 : F → cte	R7 : F→ cte		R7 : F→ cte				
9	D7				D8				11	6
10	D7				D8					12
11			R2 : E→ E+ T	D10		R2 : E → E+ T				
12			R4 : T →T* F	R4 : T → T* F		R4 : T → T* F				

Análisis sintáctico

0. A'→ A
1. A→ id := E
2. E→ E+ T
3. E→ T
4. T → T* F
5. T → F
6. F→ id
7. F→ cte

Parsing ascendente

PILA	PRG POR ANALIZAR
0	**id** := id + id * cte $
0 id 2	**:=** id + id * cte $
0 id 2 := 3	**id** + id * cte $
0 id 2 := 3 id 7	**+** id * cte $
0 id 2 := 3 F 6	**+** id * cte $
0 id 2 := 3 T 5	**+** id * cte $
0 id 2 := 3 E 4	**+** id * cte $
0 id 2 := 3 E 4 + 9	**id** * cte $
0 id 2 := 3 E 4 + 9 id 7	* cte $
0 id 2 := 3 E 4 + 9 F 6	* cte $
0 id 2 := 3 E 4 + 9 T 11	* cte $
0 id 2 := 3 E 4 + 9 T 11 * 10	**cte** $
0 id 2 := 3 E 4 + 9 T 11 * 10 cte 8	**$**
0 id 2 := 3 E 4 + 9 T 11 * 10 F 12	**$**
0 id 2 := 3 E 4 + 9 T 11	**$**
0 id 2 := 3 E 4	**$**
0 A 1 **COMPILACION EXITOSA!**	**$**

	id	:=	+	*	cte	$	A	E	T	F
0	D2						1			
1										
2		D3								
3	D7				D8			4	5	6
4			D9			R1 : A→ id := E				
5			R3 : E → T	D10		R2 : E→ E+ T				
6			R5 : T → F	R5 : T → F		R5 : T → F				
7			R6 : F → id	R6 : F→ id		R6 : F→ id				
8			R7 : F → cte	R7 : F→ cte		R7 : F→ cte				
9	D7				D8				11	6
10	D7				D8					12
11			R2 : E→ E+ T	D10		R2 : E → E+ T				
12			R4 : T →T* F	R4 : T → T* F		R4 : T → T* F				

Análisis sintáctico

Parsing ascendente

Finalmente el parsing aplicó las reglas :

6, 5, 3, 6, 5, 7, 4, 2, 1, 0

Análisis sintáctico

Parsing ascendente

¿Preguntas?

Análisis sintáctico

Conjuntos primeros y siguientes

Análisis sintáctico

Conjunto primeros

Se define como “primero” de un No Terminal dado, todo aquel Terminal que aparece en primer lugar dentro de algunas de las reglas que definen a dicho No Terminal.

1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <TERMINO> <R>
3. <R> → + <EXPRESION>
4. <R> → λ
5. <TERMINO> → <FACTOR><Q>
6. <Q> → * <TERMINO>
7. <Q> → λ
8. <FACTOR> → id
9. <FACTOR> → cte

Análisis sintáctico

Conjunto primeros

Prim(ASIG)
Por regla 1 el primer terminal que aparece es un “id”, entonces:
Prim(ASIG) = { id }
Ninguna otra regla tiene ASIG como no terminal del lado izquierdo

1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <TERMINO> <R>
3. <R> → + <EXPRESION>
4. <R> → λ
5. <TERMINO> → <FACTOR><Q>
6. <Q> → * <TERMINO>
7. <Q> → λ
8. <FACTOR> → id
9. <FACTOR> → cte

Como EXPRESION tiene como primer elemento a TERMINO, el conjunto Prim(TERMINO) está incluido en Prim(EXPRESION)
Prim(EXPRESION) = Prim(TERMINO) = { id, cte }

Prim(R) = { + }

Como TERMINO tiene como primer elemento a FACTOR, lo que esté en conjunto primeros de FACTOR estará en primeros de TERMINO
Prim(TERMINO) = Prim(FACTOR) = { id, cte }

Análisis sintáctico

Conjunto primeros

Para Q, el primero es un signo "*" ya que el [vacío] no se tiene en cuenta,
Prim(Q) = { * }

En ambas reglas de FACTOR lo que aparece primero son el "id" y la "cte", entonces:

Prim(FACTOR) = { id, cte }

1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <TERMINO> <R>
3. <R> → + <EXPRESION>
4. <R> → λ
5. <TERMINO> → <FACTOR><Q>
6. <Q> → * <TERMINO>
7. <Q> → λ
8. <FACTOR> → id
9. <FACTOR> → cte

Si al momento de obtener los primeros de un No Terminal, el primer símbolo de su regla es otro No Terminal, que deriva en λ entonces deberán obtenerse los primeros del siguiente símbolo

Análisis sintáctico

Conjunto siguientes

Se define como "siguiente" de un No Terminal dado, todo aquel Terminal que aparece inmediatamente a la derecha del No Terminal en cualquier derivación.

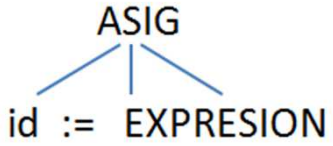
1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <TERMINO> <R>
3. <R> → + <EXPRESION>
4. <R> → λ
5. <TERMINO> → <FACTOR><Q>
6. <Q> → * <TERMINO>
7. <Q> → λ
8. <FACTOR> → id
9. <FACTOR> → cte

Análisis sintáctico

Conjunto siguientes

Sgt(ASIG)

El símbolo **$** está incluido en el conjunto Siguiente del símbolo distinguido (start) ya que representa el final de la cadena de datos a analizar.



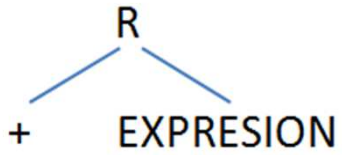
Sgt(ASIG) = { $ }

1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <TERMINO> <R>
3. <R> → + <EXPRESION>
4. <R> → λ
5. <TERMINO> → <FACTOR><Q>
6. <Q> → * <TERMINO>
7. <Q> → λ
8. <FACTOR> → id
9. <FACTOR> → cte

Al observar todas las reglas en donde se encuentra derivado el No terminal EXPRESION (lado derecho) notamos que:

EXPRESION posee, por un lado la regla 1, por la cual lo que se encuentre al final de ASIG seguirá al final de EXPRESION.

Por otro lado, por la regla 3 lo que haya al final de R será lo mismo que hay al final de EXPRESION



Sgt(EXPRESION) = Sgt(ASIG) U Sgt(R) = { $ }

En la regla 2 lo que hay al final de EXPRESION es lo que tiene R al final

Sgt(R) = Sgt(EXPRESION) = { $ }

Análisis sintáctico

Conjunto siguientes

Por regla 2, lo que "viene" después de TERMINO es lo primero de R. Pero si R es vacío, lo que "viene" es lo último de EXPRESION
Por regla 6 los siguientes de Q forman parte de los siguientes de TERMINO.
Sgt(TERMINO) = Prim(R) U Sgt(EXPRESION) U Sgt(Q) = { +, $ }

1. <ASIG> → id := <EXPRESION>
2. <EXPRESION> → <TERMINO> <R>
3. <R> → + <EXPRESION>
4. <R> → λ
5. <TERMINO> → <FACTOR><Q>
6. <Q> → * <TERMINO>
7. <Q> → λ
8. <FACTOR> → id
9. <FACTOR> → cte

Por regla 5 los siguientes de TERMINO están incluidos en los Sgt(Q)
Sgt(Q) = Sgt(TERMINO) = { +, $ }

Por regla 5, después de FACTOR están los primeros de Q. Por regla 6, después de FACTOR encontramos el primero de Q.
Si Q es vacío en la regla 5, lo que sigue es el siguiente de TERMINO
Sgt(FACTOR) = Prim(Q) U Sgt(TERMINO) = { *, +, $ }