Analizador Sintáctico Ascendente

Un **Analizador Sintáctico (A. St.) Ascendente** construye el árbol desde las hojas hacia la raíz. Funciona por **reducción-desplazamiento**, lo cual quiere decir que, siempre que puede, aplica una regla (reduce) y, cuando no, desplaza a su pila de trabajo el *token* que le está pasando el A. Léxico. El *parse* del árbol que construye es justo el inverso del que se obtendría al construir un árbol descendente aplicando derivaciones a la derecha.

Se puede construir un Analizador Sintáctico Ascendente eficiente usando una gramática LR. Las gramáticas LR son lo suficientemente expresivas para generar la mayoría de las construcciones sintácticas de los lenguajes de programación. Un A. St. LR (k), mirando un máximo de *k* símbolos terminales (los *k* siguientes *tokens* que el A. Léxico le pasa al A. St.) sabe con certeza qué acción ha de realizar. En concreto, vamos a trabajar con gramáticas LR(1), con lo que solo se necesita saber cuál es el siguiente *token* de la cadena de entrada.

Este analizador utiliza **una pila** auxiliar de trabajo, en la que va construyendo de manera implícita el árbol de derivación, y cuenta con una tabla de análisis que tiene dos partes: **la tabla ACCIÓN y la tabla GOTO**.

En la **pila** de trabajo se van insertando símbolos gramaticales y estados, de tal forma que encima de cada símbolo hay un estado (en la cima hay un estado). Inicialmente, la pila tiene un símbolo de fondo de pila. En cada instante, el analizador mira el estado de la cima de la pila y el *token* actual, y accede a la tabla ACCIÓN para saber qué hacer.

La tabla **ACCIÓN** tiene, por filas, los posibles estados y, por columnas, los símbolos terminales de la gramática (los *tokens*); sus celdas contienen acciones. La tabla **GOTO** que tiene, por filas, los posibles estados y, por columnas, los no terminales de la gramática, contiene estados.

En cada celda de la tabla ACCIÓN habrá una de las cuatro posibles acciones que puede llevar a cabo este analizador: Aceptar, Reducir, Desplazar o Error. Es importante destacar que, si la gramática no es adecuada para este método de A. St. (**si la gramática no es LR(1)**), al tratar de construir la tabla se obtendrán, en algún caso, dos o más acciones diferentes para una misma celda de la tabla. En ese caso se dice que se ha producido un conflicto. Hay dos tipos de **conflicto** posibles: **reducción-reducción** (una misma celda indica que hay que reducir por dos reglas diferentes), **y reducción-desplazamiento** (una celda indica que hay que reducir y también que hay que desplazar).

Cada una de las celdas podrá contener las acciones: Aceptar, Reducir *n* (que significa "reducir por la regla *n*"), Desplazar *m* (que significa "desplazar el *token* a la pila y apilar encima el estado *m*) o la celda estará en blanco (una celda en blanco corresponde a la acción error).

- Si la acción es aceptar, el analizador termina con éxito, es decir, reconociendo la cadena como correcta.
- Si la acción es error (la celda está en blanco), el analizador llama al gestor de errores (con un código indicativo del error), pues ha detectado que la cadena tiene un error sintáctico.
- Cuando la acción es reducir por la regla *n*, se aplica dicha regla. Suponiendo que la regla *n* es $X \rightarrow A\ z\ B\ C$, el analizador tendrá que sacar de la pila 8 elementos (el doble que el número de símbolos que hay en el consecuente de la regla), que corresponderán a cada uno de los símbolos del consecuente y a los estados que se han apilado sobre ellos. Quedará entonces en la cima un estado (llamémosle *s*). El analizador apilará entonces el antecedente de la regla (*X*) y necesita apilar un estado por encima. Accediendo a *GOTO[s,X]* encuentra cuál es el estado que tiene que apilar.
- Si la acción es desplazar, en la tabla pondrá desplazar *m*. Esto quiere decir que el *token* que está pasando el A. Léxico se pone en la cima de la pila y, encima de él, se apila el estado *m*.

Se incluye a continuación el **algoritmo de un Analizador Sintáctico LR:**

Entrada: Una cadena de entrada *w* con el delimitador *$* por la derecha
La tabla de análisis LR con las funciones *ACCIÓN* y *GOTO* para una gramática *G'*

Salida: Un análisis ascendente de *w*, en caso de que pertenezca a *L(G)*, o la indicación de *error* en caso contrario

Procedimiento:

Inicialmente, la pila contiene el estado inicial I_0, y la cadena de *tokens* de la entrada *w$* está pendiente de ser analizada. El analizador ejecuta entonces el siguiente algoritmo, que terminará al encontrar una acción de *aceptar* o de *error*:

```
a:= AL()     // se pide un token al Analizador Léxico
repeat
    sea s el estado de la cima de la pila
    if ACCIÓN[s,a] = desp s'
    then begin
       meter a en la pila
       meter s' en la pila
       a:= AL()     // se pide el siguiente token al Analizador Léxico
    end
    else if ACCIÓN[s,a] = red A→ β
    then begin
       sacar 2*|β| símbolos de la pila
       sea s' el estado que está ahora en la cima de la pila
       meter A en la pila
       obtener GOTO[s',A] y meterlo en la pila
       // se genera el parse correspondiente a esta regla
    end
    else if ACCION[s,a] = aceptar then return
    else error()
end
```

Una vez explicado su funcionamiento, para poder construir un A. St. LR es necesario aprender a construir las tablas ACCIÓN y GOTO. Se va a explicar aquí el método más sencillo de construcción de estas tablas, llamado SLR o *Simple LR*(1). En primer lugar, es necesario estudiar una serie de conceptos previos, que son: gramática aumentada, forma sentencial, prefijo viable, ítem LR(0), cierre de un conjunto de items, Goto de un conjunto de ítems con un símbolo gramatical y, por último, construcción de la colección canónica de ítems LR(0).

Gramática aumentada

Si *G* es una gramática con símbolo inicial *S*, la gramática aumentada *G'* es la que incluye el símbolo *S'* y la producción $S' \rightarrow S$.

Forma sentencial

Una forma sentencial de G es una secuencia de símbolos terminales y no terminales tal que se puede llegar a ella desde el axioma aplicando reglas de G: $\{X \in (T \cup N)^* / S \overset{*}{\Rightarrow} X\}$. Una **cadena o sentencia** es una forma sentencial en la que todos los símbolos son terminales. El conjunto de todas las cadenas que se pueden generar mediante las reglas de la gramática constituye el **lenguaje generado por la gramática**: $L(G) = \{w \in T^* / S \overset{*}{\Rightarrow} w\}$. La cadena se corresponde con las hojas del árbol de derivación mientras que una forma sentencial se correspondería con los nodos hijos (no hojas) en un punto intermedio de la construcción del árbol.

Prefijo viable

Es un prefijo de una forma sentencial derecha que no excede del extremo derecho del consecuente de la regla a aplicar. Por tanto, dicho prefijo estará contenido en la pila del analizador cuando se plantee aplicar dicha reducción.

Mirando el estado de la cima de la pila y el *token* actual, el A. St. sabe si puede reducir y por qué regla. El estado de la cima de la pila está encima del prefijo viable, así que, realmente, este prefijo es el que permite saber cuál es la siguiente acción a realizar. El conjunto de todos los prefijos viables constituye un lenguaje regular y, por tanto, se puede reconocer con un Autómata Finito Determinista. Construyendo este autómata se podrán obtener automáticamente las tablas ACCIÓN y GOTO del A. St. Los estados del autómata reconocedor de los prefijos viables son los elementos de la colección canónica de items LR(0). A continuación se explica cómo se obtiene dicha colección, explicando primero el concepto de ítem LR(0) y de las operaciones Cierro y Goto de un conjunto de items.

Ítem LR(0)

Un ítem LR(0) tiene la apariencia de una regla gramatical pero con un punto ("·") en alguna posición del consecuente. Este punto separa la parte del consecuente que ya se ha recibido, de la que se espera recibir para poder aplicar esa reducción. Por ejemplo, el ítem $E \rightarrow E + \cdot T$, indica que ya se ha recibido hasta el *token* '+'. Para la regla $A \rightarrow \lambda$, el ítem $A \rightarrow \cdot$ es el único posible.

Cierre de un conjunto de items LR(0)

Si I es un conjunto de items LR(0) de una gramática G, entonces *cierre(I)* es el conjunto de items construido a partir de I mediante las dos siguientes reglas:

1. Inicialmente, todos los elementos de I se añaden a *cierre(I)*
2. Si $A \rightarrow \alpha \cdot B\beta$ está en *cierre(I)* y $B \rightarrow \gamma$ es una producción de G, entonces el ítem $B \rightarrow \cdot \gamma$ se añade a *cierre(I)* si todavía no está. Se sigue aplicando esta regla 2 hasta que no se puedan añadir más items nuevos a *cierre(I)*.

El algoritmo, por tanto, es el siguiente:

```
function cierre (I)
begin
   J:= I
   repeat
    for
      cada ítem A→ α · B β en J y cada regla B→ γ de G tal que B→ · γ no esté ya en J
    do
      añadir B→ · γ  a J
   until no se puedan añadir más elementos a J
   return J
end
```

Goto de un conjunto de items y un símbolo gramatical

Se define *goto(I,X)* como el cierre del conjunto de todos los ítems $[A \rightarrow \alpha X \cdot \beta]$ tales que $[A \rightarrow \alpha \cdot X \beta]$ esté en I. Intuitivamente, si I es el conjunto de items válidos para algún prefijo viable γ, *goto(I,X)* es el conjunto de items válidos para el prefijo viable γX.

Construcción de la colección canónica de items LR(0) para una gramática aumentada G'

El algoritmo es el siguiente:

```
function colección (G')
begin
   C:= {cierre({[S'→ · S]})}
   repeat
    for
      cada conjunto de items I en C y cada símbolo gramatical X tal que goto(I,X) no
         esté vacío y no esté ya en C
    do
      añadir goto(I,X) a C
   until no se puedan añadir más conjuntos de elementos a C
   return C
end
```

La colección canónica de items LR(0) se corresponde con los estados del Autómata Finito Determinista capaz de reconocer todos los prefijos viables de la gramática. De este autómata se obtienen las tablas ACCIÓN y GOTO del analizador mediante el procedimiento que se detalla a continuación.

Construcción de la tabla de análisis de un SLR

Entrada: Una gramática aumentada *G'*

Salida: Las funciones *ACCIÓN* y *GOTO* de la tabla de un Analizador Sintáctico SLR para *G'*

Procedimiento:

1. Construir la colección canónica de items LR(0) para *G': C:= {I_0, I_1, ..., I_n}*
2. El estado *i* es el que se obtiene a partir de I_i. Las acciones para el estado *i* se determinan de la siguiente manera:
   a) Si *[A→ α · a β] ∈ I_i* y *goto(I_i,a) = I_j*, entonces *ACCIÓN[i,a]:= "**desplazar** j"*. En este caso, *a* debe ser un terminal y *j* es el estado que se apila tras el símbolo desplazado.
   b) Si *[A→ α ·] ∈ I_i*, entonces *ACCIÓN[i,a]:= "**reducir** por A→ α"*, para todo terminal *a* perteneciente a *FOLLOW(A)*. En este caso, *A* es cualquier no terminal excepto *S'*.
   c) Si *[S'→ S ·] ∈ I_i*, entonces *ACCIÓN[i,$]:= "**aceptar**"*.
3. Si estas reglas generan conflicto en alguna casilla (más de una acción), se dice que la gramática no es SLR(1). En tal caso, no es posible construir un Analizador Sintáctico mediante este algoritmo para dicha gramática *G'*.
4. Las transiciones *GOTO* del estado *i* se construyen para todos los no terminales *A* mediante la regla: si *goto(I_i,A) = I_j*, entonces *GOTO[i,A] = j*.
5. Todas las casillas no definidas por las reglas 2 y 3 quedan en blanco y corresponden a los casos de *"**error**"*.
6. El estado inicial del analizador es el construido a partir del conjunto de items que contiene a *[S'→ · S]*.

Ejemplo

Se quiere construir la tabla SLR para la siguiente gramática G:

S→ x B z | A
B→ y A | λ
A→ z B z

La gramática aumentada G' será:

S'→ S
S→ x B z | A
B→ y A | λ
A→ z B z

Se construye la colección canónica *C* de items LR(0) para *G'* que estará formada por los siguientes estados:

I_0 = cierre ({S'→·S}) = {S'→ · S, S→ · x B z, S→ · A, A→ · z B z} → *C*
I_1 = goto (I_0, S) = cierre ({S'→ S ·}) = {S'→ S ·} → *C*
I_2 = goto (I_0, B) = cierre (∅) = ∅
I_2 = goto (I_0, A) = cierre ({S→ A ·}) = {S→ A ·} → *C*
I_3 = goto (I_0, x) = cierre ({S→ x · B z}) = {S→ x · B z, B→ · y A, B→ ·} → *C*
I_4 = goto (I_0, y) = cierre (∅) = ∅
I_4 = goto (I_0, z) = cierre ({A→ z · B z}) = {A→ z · B z, B→ · y A, B→ ·} → *C*
I_5 = goto (I_1, S) = cierre (∅) = ∅
I_5 = goto (I_1, B) = cierre (∅) = ∅
I_5 = goto (I_1, A) = cierre (∅) = ∅

I_5= goto (I_1, x)= cierre (∅)= ∅
I_5= goto (I_1, y)= cierre (∅)= ∅
I_5= goto (I_1, z)= cierre (∅)= ∅
I_5= goto (I_2, S)= cierre (∅)= ∅
I_5= goto (I_2, B)= cierre (∅)= ∅
I_5= goto (I_2, A)= cierre (∅)= ∅
I_5= goto (I_2, x)= cierre (∅)= ∅
I_5= goto (I_2, y)= cierre (∅)= ∅
I_5= goto (I_2, z)= cierre (∅)= ∅
I_5= goto (I_3, S)= cierre (∅)= ∅
I_5= goto (I_3, B)= cierre ({S→ x B · z})= {S→ x B · z} → C
I_6= goto (I_3, A)= cierre (∅)= ∅
I_6= goto (I_3, x)= cierre (∅)= ∅
I_6= goto (I_3, y)= cierre ({B→ y · A})= {B→ y · A, A→ · z B z} → C
I_7= goto (I_3, z)= cierre (∅)= ∅
I_7= goto (I_4, S)= cierre (∅)= ∅
I_7= goto (I_4, B)= cierre ({A→ z B · z})= {A→ z B · z} → C
I_8= goto (I_4, A)= cierre (∅)= ∅
I_8= goto (I_4, x)= cierre (∅)= ∅
I_8= goto (I_4, y)= cierre ({B→ y · A})= {B→ y · A, A→ · z B z}= I_6 ∈ C
I_8= goto (I_4, z)= cierre (∅)= ∅
I_8= goto (I_5, S)= cierre (∅)= ∅
I_8= goto (I_5, B)= cierre (∅)= ∅
I_8= goto (I_5, A)= cierre (∅)= ∅
I_8= goto (I_5, x)= cierre (∅)= ∅
I_8= goto (I_5, y)= cierre (∅)= ∅
I_8= goto (I_5, z)= cierre ({S→ x B z ·})= {S→ x B z ·} → C
I_9= goto (I_6, S)= cierre (∅)= ∅
I_9= goto (I_6, B)= cierre (∅)= ∅
I_9= goto (I_6, A)= cierre ({B→ y A ·})= {B→ y A ·} → C
I_{10}= goto (I_6, x)= cierre (∅)= ∅
I_{10}= goto (I_6, y)= cierre (∅)= ∅
I_{10}= goto (I_6, z)= cierre ({A→ z · B z})= {A→ z · B z, B→ · y A, B→ ·}= I_4 ∈ C
I_{10}= goto (I_7, S)= cierre (∅)= ∅
I_{10}= goto (I_7, B)= cierre (∅)= ∅
I_{10}= goto (I_7, A)= cierre (∅)= ∅
I_{10}= goto (I_7, x)= cierre (∅)= ∅
I_{10}= goto (I_7, y)= cierre (∅)= ∅
I_{10}= goto (I_7, z)= cierre ({A→ z B z ·})= {A→ z B z ·} → C
I_{11}= goto (I_8, S)= cierre (∅)= ∅
I_{11}= goto (I_8, B)= cierre (∅)= ∅
I_{11}= goto (I_8, A)= cierre (∅)= ∅
I_{11}= goto (I_8, x)= cierre (∅)= ∅
I_{11}= goto (I_8, y)= cierre (∅)= ∅
I_{11}= goto (I_8, z)= cierre (∅)= ∅
I_{11}= goto (I_9, S)= cierre (∅)= ∅
I_{11}= goto (I_9, B)= cierre (∅)= ∅
I_{11}= goto (I_9, A)= cierre (∅)= ∅
I_{11}= goto (I_9, x)= cierre (∅)= ∅
I_{11}= goto (I_9, y)= cierre (∅)= ∅
I_{11}= goto (I_9, z)= cierre (∅)= ∅
I_{11}= goto (I_{10}, S)= cierre (∅)= ∅
I_{11}= goto (I_{10}, B)= cierre (∅)= ∅
I_{11}= goto (I_{10}, A)= cierre (∅)= ∅
I_{11}= goto (I_{10}, x)= cierre (∅)= ∅
I_{11}= goto (I_{10}, y)= cierre (∅)= ∅
I_{11}= goto (I_{10}, z)= cierre (∅)= ∅

Por tanto, la colección canónica es C = {I_0, I_1, I_2, I_3, I_4, I_5, I_6, I_7, I_8, I_9, I_{10}}

Se calculan también los siguientes conjuntos:

Follow (S)= {$}
Follow (B)= {z}
Follow (A)= {z, $}

Se construye la tabla SLR:

estado	ACCIÓN				GOTO		
	x	**y**	**z**	**$**	**S**	**B**	**A**
0	Desp 3		Desp 4		1		2
1				Acept			
2				Red S→A			
3		Desp 6	Red B→λ			5	
4		Desp 6	Red B→λ			7	
5			Desp 8				
6			Desp 4				9
7			Desp 10				
8				Red S→xBz			
9			Red B→yA				
10			Red A→zBz	Red A→zBz			

Como solo hay una entrada máximo por celda (no hay conflictos), la Gramática G es SLR.