Construcción de tablas de análisis sintáctico LL(1)

Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática

Diego Centeno
Gerardo Cortés
Juan Diego Alfaro

Resumen. A la medida en que los lenguajes de programación se hacían populares y su uso se generalizara, se hizo necesario crear técnicas de interpretación de palabras y frases parecidas al lenguaje humano, a código ejecutable por una máquina. Uno de esas técnica, es la construcción de tablas de análisis sintáctico LL(1), la cual permite conocer si un lenguaje puede ser interpretado en código de máquina, asegurando que no existan problemas de ambigüedad en el mismo.

1. Introducción.

Un lenguaje de programación es un conjunto de expresiones que describen un conjunto de acciones que un computador debe ejecutar a fin de realizar una tarea determinada. Estas expresiones están formadas producciones gramaticales, que a su vez están formadas por palabras, generalmente en inglés, que deben ser interpretadas en ordenes que un computador pueda ejecutar. Al conjunto de producciones gramaticales de un lenguaje se le denomina, gramática del lenguaje.

Para interpretar estos lenguajes, se construyen programas llamados compiladores, los cuales, mediante técnicas ya establecidas, determinan que el conjunto de instrucciones que conforman un programa, están escritas de manera correcta. Luego de determinar si el programa está escrito correctamente, transforma cada una de esas instrucciones en código ejecutable (binario), que será ejecutado por el computador.

Cada una de las producciones gramaticales que conforman un lenguaje, deben ser capaces de simbolizar una y sólo una operación, de modo que el compilador no tenga problemas de ambigüedad al momento de transformar el programa en lenguaje de máquina. Es en esta parte del proceso, donde la construcción de tablas de análisis sintáctico entra en juego, para determinar que no exista ambigüedad en el lenguaje, y a la vez proporcionando una herramienta para determinar que cada una de las instrucciones de un programa esté bien construida.

Para esto, en el segmento 2, explicaremos el algoritmo que interesa a este artículo exponer, a saber, la construcción de tablas de análisis sintáctico LL(1). En el segmento 3 expondremos un ejemplo de dicho algoritmo, y luego nuestras conclusiones.

2. Explicación del algoritmo.

Una tabla de análisis sintáctico LL(1) es una matriz bidimensional M con A filas y b columnas, en la que cada fila representa un no terminal (son elementos de la gramática que pueden ser expresados como una producción gramatical) y cada columna representa un terminal (son los elementos atómicos de la gramática) o el símbolo *$* (que simboliza el final de una hilera de caracteres o símbolos); A es la cantidad de no terminales de la gramática y b es la cantidad de terminales.

La tabla es utilizada por un analizador sintáctico predictivo guiado por tablas para analizar sintácticamente una hilera de caracteres, con el fin de determinar si esta pertenece o no al lenguaje definido por la gramática en cuestión.

¿Qué se necesita para crear una tabla de análisis sintáctico LL(1)? Lo primero que se necesita es una gramática *G*, la cual define el lenguaje que va a reconocer el analizador sintáctico. Luego necesitamos, a partir de dicha gramática, conocer todos los elementos posibles que una con los que una producción gramatical podría iniciar, por lo que obtenemos el conjunto *primero()* de cada no terminal de la gramática. Seguidamente necesitamos conocer es conjunto de todos los posibles elementos que podrían seguir tras finalizar cada producción gramatical, por lo que obtenemos el conjunto *siguiente()* de cada no terminal de la gramática. Con ello finalmente tenemos lo necesario para construir la tabla de análisis sintáctico LL(1). Asumiremos que se conoce el algoritmo para el cálculo de estos conjuntos.

```
Para cada producción A → α de la gramática G {
  Para cada terminal α en PRIMERO(α) {
    Colocar A → α en la entrada M[A, b]
  }
  Si ε está en PRIMERO(α) {
    Colocar A → α en la entrada M[A, c] para cada terminal b en SIGUIENTE(A)
  }
  Si ε está en PRIMERO(α) y $ está en SIGUIENTE(A) {
    Colocar A → α en la entrada M[A, $]
  }
}
```

Figura1. *Algoritmo de construcción de una tabla de análisis sintáctico LL(1) en pseudo código.*

A continuación explicaremos el algoritmo para la construcción de la tabla. Se debe aplicar el algoritmo a cada una de las producciones existentes en la gramática. Para cada producción *A* -> *α* (A deriva a a)se procede como sigue (ver figura1):

- Se examina el conjunto *primero()* del lado derecho de la producción en cuestión. La producción completa se coloca en todas las entradas M[*A*, *b*] tales que *A* es el lado izquierdo de la producción, y *b* es un símbolo terminal que está en el conjunto *primero()* de *α*.
- Luego se debe verificar si el símbolo *ε* (épsilon) está en el *primero()* de *α*, en cuyo caso la producción completa se debe colocar en todas las entradas M[*A*, *c*] tales que *A* es el lado izquierdo de la producción y *c* es un símbolo terminal que está en el conjunto *siguiente()* de *A*.
- Si *ε* está en el conjunto *primero()* de *α,* como en el caso anterior, se verifica también si el símbolo *$* está en el conjunto *siguiente()* de *A*. En este caso, se agrega la producción completa en la entrada M[*A*, *$*], donde *A* es el lado izquierdo de la producción.

Se debe tomar en cuenta que el algoritmo debe aplicarse a todas las producciones, por lo que el ciclo se repite tantas veces como producciones tenga *G*.

3. Ejemplo de construcción de una tabla de análisis sintáctico LL(1).

A fin de ilustrar mejor el algoritmo, presentamos el siguiente ejemplo. Tomaremos para nuestro ejemplo un lenguaje formado por la gramática de la figura 2.

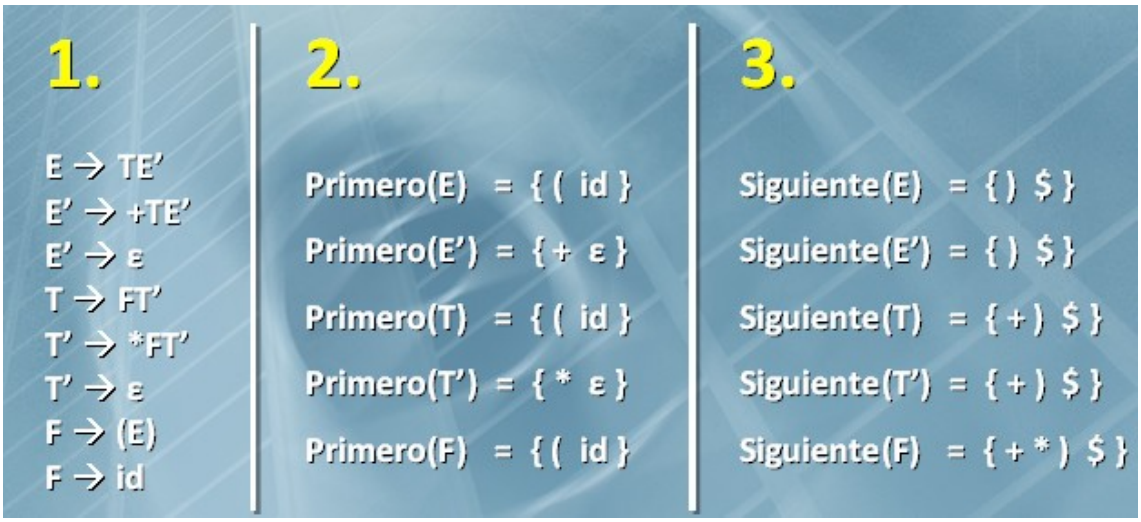

Figura2. *Gramátia del lenguaje con los conjuntos primero() y siguiente() de cada produccion.*

Comenzamos con la primera producción, que es E -> TE'. Localizamos la fila que corresponde al no terminal E en la tabla. Luego, evaluamos el conjunto *primero()* del lado derecho. Como vimos antes, el *primero()* de T es { (id }. Localizamos la columna del terminal (, y colocamos la producción en la entrada [E, (] de la tabla. El siguiente elemento de *primero()* de T es id, de modo que localizamos la columna del terminal id en la tabla y colocamos la producción en cuestión en la entrada [E, id] de la tabla. El siguiente paso es verificar si ε está en el conjunto *primero()* de T. Como no está, continuamos con la siguiente producción. Al final de procesar esta producción, la tabla se ve como en la figura 3.

No terminales	Símbolos de entrada					
	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'						
T						
T'						
F						

Figura3. *Construcción de la tabla de análisis sintáctico después de procesar la producción E->TE'.*

La siguiente producción es E' -> +TE'. Localizamos la fila que corresponde al no terminal E' en la tabla. Luego, evaluamos el conjunto *primero()* del lado derecho. Como + es un terminal, su *primero()* es { + }. Localizamos la columna del terminal +, y colocamos la producción en la entrada [E', +] de la tabla. El siguiente paso es verificar si ε está en el conjunto *primero()* de +. Como no está, continuamos con la siguiente producción. Al final de procesar esta producción, la tabla se ve como en la figura 4.

No terminales	Símbolos de entrada					
	id	+	*	(	)	$
E	E → TE′			E → TE′		
E′		E′ → +TE′				
T						
T′						
F						

Figura4. *Construcción de la tabla de análisis sintáctico después de procesar la producción E'->+TE'.*

La siguiente producción es E' -> ε. Localizamos la fila que corresponde al no terminal E' en la tabla. Luego, evaluamos el conjunto *primero()* del lado derecho. Vemos que ε es el único símbolo en el *primero()* del lado derecho, por lo tanto el siguiente paso es evaluar el conjunto *siguiente()* del lado izquierdo. Como vimos antes, el *siguiente()* de E' es {) $ }. Localizamos la columna del terminal) en la tabla, y colocamos la producción en la entrada [E',)]. El siguiente elemento de *siguiente()* de E' es $. Vemos que ε está en *primero()* de ε y $ está en *siguiente()* de E', por lo tanto debemos colocar la producción en la entrada [E', $]. Al final de procesar esta producción, la tabla se ve como en la figura 5.

No terminales	Símbolos de entrada					
	id	+	*	(	)	$
E	E → TE′			E → TE′		
E′		E′ → +TE′			E′ → ε	E′ → ε
T						
T′						
F						

Figura5. *Construcción de la tabla de análisis sintáctico después de procesar la producción E'->* ε.

De esa manera se procede con cada uno de las producciones gramaticales de la gramática del lenguaje, de modo que la tabla resultante para efectos de nuestro ejemplo es igual a la de la figura 6.

No terminales	Símbolos de entrada					
	id	+	*	(	)	$
E	E → TE′			E → TE′		
E′		E′ → +TE′			E′ → ε	E′ → ε
T	T → FT′			T → FT′		
T′		T′ → ε	T′ → *FT′		T′ → ε	T′ → ε
F	F → id			F → (E)		

Figura6. *Tabla de análisis sintáctico LL(1) finalizada.*

4. Conclusiones.

Construir tablas de análisis sintáctico parecería un trabajo tedioso para nosotros, sin embargo, para un computador es una herramienta muy útil, y que a la larga simplifica el esfuerzo de los creadores de lenguajes de programación tanto como a los programadores.

Si bien es cierto, las tablas de análisis sintáctico LL(1) no brinda toda la información para determinar la eficiencia de un lenguaje de programación. Por otra parte son lo suficientemente expresivas como para detectar la ambigüedad de un lenguaje y brindar al programador un punto de partida en la toma de decisiones mientras construye un lenguaje de programación.

Bibliografía: *"El Libro del Dragón"*