Introducción

En la clase de hoy, mostraremos dos reglas de eliminación del indeterminismo que nos permiten transformar muchas gramáticas no ambiguas en gramáticas LL(1).

Vamos a empezar recordando el algoritmo que vimos en la última clase para diseñar un analizador sintáctico.

Algoritmo para diseñar un analizador sintáctico

(1) Diseñar una gramática incontextual no ambigua que genere el lenguaje que estemos considerando.

(2) Transformar la gramática del paso (1) en una gramática equivalente cuyo autómata con pila asociado se pueda programar.

(3) Programar el autómata con pila asociado a la gramática del paso (2).

El programa del paso (3) será entonces el analizador sintáctico.

Gramáticas LL(1)

Para poder diseñar un analizador sintáctico es necesario partir de una gramática no ambigua, ya que si no lo hiciéramos, habría instrucciones del lenguaje de programación que se podrían interpretar de diferentes maneras y dar resultados de ejecución diferentes.

Nos interesa entonces tener gramáticas no ambiguas cuyos autómatas con pila asociados sean programables. Una clase importante de gramáticas no ambiguas con esta condición es la clase de las llamadas gramáticas LL(1), que estudiamos en la última clase.

Recordemos el concepto de gramática LL(1).

Las funciones Primeros y Siguientes

Sea $G = (V, \Sigma, P, S)$ una gramática incontextual.

(a) Para toda palabra $\alpha \in (V \cup \Sigma)^*$ definimos el conjunto Primeros(α) de la siguiente manera. Si $\alpha \not\Rightarrow_G^* \lambda$, definimos

$$\text{Primeros}(\alpha) = \{a \in \Sigma : \text{ existe una palabra } \beta \text{ tal que } \alpha \Rightarrow_G^* a\beta\}.$$

Y si $\alpha \Rightarrow_G^* \lambda$, definimos

$$\text{Primeros}(\alpha) = \{a \in \Sigma : \text{ existe una palabra } \beta \text{ tal que } \alpha \Rightarrow_G^* a\beta\} \cup \{\lambda\}.$$

(b) Para toda variable $X \in V$, definimos

$$\text{Siguientes}(X) = \{a \in \Sigma : \text{ existen palabras } \alpha, \beta \text{ tales que } S \Rightarrow_G^* \alpha X a \beta\}.$$

Definición de gramática LL(1)

(c) Si $X \in V$ y $a \in \Sigma$, definimos

$\text{TABLA}[X, a] = \{(X, \beta) \in P : a \in \text{Primeros}(\beta \cdot \text{Siguientes(X)})\}$.

(d) Decimos entonces que la gramática G es LL(1), si el algoritmo de construcción de la tabla de análisis no genera conflictos, es decir, si para todo $X \in V$ y para todo $a \in \Sigma$ se tiene que $\text{TABLA}[X, a]$ no contiene más de un elemento.

Reglas para transformar una gramática no ambigua en una gramática LL(1)

Dos gramáticas G_1 y G_2 son equivalentes, si generan el mismo lenguaje, es decir, si $L(G_1) = L(G_2)$.

Las dos siguientes reglas que veremos nos permiten transformar en muchos casos una gramática no ambigua en una gramática LL(1) equivalente.

Con dichas reglas se pretende eliminar el indeterminismo en el autómata con pila asociado a la gramática no ambigua de la que partimos.

Las dos reglas que veremos son las llamadas regla de factorización y regla de recursión.

Recuérdese que, para simplificar la notación, si $A \longrightarrow \alpha_1, \ldots,$ $A \longrightarrow \alpha_n$ son reglas de una gramática incontextual con una misma parte izquierda, escribiremos $A \longrightarrow \alpha_1 | \ldots | \alpha_n$.

Regla de Factorización

Si tenemos producciones en la gramática de la forma $A \longrightarrow \alpha\beta_1|\ldots|\alpha\beta_n$ donde $\alpha \neq \lambda$ y $n \geq 2$, reemplazar estas producciones por

$A \longrightarrow \alpha A'$,

$A' \longrightarrow \beta_1|\ldots|\beta_n$

donde A' es una nueva variable.

Obsérvese que la aplicación de la Regla de Factorización no cambia el lenguaje generado por la gramática, ya que los dos bloques de producciones generan el lenguaje de la expresión regular $\alpha \cdot (\beta_1 \cup \cdots \cup \beta_n)$.

Regla de Recursión

Si tenemos producciones en la gramática de la forma $A \longrightarrow A\alpha_1|\ldots|A\alpha_n|\beta_1|\ldots|\beta_m$ donde $n \geq 1$ y los β_i no comienzan por A, reemplazar estas producciones por

$A \longrightarrow \beta_1 A'|\ldots|\beta_m A'$,

$A' \longrightarrow \alpha_1 A'|\ldots|\alpha_n A'|\lambda$

donde A' es una nueva variable.

Obsérvese que la aplicación de la Regla de Recursión no cambia el lenguaje generado por la gramática, ya que los dos bloques de producciones generan el lenguaje de la expresión regular $(\beta_1 \cup \cdots \cup \beta_m) \cdot (\alpha_1 \cup \cdots \cup \alpha_n)^*$.

Ejemplo 1

Consideremos la siguiente gramática incontextual G para diseñar una calculadora de dígitos decimales, donde E es el símbolo inicial.

1. $E \longrightarrow T$
2. $E \longrightarrow TOE$
3. $T \longrightarrow A$
4. $T \longrightarrow TPA$
5. $O \longrightarrow +$
6. $O \longrightarrow -$
7. $P \longrightarrow *$
8. $P \longrightarrow /$
9. $A \longrightarrow \underline{int}$
10. $A \longrightarrow \underline{float}$

Ejemplo 1

En G, la variable E genera de manera unívoca una suma/resta de términos y la variable T genera también de manera unívoca un producto/división de factores, que pueden ser o bien números enteros (tipo $\underline{int}$) o bien números decimales (tipo $\underline{float}$). Por tanto, G no es ambigua.

Sin embargo, G no es LL(1), ya que por ejemplo tenemos que las producciones $1, 2 \in TABLA[E, \underline{int}]$.

Queremos transformar entonces la gramática G en una gramática LL(1) equivalente. Para ello, utilizamos las reglas de factorización y recursión.

Ejemplo 1

Aplicando la regla de factorización, reemplazamos las producciones $E \longrightarrow T$ y $E \longrightarrow TOE$ por las producciones $E \longrightarrow TX$, $X \longrightarrow \lambda$ y $X \longrightarrow OE$.

Y aplicando la la regla de recursión, reemplazamos las producciones $T \longrightarrow A$ y $T \longrightarrow TPA$ por las producciones $T \longrightarrow AY$, $Y \longrightarrow PAY$ e $Y \longrightarrow \lambda$.

Ejemplo 1

Por tanto, obtenemos la siguiente gramática G' equivalente a G:

1. $E \longrightarrow TX$
2. $X \longrightarrow \lambda$
3. $X \longrightarrow OE$
4. $T \longrightarrow AY$
5. $Y \longrightarrow PAY$
6. $Y \longrightarrow \lambda$
7. $O \longrightarrow +$
8. $O \longrightarrow -$
9. $P \longrightarrow *$
10. $P \longrightarrow /$
11. $A \longrightarrow \underline{int}$
12. $A \longrightarrow \underline{float}$

Ejemplo 1

La tabla de análisis de G' es la siguiente:

	$+$	$-$	$*$	$/$	int	float
E					1	1
X	3	3				
T					4	4
Y	6	6	5	5		
O	7	8				
P			9	10		
A					11	12

Ejemplo 1

Obsérvese que de la derivación

$$TX \Rightarrow^4 AYX \Rightarrow^{11} \underline{int}\, YX$$

se deduce que $\underline{int} \in$ Primeros(TX) y, por tanto, la producción $1 \in$ TABLA$[S, \underline{int}]$.

Por otra parte, de la derivación

$$TX \Rightarrow^4 AYX \Rightarrow^{12} \underline{float}\, YX$$

se deduce que $\underline{float} \in$ Primeros(TX) y, por tanto, la producción $1 \in$ TABLA$[S, \underline{float}]$.

Ejemplo 1

Obsérvese que de la derivación

$OX \Rightarrow^{7} +E$

se deduce que $+ \in$ Primeros(OX) y, por tanto, la producción $3 \in$ TABLA$[X, +]$.

Por otra parte, de la derivación

$OX \Rightarrow^{8} -E$

se deduce que $- \in$ Primeros(OX) y, por tanto, la producción $3 \in$ TABLA$[X, -]$.

Ejemplo 1

Obsérvese que de la derivación

$AY \Rightarrow^7 \underline{int}Y$

se deduce que $\underline{int} \in$ Primeros(AY) y, por tanto, la producción $4 \in$ TABLA$[T, \underline{int}]$.

Análogamente, tenemos que la producción $4 \in$ TABLA$[T, \underline{float}]$.

Ejemplo 1

Obsérvese que de la derivación

$PAY \Rightarrow^{9} *AY$

se deduce que $* \in$ Primeros(PAY) y, por tanto, la producción $5 \in$ TABLA$[Y, *]$.

Análogamente, tenemos que la producción $5 \in$ TABLA$[T, /]$.

Por otra parte, es claro que por las producciones 8-12, se deduce directamente que $7 \in$ TABLA$[O, +]$, $8 \in$ TABLA$[O, -]$, $9 \in$ TABLA$[P, *]$, $10 \in$ TABLA$[P, /]$, $11 \in$ TABLA$[A, \underline{int}]$ y $12 \in$ TABLA$[A, \underline{float}]$.

Ejemplo 1

Obsérvese que Siguientes$(X) = \emptyset$. Por tanto, la producción 2 no aparece en la tabla de análisis.

Y de las derivaciones

(1) $E \Rightarrow^1 TX \Rightarrow^3 TOE \Rightarrow^4 AYOE \Rightarrow^7 AY + E$

(2) $E \Rightarrow^1 TX \Rightarrow^3 TOE \Rightarrow^4 AYOE \Rightarrow^8 AY - E$

se deduce que $+, - \in$ Siguientes(Y) y, por tanto, la producción 6 pertenece a TABLA$[Y, +]$ y a TABLA$[Y, -]$.

Ejemplo 1

Por tanto, G' es LL(1), ya que su tabla de análisis no tiene conflictos.

Siguiendo entonces el algoritmo que mostramos en la última clase de teoría, el autómata con pila asociado a G se puede programar utilizando la tabla de análisis de G'.

Ejemplo 2

Consideramos la siguiente gramática G, que genera declaraciones de variables en C cuyos tipos son *integer* y *float*.Como estamos en la fase de análisis sintáctico del compilador, representamos a las variables por la categoría sintáctica "identificador", que denotamos por $\underline{id}$. Y representamos por $\underline{int}$ al tipo entero, y por $\underline{float}$ al tipo float. La gramática G está entonces definida por las siguientes producciones:

1. $S \longrightarrow D\,;\, S$
2. $S \longrightarrow D\,;$
3. $D \longrightarrow TV$
4. $T \longrightarrow \underline{int}$
5. $T \longrightarrow \underline{float}$
6. $V \longrightarrow V, \underline{id}$
7. $V \longrightarrow \underline{id}$

Ejemplo 2

La gramática G no es LL(1), porque hay conflictos al construir su tabla de análisis. Por una parte, como $\underline{int} \in$ Primeros(D), se tiene que las producciones 1 y 2 pertenecen a TABLA[S,$\underline{int}$]. Análogamente, las producciones 1 y 2 pertenecen a TABLA[S,$\underline{float}$].

Transformamos entonces la gramática G en una gramática LL(1) equivalente, aplicando las reglas de factorización y recursión.

Aplicando la regla de factorización, reemplazamos las producciones 1 y 2 de G por las producciones:

$S \longrightarrow D\,;\,S'$

$S' \longrightarrow S$

$S' \longrightarrow \lambda$

Ejemplo 2

Y aplicando la regla de recursión, reemplazamos las producciones 6 y 7 de G por:

$V \longrightarrow \underline{id}\, V'$

$V' \longrightarrow, \underline{id}\, V'$

$V' \longrightarrow \lambda$

Obtenemos entonces la gramática G' equivalente a G dada por las siguientes producciones:

Ejemplo 2

1. $S \longrightarrow D\,;\, S'$
2. $S' \longrightarrow S$
3. $S' \longrightarrow \lambda$
4. $D \longrightarrow TV$
5. $T \longrightarrow \underline{int}$
6. $T \longrightarrow \underline{float}$
7. $V \longrightarrow \underline{id}\, V'$
8. $V' \longrightarrow, \underline{id}\, V'$
9. $V' \longrightarrow \lambda$

La tabla de análisis para G' es entonces la siguiente:

Ejemplo 2

TABLA	id	int	float	,	;
S		1	1		
S'		2	2		
D		4	4		
T		5	6		
V	7				
V'				8	9

Obsérvese que de la derivación

$D \Rightarrow^4 TV \Rightarrow^5 \underline{int}\, V$

se deduce que $\underline{int} \in$ Primeros(D) y, por tanto, la producción $1 \in$ TABLA$[S, \underline{int}]$.

Ejemplo 2

Análogamente, de la derivación

$D \Rightarrow^4 TV \Rightarrow^6 \underline{float}\, V$

se deduce que $\underline{float} \in$ Primeros(D) y, por tanto, la producción $1 \in$ TABLA$[S, \underline{float}]$.

De la derivación

$S \Rightarrow^1 D\,; S' \Rightarrow^4 TV\,;\, S' \Rightarrow^5 \underline{int}\, V\,;\, S'$

se deduce que $\underline{int} \in$ Primeros(S) y, por tanto, la producción $2 \in$ TABLA$[S', \underline{int}]$.

Y de la derivación

$S \Rightarrow^1 D\,;\, S' \Rightarrow^4 TV\,;\, S' \Rightarrow^6 \underline{float}\, V\,;\, S'$

se deduce que $\underline{float} \in$ Primeros(S) y, por tanto, la producción $2 \in$ TABLA$[S', \underline{float}]$.

Ejemplo 2

Por otra parte, tenemos que Primeros$(T) = \{\underline{int}, \underline{float}\}$ por las producciones 5 y 6. Por tanto, la producción 4 pertenece a TABLA$[D, \underline{int}]$ y pertenece también a TABLA$[D, \underline{float}]$.

Tenemos que la producción 7 pertenece a TABLA$[V, \underline{id}]$, ya que $\underline{id}$ es el primer símbolo de la parte derecha de 7. Y análogamente, la producción 8 pertenece a TABLA$[V', ,]$, ya que , es el primer símbolo de la parte derecha de 8.

Por último, de la derivación

$S \Rightarrow^1 D\,;\, S' \Rightarrow^4 TV\,;\, S' \Rightarrow^7 T\,\underline{id}\,V'\,;\, S'$

deducimos que $; \in$ Siguientes(V') y, por tanto, la producción $9 \in$ TABLA$[V', ;]$.

Ejemplo 3

Queremos escribir un programa para reconocer expresiones aritméticas de un lenguaje de programación, como C o Java. Este programa es una parte importante del analizador sintáctico del compilador. Para simplificar la exposición, suponemos que en las expresiones aritméticas aparecen únicamente las operaciones suma y producto. Consideramos entonces la siguiente gramática no ambigua G para generar expresiones aritméticas:

1. $E \longrightarrow E + T$
2. $E \longrightarrow T$
3. $T \longrightarrow T * F$
4. $T \longrightarrow F$
5. $F \longrightarrow (E)$
6. $F \longrightarrow \underline{int}$
7. $F \longrightarrow \underline{float}$
8. $F \longrightarrow \underline{id}$
9. $F \longrightarrow \underline{id}(E)$

Ejemplo 3

Hemos incluido la producción 9 para poder tratar llamadas a funciones. Se trata de una gramática standard que se utiliza en el diseño de compiladores para tratar las expresiones aritméticas. Se puede comprobar fácilmente que la gramática G no es ambigua, ya que con la variable E se genera una suma de términos de manera unívoca, con la variable T se genera un producto de factores también de manera unívoca, y con la variable F se generan los átomos de las expresiones aritméticas, que pueden ser expresiones aritméticas entre paréntesis, elementos del tipo integer, elementos del tipo float, identificadores o llamadas a funciones.

Por ejemplo, tenemos la siguiente derivación para la palabra $x = (\underline{id} * \underline{int} + \underline{float}) * \underline{id}$.

Ejemplo 3

$$E \Rightarrow^2 T \Rightarrow^3 T * F \Rightarrow^8 T * \underline{id} \Rightarrow^4 F * \underline{id}$$
$$\Rightarrow^5 (E) * \underline{id} \Rightarrow^1 (E + T) * \underline{id} \Rightarrow^4 (E + F) * \underline{id}$$
$$\Rightarrow^7 (E + \underline{float}) * \underline{id} \Rightarrow^2 (T + \underline{float}) * \underline{id} \Rightarrow^3 (T * F + \underline{float}) * \underline{id}$$
$$\Rightarrow^6 (T * \underline{int} + \underline{float}) * \underline{id} \Rightarrow^4 (F * \underline{int} + \underline{float}) * \underline{id}$$
$$\Rightarrow^8 (\underline{id} * \underline{int} + \underline{float}) * \underline{id}.$$

Sin embargo, G no es LL(1). Por ejemplo, tenemos que las producciones 1 y 2 están en TABLA$[E, \underline{id}]$, ya que $\underline{id} \in$ Primeros$(E) \cap$ Primeros(T).

Queremos transformar entonces la gramática G en una gramática LL(1) equivalente. Para ello, utilizamos las reglas de factorización y recursión.

Ejemplo 3

Aplicando la regla de factorización a las producciones 8 y 9, reemplazamos dichas producciones por las siguientes:

$F \longrightarrow \underline{id}\, F'$

$F' \longrightarrow (E)$

$F' \longrightarrow \lambda$

Aplicando ahora la regla de recursión a las producciones 1 y 2, reemplazamos dichas producciones por las siguientes:

$E \longrightarrow T\, E'$

$E' \longrightarrow +\, T\, E'$

$E' \longrightarrow \lambda$

Ejemplo 3

Por último, aplicamos la regla de recursión a las producciones 3 y 4 reemplazando dichas producciones por las siguientes:

$T \longrightarrow F\,T'$

$T' \longrightarrow *\,F\,T'$

$T' \longrightarrow \lambda$

Obtenemos entonces la gramática LL(1) G' equivalente a G dada por las siguientes producciones:

Ejemplo 3

1. $E \longrightarrow T\,E'$.
2. $E' \longrightarrow +T\,E'$.
3. $E' \longrightarrow \lambda$.
4. $T \longrightarrow FT'$.
5. $T' \longrightarrow *FT'$.
6. $T' \longrightarrow \lambda$.
9. $F \longrightarrow (E)$.
10. $F \longrightarrow \underline{int}$.
11. $F \longrightarrow \underline{float}$.
12. $F \longrightarrow \underline{id}\,F'$.
13. $F' \longrightarrow \lambda$.
14. $F' \longrightarrow (E)$.

Ejemplo 3

Se tiene entonces que G' es una gramática equivalente a G, ya que la aplicación de las reglas de factorización y recursión no cambia el lenguaje generado por la gramática de partida. Es decir, el lenguaje generado por G' es el lenguaje de las expresiones aritméticas. Se tiene además que G' es LL(1), y por tanto podemos utilizar el algoritmo para programar el autómata con pila asociado a una gramática LL(1) visto anteriormente. El programa resultante es entonces el analizador sintáctico para expresiones aritméticas. Por tanto, dicho programa recibe como entrada una tira de símbolos y determina si la entrada es una expresión aritmética.