Introducción

En la primera parte de la clase de hoy, terminaremos el ejemplo del diseño de un programa para reconocer expresiones aritméticas, que empezamos en la última clase.

Y en la segunda parte de la clase, estudiaremos la fase del análisis semántico del compilador, y mostraremos el tipo de gramáticas que se utilizan en esta fase, que son las llamadas "gramáticas de atributos".

Empezaremos recordando las reglas de factorización y recursión, las cuales nos permiten transformar muchas gramáticas incontextuales no ambiguas en gramáticas LL(1).

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

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. Se trata de un programa complejo que no se puede diseñar directamente. Para poderlo diseñar, hay que utilizar las técnicas que hemos estudiado en este tema. 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:

Ejemplo

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

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

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

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, en la última clase, utilizando las reglas de factorización y recursión, obtuvimos la siguiente gramática G' equivalente a G:

Ejemplo

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

Ejemplo

Introducimos la producción 1 de G' para poner el carácter \$ al final de la palabra de entrada. 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. Vamos a demostrar entonces que G' es LL(1). Para ello, calculamos en primer lugar los Primeros y los Siguientes de las variables de G'. Se deduce directamente de las producciones de G' que

$\text{Primeros}(S) = \text{Primeros}(E) = \text{Primeros}(T) = \text{Primeros}(F) = \{(, \underline{id}, \underline{int}, \underline{float}\}$,

$\text{Primeros}(E') = \{+, \lambda\}$,

$\text{Primeros}(T') = \{*, \lambda\}$,

$\text{Primeros}(F') = \{), \lambda\}$.

Ejemplo

Tenemos que

Siguientes$(E) =$ Siguientes$(E') = \{\$,)\}$,

debido a las siguientes derivaciones:

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$$,

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^4 T\$ \Rightarrow^5 FT'\$ \Rightarrow^7 F\$ \Rightarrow^8 (E)\$ \Rightarrow^2 (TE')\$$.

Ejemplo

Tenemos que

Siguientes$(T) =$ Siguientes$(T') = \{\$,), +\}$,

debido a las siguientes derivaciones:

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^4 T\$ \Rightarrow^5 FT'\$$,

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^4 T\$ \Rightarrow^5 FT'\$ \Rightarrow^7 F\$ \Rightarrow^8 (E)\$ \Rightarrow^2$
$(TE')\$ \Rightarrow^4 (T)\$ \Rightarrow^5 (FT')\$$,

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^3 T + TE'\$ \Rightarrow^5 FT' + TE`\$$.

Ejemplo

Y tenemos que

Siguientes(F) = Siguientes(F') = $\{\$,), +, *\}$,

debido a las siguientes derivaciones:

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^4 T\$ \Rightarrow^5 FT'\$ \Rightarrow^7 F\$ \Rightarrow^9 \underline{id}\, F'\$,$

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^4 T\$ \Rightarrow^5 FT'\$ \Rightarrow^7 F\$ \Rightarrow^8 (E)\$ \Rightarrow^2$
$(TE')\$ \Rightarrow^4 (T)\$ \Rightarrow^5 (FT')\$ \Rightarrow^7 (F)\$ \Rightarrow^9 (\underline{id}\, F')\$,$

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^3 T + TE'\$ \Rightarrow^5 FT' + TE`\$ \Rightarrow^7$
$F + TE`\$ \Rightarrow^9 \underline{id}\, F' + TE`\$,$

$S \Rightarrow^1 E\$ \Rightarrow^2 TE'\$ \Rightarrow^3 T + TE'\$ \Rightarrow^5 FT' + TE`\$ \Rightarrow^6$
$F * FT' + TE'\$ \Rightarrow^9 \underline{id}\, F' * FT' + TE'\$.$

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

Ejemplo

TABLA	$\underline{id}$	$\underline{int}$	$\underline{float}$	$+$	$*$	$($	$)$	\$
S	1	1	1			1		
E	2	2	2			2		
E'				3			4	4
T	5	5	5			5		
T'				7	6		7	7
F	9	12	13			8		
F'				10	10	11	10	10

Ejemplo

Obsérvese que:

la producción 4 pertenece a TABLA$[E',)]$ y a TABLA$[E', \$]$, porque Siguientes$(E') = \{\$,)\}$,

la producción 7 pertenece a TABLA$[T', +]$, a TABLA$[T',)]$ y a TABLA$[T', \$]$, porque Siguientes$(T') = \{\$,), +\}$,

y la producción 10 pertenece a TABLA$[F', +]$, a TABLA$[F', *]$, a TABLA$[F',)]$ y a TABLA$[F', \$]$, porque Siguientes$(F') = \{\$,), +, *\}$.

Como no hay conflictos en la tabla de análisis de G', tenemos que G' es LL(1). Por tanto, el autómata con pila asociado a G' se puede programar, utilizando el algoritmo que vimos para programar el autómata con pila asociado a una gramática LL(1), es decir, el algoritmo que mostramos a continuación. El programa asociado al autómata con pila de G' es entonces el analizador sintáctico para reconocer expresiones aritméticas.

Programación de autómatas con pila asociados a gramáticas LL(1)

public boolean analisis_sintactico (String entrada)
{ Stack⟨Character⟩ pila = new Stack⟨Character⟩();
int q = 0, i = 0; boolean b = true; char c = entrada.charAt(0);
(1) Poner '$' en la pila
(2) Aplicar la transicion $((q_0, \lambda, \lambda), (f, S))$
(3) while ((tope de la pila ! = '$' || caracter de entrada ! = '$')
&& b)
{ if (tope de la pila == 'a' && caracter de entrada == 'a')
{aplicar la transicion $((f, a, a), (f, \lambda))$; leer siguiente caracter;}
else if (tope de la pila == 'X' && caracter de entrada == 'a'
&&TABLA$[X, a] == X \longrightarrow X1 \dots X_n$)
aplicar la transicion $((f, \lambda, X), (f, X1 \dots X_n))$;
else b = false; }
(4) if (tope de la pila == '$' && caracter de entrada == '$')
return true; else return false;
}

Análisis semántico

Como ya hemos indicado anteriormente, el analizador sintáctico verifica que la palabra que recibe como entrada cumple las reglas del lenguaje de programación. Y si es así, proporciona como representación del proceso de análisis realizado el árbol de derivación de la palabra que recibe como entrada, el cual es utilizado por el analizador semántico para verificar las restricciones semánticas del lenguaje de programación.

Además, en la fase del análisis semántico se realiza la construcción de la llamada tabla de símbolos del compilador, en la cual se almacena la información necesaria sobre los identificadores que se utilizan en el programa: nombre, tipo, valor, dirección de memoria, categoría (si es variable, constante, método, función, ...), tamaño (si se trata de un vector o una matriz),etc. El analizador semántico utiliza entonces el árbol de derivación de la palabra de entrada y la tabla de símbolos del compilador.

Gramáticas de atributos

Para realizar el análisis semántico, se utilizan las llamadas gramáticas de atributos, las cuales son un refinamiento de las gramáticas incontextuales que hemos estudiado hasta ahora.

Queremos dar significado a los símbolos de una gramática incontextual. Para ello, asignamos atributos a los símbolos de la gramática.

En general, los atributos representan propiedades del lenguaje de programación, como pueden ser el tipo de datos de una variable, el valor de una expresión o la ubicación de una variable en memoria.

Gramáticas de atributos

Entonces, para poder realizar el análisis semántico hay que manipular los atributos asignando a las reglas de la gramática incontextual código de un lenguaje de programación. A dichos códigos se les llama acciones semánticas.

Definimos entonces una gramática de atributos como una gramática incontextual $G = (V, \Sigma, P, S)$, en la cual hemos definido atributos para un subconjunto de símbolos de $V \cup \Sigma$ y hemos definido acciones semánticas para las producciones de P.

Observaciones

(1) Los atributos que aparezcan en una acción semántica asociada a una regla de la gramática deben ser atributos de los símbolos de esa regla de la gramática.

(2) Los símbolos de una regla de una gramática de atributos no pueden estar repetidos. Para ello, hay que renombrar los símbolos que se repitan en una regla de la gramática.

El punto (2) es necesario, porque si no se renombran los símbolos en las reglas de una gramática, los atributos pueden quedar indefinidos.

Observaciones

Entonces, el análisis semántico para una expresión que estemos considerando se efectúa recorriendo el árbol de derivación de la expresión y ejecutando las acciones semánticas correspondientes.

Si a es un atributo asignado a un símbolo X de una gramática, nos referiremos a él como $X.a$.

Ejemplo

Consideremos la siguiente gramática incontextual G para declarar variables de tipo entero o de tipo real.

1. $S \to TX$
2. $T \to \underline{int}$
3. $T \to \underline{float}$
4. $X \to L\,;$
5. $L \to \underline{id}, L$
6. $L \to \underline{id}$

Asignamos el atributo $.\text{tipo}$ a los símbolos T, X, L e $\underline{id}$.

Asignamos entonces las siguientes acciones semánticas a las reglas de G:

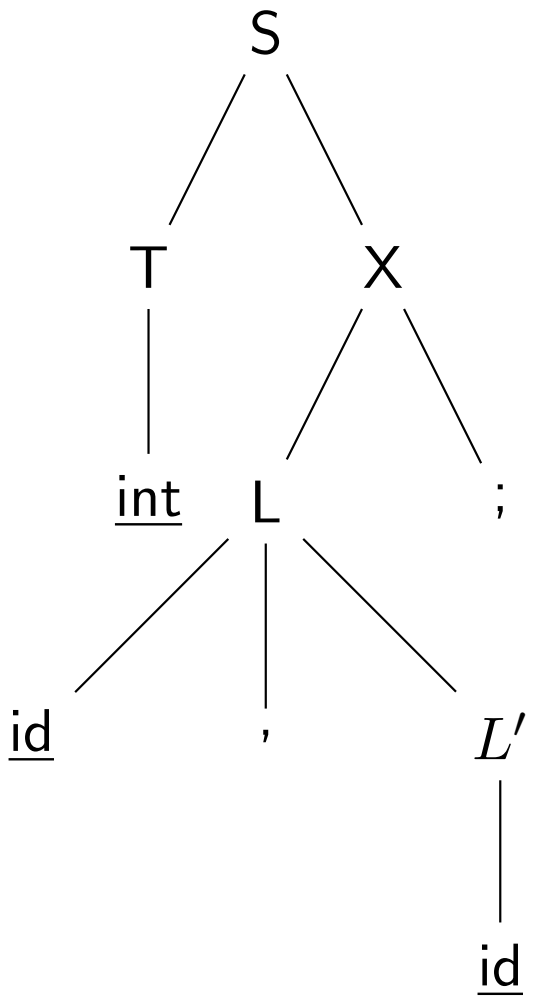
Ejemplo

Regla gramatical	Acción semántica
1. $S \to TX$	X.tipo = T.tipo
2. $T \to \underline{int}$	T.tipo = integer
3. $T \to \underline{float}$	T.tipo = real
4. $X \to L\,;$	L.tipo = X.tipo
5. $L \to \underline{id}, L'$	$\underline{id}$.tipo = L.tipo; L'.tipo = L.tipo
6. $L \to \underline{id}$	$\underline{id}$.tipo = L.tipo

Consideremos ahora la declaración de variables $\underline{int}\ \underline{id}\ ,\ \underline{id};$

Para hacer entonces el análisis semántico de esta declaración, en primer lugar se considera el árbol de derivación de la declaración:

Ejemplo



Ejemplo

Entonces, por la acción semántica asociada a la regla 2, el analizador semántico deduce que T.tipo $= \underline{integer}$. Ahora, por la acción semántica asociada a la regla 1, deduce que X.tipo $= \underline{integer}$. A continuación, por la acción semántica asociada a la regla 4, deduce que L.tipo $= \underline{integer}$. Entonces, por la acción semántica asociada a la regla 5, deduce que el tipo del primer identificador es un entero. Aplicando de nuevo la acción semántica asociada a la regla 5, deduce que L'.tipo $= \underline{integer}$. Por último, aplicando la acción semántica asociada a la regla 6, deduce que el tipo del segundo identificador de la declaración es también un entero.

Tipos de atributos

Los tipos principales de atributos que se utilizan en el análisis semántico son los llamados atributos sintetizados y atributos heredados.

Un atributo es sintetizado, si su valor se calcula a partir de los valores de los atributos de sus nodos hijos en el árbol de derivación.

Y un atributo es heredado, si su valor se calcula a partir de los valores de los atributos de sus nodos hermanos o nodos padres en el árbol de derivación.

En el ejemplo anterior, tenemos que el atributo .tipo para T es sintetizado, y el atributo .tipo para L, X e $\underline{id}$ es heredado.

Ejemplo

Consideremos la siguiente gramática incontextual G para generar expresiones aritméticas:

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

Asignamos el atributo .valor a los símbolos E, T, F, $\underline{id}$, $\underline{int}$, $\underline{float}$.

Ejemplo

Asignamos entonces las siguientes acciones semánticas a las reglas de G:

Regla gramatical	Acción semántica
1. $E \to E' + T$	E.valor $= E'$.valor $+ T$.valor
2. $E \to T$	E.valor $= T$.valor
3. $T \to T' * F$	T.valor $= T'$.valor $* F$.valor
4. $T \to F$	T.valor $= F$.valor
5. $F \to (E)$	F.valor $= E$.valor
6. $F \to \underline{id}$	F.valor $= \underline{id}$.valor
7. $F \to \underline{int}$	F.valor $= \underline{int}$.valor
8. $F \to \underline{float}$	F.valor $= \underline{float}$.valor

Ejemplo

En este caso, los atributos E.valor, T.valor, F.valor, $\underline{id}$.valor, $\underline{int}$.valor y $\underline{float}$.valor son sintetizados.

Procediendo entonces como en el ejemplo anterior, el analizador semántico calculará el valor de una expresión aritmética, utilizando el árbol de derivación de la expresión y las acciones semánticas de la gramática.