Gramáticas de atributos

Teoría de Lenguajes

Franco Frizzo

27 de mayo de 2024

En esta clase trabajaremos con una extensión de las gramáticas independientes del contexto: las *gramáticas de atributos*. Estas nos permiten ampliar el poder expresivo de las gramáticas, logrando mover información entre distintas partes del árbol de derivación y capturando así ciertas sutilezas que en principio solo podrían ser expresadas por gramáticas sensibles al contexto.

Además, las gramáticas de atributos brindan una forma de asignar *semántica* a las cadenas de un lenguaje. Esto es útil en las etapas del proceso de compilación posteriores al análisis sintáctico: el análisis semántico (donde se puede verificar propiedades no sintácticas de las cadenas, como por ejemplo, la correctitud de tipos) y la traducción de cadenas a otros lenguajes (ya sea para generar el resultado final de la compilación o, más frecuentemente, alguna forma de código intermedio que será utilizado más adelante).

Definiciones

Una **gramática de atributos** es una gramática independiente del contexto extendida con:

- Un conjunto de **atributos** para cada símbolo.
- Un conjunto de **reglas semánticas** para cada producción.

Atributos

Los atributos son variables asociadas a cada símbolo de la gramática, que pueden tomar distintos valores cada vez que el símbolo aparece como nodo de un árbol sintáctico. Cada atributo está definido por un nombre y un tipo de dato (booleano, entero, real, string, etc.), y puede ser:

- **Sintetizado**: Si su valor en un nodo N se define en función de los valores de los atributos de N y sus hijos.
- **Heredado**: Si su valor en un nodo N se define en función de los valores de los atributos de N, su padre y sus hermanos.

Es importante destacar que ser sintetizado o heredado es una propiedad del atributo; es decir, ningún atributo puede ser sintetizado en algunos nodos del árbol y heredado en otros.

Los símbolos terminales solo pueden tener atributos sintetizados, y el valor de los mismos siempre es provisto por el analizador léxico.

Reglas semánticas

Las **reglas semánticas** son expresiones que definen el valor que tomará un atributo al aplicar una producción. Cada regla semántica está asociada a una producción, y es de la forma

$$X.a = f(Y_1.a_1, \ldots, Y_k.a_k)$$

donde $X, Y_1, \ldots, Y_k$ son símbolos que aparecen en dicha producción, y la función f determina el valor del atributo a del símbolo X a partir de los valores de los atributos $a_1, \ldots, a_k$ de los símbolos $Y_1, \ldots, Y_k$. Es muy importante que la función f no tenga efectos secundarios.

Podemos notar que si a es sintetizado, X será la cabeza de la producción, mientras que si a es heredado, X aparecerá en el cuerpo de la producción.

Reglas semánticas condicionales

También podemos definir **reglas semánticas condicionales**. Las mismas son de la forma

$$\textsc{Condición}\text{: } p(X_1.a_1 \ldots X_k.a_k)$$

donde $X_1, \ldots, X_n$ son símbolos de una producción y p es un predicado acerca de los valores de los atributos $a_1, \ldots, a_k$ de los símbolos $X_1, \ldots, X_k$. Este tipo de reglas semánticas permiten restringir el lenguaje generado por la gramática, invalidando los árboles sintácticos en los que algún nodo no cumpla con la condición.

Árboles sintácticos atribuidos

Un **árbol sintáctico atribuido** (o decorado) es un árbol sintáctico al que se le han agregado los valores de los atributos de cada uno de sus nodos.

Gramáticas de atributos bien definidas

Decimos que una gramática de atributos está **bien definida** si, para todo árbol sintático, siempre es posible computar los valores de los atributos. Para esto tienen que pasar dos cosas:

1. Deben estar definidas todas las reglas semánticas necesarias. En particular, para todo símbolo X que tenga un atributo a:
   - si a es *sintetizado*, toda producción de la forma $X \to \alpha$ debe tener asociada exactamente una regla semántica que defina el valor de $X.a$.
   - si a es *heredado*, toda producción de la forma $Y \to \alpha_1 X \alpha_2$ debe tener asociada exactamente una regla semántica que defina el valor de $X.a$.
2. Debe existir, para *todo árbol sintático*, un orden que permita computar los atributos respetando sus **dependencias**. Dado un árbol atribuido, decimos que $N_2.a_2$ *depende* de $N_1.a_1$ si el valor de $N_2.a_2$ se computa en función del valor de $N_1.a_1$. Notar que N_1 y N_2 no son símbolos de la gramática sino nodos del árbol, es decir, apariciones puntuales de un símbolo en el mismo.

   Dado un árbol sintáctico, podemos construir su **grafo de dependencias**, cuyos nodos son los atributos de todos los nodos del árbol, con una arista de $N_1.a_1$ a $N_2.a_2$ siempre que $N_2.a_2$ dependa de $N_1.a_1$. Cada aplicación de una regla semántica condicional también agrega un nodo al grafo, con aristas entrantes desde los atributos necesarios para computar

el predicado. Existirá un orden para computar los atributos si y solo si el grafo no tiene ciclos, en cuyo caso el orden buscado será alguno de los *órdenes topológicos* del mismo.[1]

Notar que para cada árbol sintáctico se obtiene un grafo de dependencias distinto. Para que la gramática esté bien definida, no debe poder construirse *ningún* árbol cuyo grafo contenga ciclos. Existe un algoritmo, propuesto por Donald Knuth[1], que permite construir un grafo de dependencias general para la gramática. En la práctica no utilizaremos este algoritmo; en cambio, emplearemos propiedades y razonamientos *ad-hoc* para asegurarnos de construir siempre gramáticas bien definidas.

Gramáticas S-atribuidas y L-atribuidas

- Una gramática de atributos es **S-atribuida** si todos sus atributos son sintetizados.[2] Las gramáticas S-atribuidas siempre están bien definidas, ya que cualquier manera de recorrer los atributos del árbol de abajo hacia arriba es un orden topológico válido. Además, son sencillas de implementar tanto en parsers LL como LR.
- Una gramática de atributos es **L-atribuida** si todos sus atributos son:
  - o bien *sintetizados*,[3]
  - o bien *heredados* de forma tal que, en toda producción, los atributos de todo símbolo solo dependen de los atributos de símbolos que aparecen a su izquierda (o del mismo símbolo, siempre que no se produzcan dependencias circulares).

  Las gramáticas L-atribuidas también están siempre bien definidas. Para computar todos los atributos, basta hacer un recorrido DFS de izquierda a derecha por los nodos del árbol, computando los atributos heredados de cada uno de ellos antes de visitar sus subárboles, y los atributos sintetizados justo después de haberlos recorrido. Esto hace que sean muy sencillas de implementar en parsers LL.

Ejercicio 1

Sea el lenguaje

$$\mathcal{L}_1 = \{a^n b^m c^{2k} \mid 2k = nm\}.$$

a. Dar una gramática de atributos que genere $\mathcal{L}_1$.

b. Mostrar el árbol atribuido correspondiente a la cadena $aabcc$. Dibujar en el árbol las aristas del grafo de dependencias de los atributos.

c. La gramática obtenida, ¿es S-atribuida? ¿Y L-atribuida? ¿Está bien definida?

d. Modificar la gramática del inciso (a) para que el no terminal C no posea atributos sintetizados. Repetir los incisos (b) y (c).

[1] Dado un grafo dirigido, un orden topológico es una manera de ordenar sus nodos $v_1, \ldots, v_n$ tal que siempre que existe una arista de v_i a v_j, se cumple que $i < j$.

[2] Con la siguiente salvedad: si alguna producción tiene una regla semántica que define los atributos de un símbolo en función de otros atributos del mismo símbolo, no deben introducirse dependencias circulares.

[3] Vale la misma salvedad anterior.

Resolución

a. Como primera observación, notemos que $\mathcal{L}_1$ no es un lenguaje independiente del contexto.[4] Este es un clásico ejemplo en el que agregar atributos a una gramática independiente del contexto nos permitirá aumentar su poder expresivo.

Como punto de partida, definimos una gramática independiente del contexto que genere las cadenas de la forma $a^n b^m c^{2k}$, sin restricciones sobre n, m y k.

Sea $G_1 = \langle \{S, A, B, C\}, \{a, b, c\}, P_1, S \rangle$, con P_1:

$$\begin{array}{lcl} S & \to & ABC \\ A & \to & aA \mid \lambda \\ B & \to & bB \mid \lambda \\ C & \to & ccC \mid \lambda \end{array}$$

Para convertir G_1 en una gramática de atributos, debemos definir:

- Qué *atributos* tendrá cada símbolo, si cada uno será *sintetizado* o *heredado*, y el tipo de datos que tendrán sus valores.
- Las *reglas semánticas* correspondientes a cada producción.

Una solución natural al problema consiste en agregar a cada uno de los no terminales A, B y C un atributo sintetizado que indique la cantidad de terminales generados a partir de ellos. Así, nuestra gramática tendrá los siguientes atributos:

Símbolo	Atributo	Tipo de atributo	Tipo de valor
A	*count*	sintetizado	entero
B	*count*	sintetizado	entero
C	*count*	sintetizado	entero

Luego, agregamos reglas semánticas a las producciones cuyas cabezas sean A, B o C, indicando cómo se computará el valor de estos atributos. Dado que existen producciones donde el no terminal de la cabeza se repite en el cuerpo, agregamos subíndices a las apariciones en el cuerpo para desambiguar las reglas semánticas.

$$\begin{array}{lcll} S & \to & ABC & \\ A & \to & aA_1 & \{\ A.count = 1 + A_1.count\ \} \\ & \mid & \lambda & \{\ A.count = 0\ \} \\ B & \to & bB_1 & \{\ B.count = 1 + B_1.count\ \} \\ & \mid & \lambda & \{\ B.count = 0\ \} \\ C & \to & ccC_1 & \{\ C.count = 2 + C_1.count\ \} \\ & \mid & \lambda & \{\ C.count = 0\ \} \end{array}$$

Con estas reglas, la derivación desde G_1 de cualquier cadena $\alpha = a^n b^m c^{2k}$ tendrá la forma $S \Rightarrow ABC \overset{*}{\Rightarrow} \alpha$, con $n = A.count$, $m = B.count$ y $2k = C.count$.

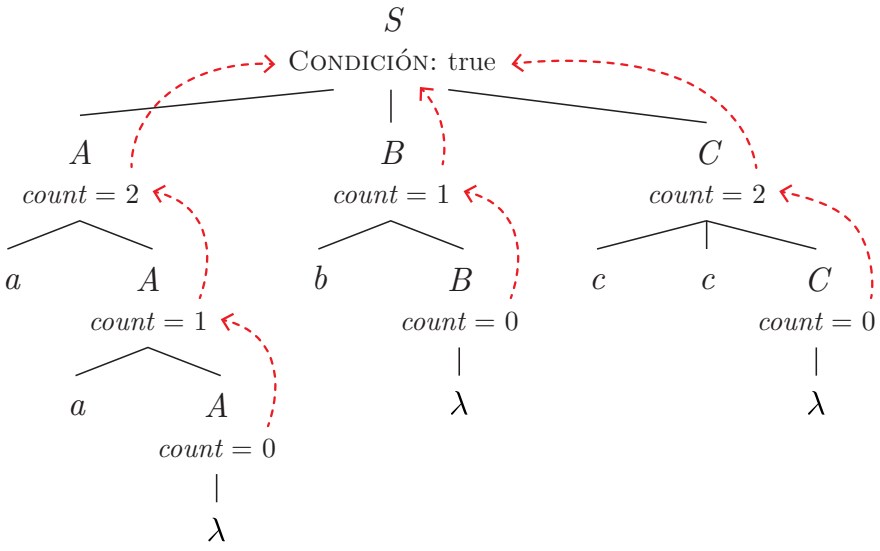
[4] Enunciamos este hecho sin demostración, apelando a la intuición del lector, ya que para probarlo es necesario usar el lema de *pumping* para lenguajes libres de contexto, con el que no trabajamos en la práctica de la materia.

Solo resta agregar una regla semántica a la producción $S \to ABC$ que impida generar cadenas en las que $2k \neq nm$. Para esto podemos usar una regla semántica condicional. Así, nuestro conjunto de producciones con sus respectivas reglas semánticas queda:

$$
\begin{array}{lcll}
S & \to & ABC & \{ \text{CONDICIÓN: } A.count * B.count == C.count \} \\
A & \to & aA_1 & \{ A.count = 1 + A_1.count \} \\
 & | & \lambda & \{ A.count = 0 \} \\
B & \to & bB_1 & \{ B.count = 1 + B_1.count \} \\
 & | & \lambda & \{ B.count = 0 \} \\
C & \to & ccC_1 & \{ C.count = 2 + C_1.count \} \\
 & | & \lambda & \{ C.count = 0 \}
\end{array}
$$

La gramática de atributos pedida queda definida por la tupla de la gramática G_1, más los atributos de cada símbolo (es muy importante indicar, para cada uno, si es sintetizado o heredado y el tipo de datos de sus valores), más las reglas semánticas de cada producción.

b. El árbol atribuido para *aabcc* será el árbol de derivación de la cadena, agregando en cada nodo los valores de sus atributos. Donde se aplica una regla semántica condicional, agregamos el valor de la condición. Indicamos con flechas las dependencias entre atributos.



Dependencia (Sintetizado)

c. La gramática es S-atribuida, ya que solo utiliza atributos sintetizados. También es L-atribuida, ya que todas las gramáticas S-atribuidas lo son. Por lo tanto, podemos afirmar que está bien definida.

d. Dado que C no puede poseer atributos sintetizados, ya no podemos hacer que la información se mueva hacia arriba por el subárbol correspondiente a este no terminal. Por lo tanto, resulta imposible verificar la condición $2k = nm$ en la raíz del árbol de derivación.

La solución consiste en convertir $C.count$ en un atributo heredado, y realizar el chequeo en la hoja más profunda del subárbol correspondiente a C, es decir, en la producción $C \to \lambda$. Tendremos, por lo tanto, los siguientes atributos:

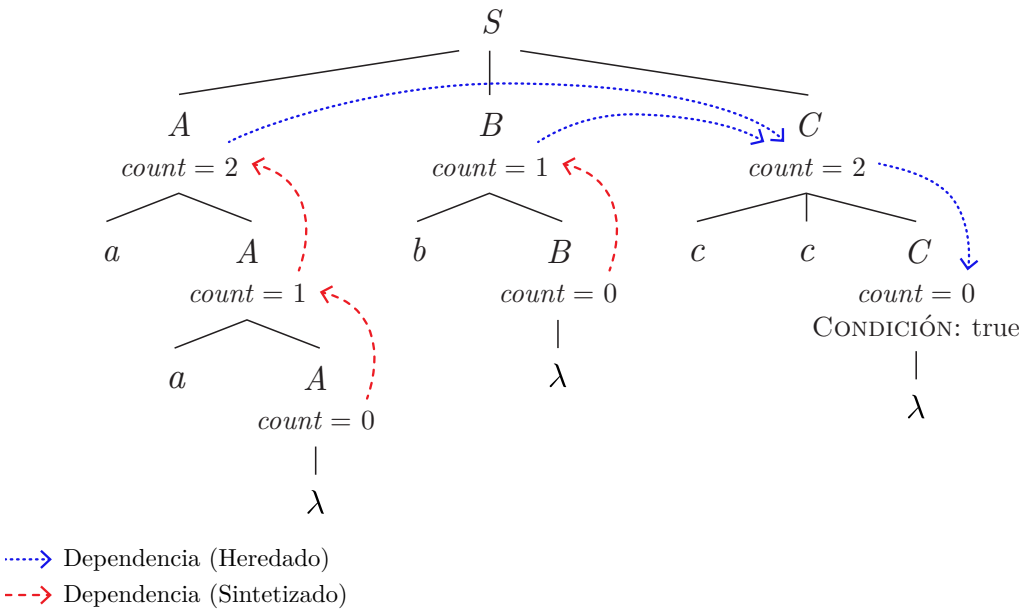
Símbolo	Atributo	Tipo de atributo	Tipo de valor
A	*count*	sintetizado	entero
B	*count*	sintetizado	entero
C	*count*	heredado	entero

y las siguientes reglas semánticas:

$$
\begin{array}{lcll}
S & \to & ABC & \{\ \underline{C.count = A.count + B.count}\ \} \\
A & \to & aA_1 & \{\ A.count = 1 + A_1.count\ \} \\
 & | & \lambda & \{\ A.count = 0\ \} \\
B & \to & bB_1 & \{\ B.count = 1 + B_1.count\ \} \\
 & | & \lambda & \{\ B.count = 0\ \} \\
C & \to & ccC_1 & \{\ \underline{C_1.count = C.count - 2}\ \} \\
 & | & \lambda & \{\ \underline{\text{CONDICIÓN: } C.count == 0}\ \}
\end{array}
$$

En ambos casos, se subrayaron los cambios con respecto a la gramática del inciso (a).

A continuación se exhibe el árbol de derivación atribuido para la cadena *aabcc*, indicando con flechas las dependencias entre los atributos.



Dependencia (Heredado)

Dependencia (Sintetizado)

La nueva gramática ya no es S-atribuida, dado que cuenta con un atributo heredado. Sin embargo, en todos los casos, el valor de este atributo se computa exclusivamente a partir de atributos de símbolos que aparecen a su izquierda en las producciones involucradas. Por lo tanto, la gramática sí es L-atribuida, lo cual nos permite afirmar que está bien definida.

Ejercicio 2

Sea la gramática $G_2 = \langle \{S\}, \{a, b\}, P_2, S \rangle$, con P_2:

$$S \to aS \mid bS \mid \lambda$$

Sintetizar un atributo de tipo entero que represente la máxima cantidad de veces consecutivas que el símbolo a aparece en una cadena.

Resolución

A diferencia del ejercicio anterior, en este caso no vamos a usar una gramática de atributos para restringir las cadenas generadas por nuestra gramática, sino para computar un valor a partir de cada cadena. Es decir, vamos a atribuirle una *semántica* o significado a las cadenas de nuestro lenguaje.

Lógicamente, debemos agregar un atributo sintetizado al no terminal S que represente la máxima cantidad de veces consecutivas que aparece a dentro de la cadena generada por el no terminal (notar que, al tratarse de un atributo del símbolo inicial, es necesariamente sintetizado). Llamamos a este atributo $aMax$.

Símbolo	Atributo	Tipo de atributo	Tipo de valor
S	$aMax$	sintetizado	entero

Para definir las reglas semánticas, los casos más sencillos son aquellas producciones que no agregan ninguna a a la cadena. En el caso de la producción $S \to \lambda$, $S.aMax$ toma, naturalmente, el valor 0. Por su parte, en la producción $S \to bS_1$. el valor del atributo pasa sin modificarse de S_1 a S.

$$\begin{array}{lcll} S & \to & aS_1 & \\ & | & bS_1 & \{\ S.aMax = S_1.aMax\ \} \\ & | & \lambda & \{\ S.aMax = 0\ \} \end{array}$$

¿Qué pasa con la producción $S \to aS_1$? Si justo al comienzo de la cadena generada por S_1 hay una serie de a consecutivas de longitud máxima, al agregar una nueva a a la cadena, el valor del atributo $aMax$ en S será mayor en 1 al valor que tenía en S. En caso contrario, se mantendrá sin cambios.

Para poder distinguir estos dos casos, agregamos un nuevo atributo sintetizado a S que representa la cantidad de a consecutivas que hay al comienzo de la cadena. Llamándolo $aIni$, nuestra gramática queda con los siguientes dos atributos:

Símbolo	Atributo	Tipo de atributo	Tipo de valor
S	$aMax$	sintetizado	entero
S	$aIni$	sintetizado	entero

Usando este nuevo atributo, podemos escribir la regla semántica para $aMax$ en la producción $S \to aS_1$ que habíamos enunciado coloquialmente:

$$S.aMax = S_1.aMax + (\text{if } S_1.aIni == S_1.aMax \text{ then 1 else 0})$$

Por último, también debemos definir reglas semánticas para el nuevo atributo $aIni$. En las producciones $S \to bS_1$ y $S \to \lambda$ se generan cadenas que no comienzan con a, por lo que el valor del atributo debe ser 0. En el caso de la producción $S \to aS_1$, como se agrega una a al comienzo de la cadena generada por S_1, tenemos que $S.aIni = 1 + S_1.aIni$.

Agregando las nuevas reglas semánticas a la gramática, obtenemos:

$$\begin{array}{lcll} S & \to & aS_1 & \{\ \underline{S.aMax = S_1.aMax + (\text{if } S_1.aIni == S_1.aMax \text{ then 1 else 0})}\ ; \\ & & & \underline{S.aIni = 1 + S_1.aIni}\ \} \\ & | & bS_1 & \{\ S.aMax = S_1.aMax\ ;\ \underline{S.aIni = 0}\ \} \\ & | & \lambda & \{\ S.aMax = 0\ ;\ \underline{S.aIni = 0}\ \} \end{array}$$

Ejercicio 3

Sea la gramática $G_3 = \langle \{S, E, T, F\}, \{+, *, (,), \mathbf{num}, x\}, P_3, S \rangle$, con P_3:

$$
\begin{array}{lcl}
S & \rightarrow & E \ (\ \mathbf{num}\ ) \\
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & \mathbf{num} \mid x \mid (\ E\ )
\end{array}
$$

El token **num** representa un número entero, cuyo valor es reconocido por el analizador léxico y almacenado en un atributo denominado *val*.

a. Dar una gramática de atributos que sintetice, para las cadenas generadas por G_2, el valor de la expresión reemplazando todas las apariciones de x por el número indicado entre paréntesis.

b. Dar el árbol atribuido e indicar las dependencias entre atributos para la cadena $3 + 4 * x\ (\ 2\ )$.

c. La gramática dada, ¿es S-atribuida? ¿L-atribuida? ¿Está bien definida?

Resolución

a. Al encarar ejercicios más complejos de gramáticas de atributos, conviene empezar pensando a grandes rasgos el flujo de información que deberá darse en nuestros árboles sintáticos.

Para cumplir con lo pedido en el enunciado, necesitamos agregar un atributo sintetizado *val* al no terminal S que contenga el valor numérico de la expresión. Dicho atributo, que vamos a computar mediante una regla semántica en la producción $S \rightarrow E\ (\ \mathbf{num}\ )$, depende tanto de la estructura de la expresión representada por E como del valor numérico para la variable x contenido en **num**.

Dado que la forma natural de computar el valor de una expresión aritmética es recorriendo el árbol sintáctico de manera *bottom-up*, comenzando por las hojas y ascendiendo por la estructura, lo que haremos será hacer bajar el valor de x por el árbol como un atributo heredado, que llamaremos x y agregaremos a los no terminales E, T y F. Esto nos permitirá tenerlo disponible para computar el valor de cada subexpresión en forma sintetizada, como un atributo *val* que también agregaremos a los no terminales E, T y F.

Así, nuestra gramática queda con los siguientes atributos:

Símbolo	Atributo	Tipo de atributo	Tipo de valor
S	*val*	sintetizado	entero
E	*val*	sintetizado	entero
F	*val*	sintetizado	entero
T	*val*	sintetizado	entero
num	*val*	sintetizado (léxico)	entero
E	x	heredado	entero
F	x	heredado	entero
T	x	heredado	entero

Resta definir las reglas semánticas. Comencemos por las más sencillas, que son las correspondientes al atributo x. El mismo debe tomarse desde **num**.*val* en la producción $S \rightarrow E\ (\ \mathbf{num}\ )$, y luego ser copiado desde la cabeza de la producción en todas las demás

producciones que contienen E, T o F en el cuerpo. Agregando estas reglas a la gramática, obtenemos:

$$\begin{array}{lcll} S & \to & E\ (\ \mathbf{num}\ ) & \{\ E.x = \mathbf{num}.val\ \} \\ E & \to & E_1 + T & \{\ E_1.x = E.x\ ;\ T.x = E.x\ \} \\ & | & T & \{\ T.x = E.x\ \} \\ T & \to & T_1 * F & \{\ T_1.x = T.x\ ;\ F.x = T.x\ \} \\ & | & F & \{\ F.x = T.x\ \} \\ F & \to & \mathbf{num} & \\ & | & x & \\ & | & (\ E\ ) & \{\ E.x = F.x\ \} \end{array}$$

En cuanto a las reglas para sintetizar el atributo val, debemos definirlas en cada producción en función de los valores de las subexpresiones involucradas utilizando donde corresponda la misma operación aritmética representada por la producción. Por ejemplo, en la producción $E \to E_1 + T$, agregaremos la regla

$$E.val = E_1.val + T.val.$$

En el caso de la producción $F \to \mathbf{num}$, basta tomar el valor de $\mathbf{num}.val$ provisto por el analizador léxico, agregando la regla

$$F.val = \mathbf{num}.val.$$

El único caso excepcional es el de la producción $F \to x$, donde el valor a considerar será el del atributo x del mismo símbolo F, es decir:

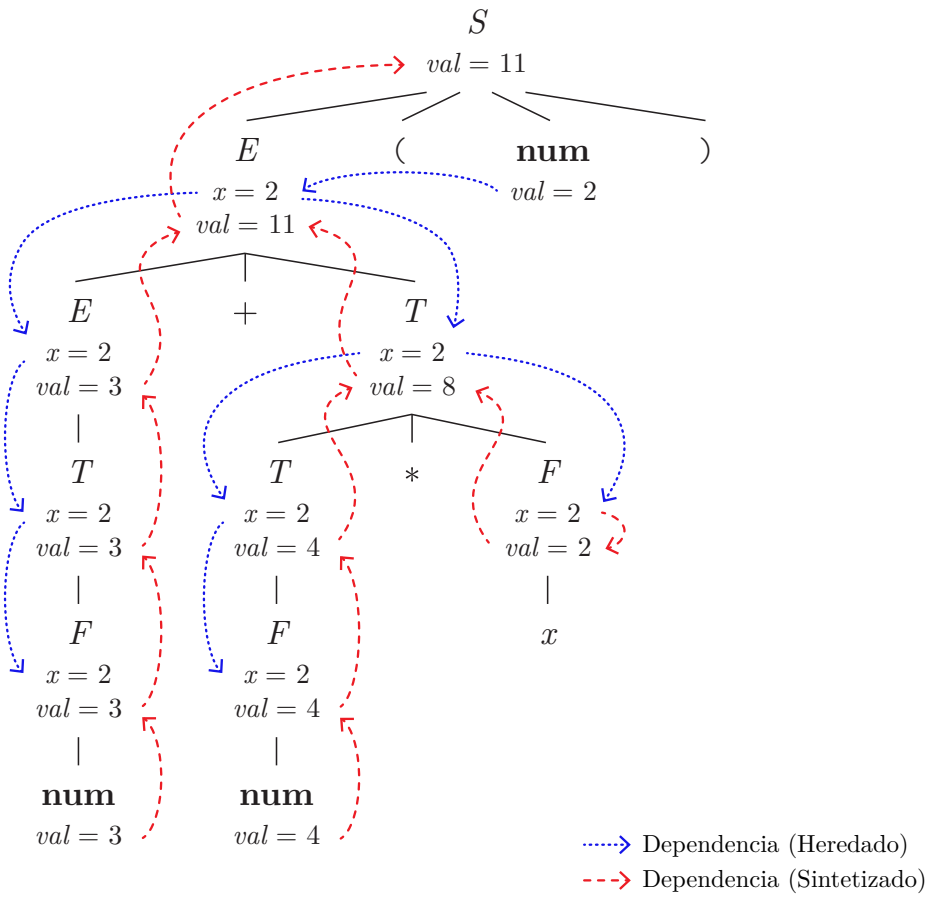
$$F.val = F.x.$$

Recordemos que a la hora de definir un atributo sintetizado, además de los atributos de los nodos hijos, podemos usar los atributos del mismo nodo en el que lo estamos definiendo, siempre y cuando lo hagamos de forma acíclica.

Agregando las reglas correspondientes a todas las producciones, resulta:

$$\begin{array}{lcll} S & \to & E\ (\ \mathbf{num}\ ) & \{\ E.x = \mathbf{num}.val\ ;\ \underline{S.val = E.val}\ \} \\ E & \to & E_1 + T & \{\ E_1.x = E.x\ ;\ T.x = E.x\ ;\underline{E.val = E_1.val + T.val}\ \} \\ & | & T & \{\ T.x = E.x\ ;\ \underline{E.val = T.val}\ \} \\ T & \to & T_1 * F & \{\ T_1.x = T.x\ ;\ F.x = T.x\ ;\underline{T.val = T_1.val * F.val}\ \} \\ & | & F & \{\ F.x = T.x\ ;\ \underline{T.val = F.val}\ \} \\ F & \to & \mathbf{num} & \{\ \underline{F.val = \mathbf{num}.val}\ \} \\ & | & x & \{\ \underline{F.val = F.x}\ \} \\ & | & (\ E\ ) & \{\ E.x = F.x\ ;\ \underline{F.val = E.val}\ \} \end{array}$$

b. Exhibimos a continuación el árbol de derivación atribuido para la cadena $3 + 4 * x\ (\ 2\ )$, indicando con flechas las dependencias entre atributos.



c. La gramática no es S-atribuida, ya que tiene atributos heredados, y no es L-atribuida, ya que la producción $S \to E$ (**num**) posee asociada la regla semántica $E.x = \textbf{num}.val$, donde el atributo heredado $E.x$ depende de un atributo de un símbolo que está ubicado a su derecha en la producción. Sin embargo, quitando esta producción, la gramática sí resulta L-atribuida. Teniendo en cuenta esto, podemos dar un orden topológico para los atributos de cualquier árbol de derivación, corroborando así que la gramática está bien definida.

   Podemos notar que todo árbol de derivación utilizará la producción problemática exactamente una vez, en la raíz del mismo. Así, basta con comenzar computando $E.x$ a partir de $\textbf{num}.val$, recorrer luego todos los atributos del subárbol generado por E de la forma habitual para gramáticas L-atribuidas, y terminar computando $S.val$ en función de $E.val$. Este orden garantiza que nunca intentaremos computar un atributo antes de otro del cual depende.

Bibliografía recomendada

Para profundizar en este tema, recomendamos consultar el capítulo 5 del "Libro del Dragón" [2]. Allí se presenta una variante de las gramáticas de atributos, las llamadas *syntax-directed definitions*, que no tienen restringidos los efectos secundarios en las reglas semánticas. Nosotros trabajaremos con la posibilidad de agregar efectos secundarios en la próxima clase.

Otra referencia útil es el ya mencionado paper de Donald Knuth[1], publicado en 1968, donde se presenta el concepto con muy buen detalle, y se propone un algoritmo para detectar circularidad

en las gramáticas de atributos a través de la construcción de un grafo de dependencias general.

Referencias

[1] Donald E. Knuth. «Semantics of context-free languages». En: *Mathematical Systems Theory* 2.2 (jun. de 1968), págs. 127-145. URL: https://sci-hub.se/https://doi.org/10.1007/BF01692511.

[2] Alfred V. Aho et al. *Compilers: Principles, Techniques, and Tools*. 2.ª ed. Addison-Wesley, 2007.

[3] Gregor V. Bochmann. «Semantic evaluation from left to right». En: *Communications of the ACM* 19.2 (feb. de 1976), págs. 55-62. DOI: 10.1145/359997.359999.

[4] Uwe Kastens. «Ordered attributed grammars». En: *Acta Informatica* 13.3 (mar. de 1980), págs. 229-256. DOI: 10.1007/bf00288644.

[5] Mehdi Jazayeri, William F. Ogden y William C. Rounds. «The intrinsically exponential complexity of the circularity problem for attribute grammars». En: *Communications of the ACM* 18.12 (dic. de 1975), págs. 697-706. DOI: 10.1145/361227.361231.

[6] Alfred V. Aho y Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Vol. 2: *Compiling*. Prentice-Hall, 1973.