

Generación de evaluadores estáticos de gramáticas de atributos bien definidas

Marcelo Arroyo, Jorge Aguirre, Nicolás Florio*

Dpto. de Computación - Fac. de Cs. Exactas, Fco-Qcas y Nat.
Universidad Nacional de Río Cuarto

Resumen

En el presente trabajo se describe una herramienta de generación estática de evaluadores de gramáticas de atributos (GA) para la familia de *GA bien definidas* (GABD), denominada **wagcc**. Las GABD imposibilitan la derivación de *árboles sintácticos* con dependencias circulares entre las instancias de los atributos. **wagcc** se basa en los conceptos desarrollados por Wu Yang en [7] para la computación estática de *planes de evaluación* con algunas mejoras como por ejemplo la prevención de la generación de planes *espúreos*, lo cual afecta significativamente al tamaño del evaluador generado y el tiempo y espacio utilizado en su generación. El desarrollo de **wagcc** confirma el resultado de Riis y Skyum [6] quienes establecieron que todas las GABD permiten el desarrollo de un evaluador estático multi-visita. Hasta el momento, no se conocen herramientas similares basadas en ésta familia de GA.

Palabras clave: Gramáticas de Atributos, Compiladores-Compiladores, Lenguajes, Gramáticas de Atributos bien definidas.

1. Introducción

Desde que D. Knuth en 1968 [4] introdujo la idea de las Gramáticas de Atributos, éstas han sido de gran interés en las ciencias de la computación y en ingeniería de software ya que permiten describir computaciones en lenguajes libres de contexto y se han utilizado ampliamente para el desarrollo de herramientas de generación de procesadores de lenguajes basados en especificaciones, conocidos generalmente como *compiladores-compiladores* o *sistemas de generación de compiladores* o *traductores*.

Una gramática de atributos es una extensión de las gramáticas libres de contexto, a las cuales se les incluyen valores asociados a los símbolos de la gramática (denominados

* {marroyo,jaguirre,nflorio}@dc.exa.unrc.edu.ar

atributos) y ecuaciones para la computación de valores de las instancias de los atributos en un árbol sintáctico derivado a partir de la gramática libre de contexto subyacente.

Las ecuaciones inducen las dependencias entre las instancias de los atributos del árbol sintáctico. El orden de evaluación debe ser consistente con las dependencias, es decir, el atributo a debe ser evaluado luego que hayan sido evaluados las instancias de los atributos de los que depende en la ecuación correspondiente. Una dependencia circular en el árbol sintáctico hará generalmente imposible la evaluación, lo que sugiere la siguiente pregunta: dada una GA, existe algún árbol sintáctico, derivado a partir de la GA, que contenga dependencias circulares?. Esta pregunta es conocido como el *problema de circularidad*, el cual se ha probado que es un problema complejo que toma tiempo exponencial en base al tamaño de la gramática[5].

Una *GABD* tiene como característica que no es posible que se generen árboles sintácticos con dependencias circulares y es la familia mas amplia (con menos restricciones) en la clasificación tradicional de GA. Las herramientas mas comunes que se utilizan están basadas en subclases de GABD como las ANCAG (absolutamente no circulares) o aún en subfamilias como las OAG (ordenadas)[2] y recientemente las EOAG (ordenadas extendidas) [9].

Para éstas familias de GA, es posible generar un único plan (orden) de evaluación para cada producción, mientras que en las GABD¹ pueden tener más de un plan asociado por producción y el evaluador deberá realizar la selección dinámicamente (lookahead behavior). Por este motivo pertenecen a las GA denominadas **multiplan**.

2. Gramáticas de Atributos

En esta sección se definen los principales conceptos a ser utilizados en las secciones siguientes.

DEFINICIÓN 2.1 Una *gramática de atributos* es una tupla $GA = \langle G, A, R \rangle$ donde G es una *gramática libre de contexto*, A es un conjunto finito de **atributos** y R es un conjunto finito de **reglas semánticas**.

$G = \langle N, T, P, S \rangle$ donde N es un conjunto de **símbolos no terminales**, T es el conjunto de **símbolos terminales**, $V = N \cup T$, P es un conjunto de pares de la forma $X \rightarrow \alpha$ denominadas **producciones**, donde $X \in V$, $\alpha \in V^*$ (se denotará a la cadena vacía como ε), $S \in N$ es el **símbolo se comienzo**.

En una GA se asocia un conjunto de atributos $A(X) = H(X) \cup S(X)$, ($H(X) \cap S(X) = \emptyset$), con cada símbolo $X \in V$. El conjunto $H(X)$ es el conjunto de atributos heredados de X y $S(X)$ es el conjunto de atributos sintetizados de X , $H(S) = \emptyset$ y $S(X) = \emptyset, \forall X \in T$.²

¹WDAG (Well Defined Attribute Grammars), por sus siglas en inglés.

²**wagcc** permite que los símbolos terminales tengan un único atributo sintetizado denominado *lexema*, el cual contiene el string asociado y es sintetizado por el analizador léxicográfico.

Una producción $p \in P$, de la forma $X_0 \rightarrow X_1 \dots X_n$, ($n \geq 0$), tiene una ocurrencia $X_i.a$ si $a \in A(X_i)$, $0 \leq i \leq n$.

Un conjunto de reglas semánticas R_p de la forma $X_i.a := f(y_1 \dots y_k)$ se asocia con una producción p con las siguientes restricciones:

1. $i = 0$ si $a \in S(X_i)$, o $1 \leq i \leq n$ si $a \in H(X_i)$.
2. cada y_j , $1 \leq j \leq k$, es un atributo que ocurre en p .
3. f es una función (denominada *función semántica*) que mapea valores de y_1, \dots, y_k al valor de $X_i.a$.

En una regla de la forma $X_i.a := f(y_1 \dots y_k)$, se dice que la ocurrencia $X_i.a$ depende de las ocurrencias y_i , $1 \leq i \leq k$.

R es el conjunto de reglas semánticas $R = \bigcup_{p \in P} R_p$.

DEFINICIÓN 2.2 Un árbol de derivación para una cadena α , generado por $G = \langle N, T, P, S \rangle$, es un árbol donde:

1. Cada nodo tiene rótulo \mathbf{X} o ε .
2. El rótulo de la raíz es \mathbf{S} .
3. Si un nodo cuyo rótulo es \mathbf{X} y tiene sus hijos rotulados X_1, \dots, X_n (de izquierda a derecha), entonces $X \rightarrow X_1 \dots X_n$ es una producción en \mathbf{P} .
4. Los rótulos de las hojas, concatenados de izquierda a derecha forman α .

DEFINICIÓN 2.3 Un árbol atribuido para una cadena α es un árbol de derivación donde cada nodo \mathbf{n} , rotulado \mathbf{X} , contiene instancias que corresponden a los atributos de \mathbf{X} . Por cada atributo $a \in A(X)$ la instancia correspondiente contenida en \mathbf{n} se denotará $\mathbf{n.a}$.

3. La jerarquía NC

Recientemente, en [7], Wu Yang propuso una nueva clasificación de GA's basadas en dependencias de los contextos inferiores (look-ahead behavior). La idea se basa en construir grafos de dependencias para los atributos que ocurren en una producción para cada posible contexto inferior. Un evaluador de esta familia de GA, deberá seleccionar el plan adecuado en base a la inspección del contexto inferior de cada símbolo.

La jerarquía NC se basa en el número de pasos descendentes tomados en cuenta por el evaluador para seleccionar un plan. Así por ejemplo, la familia $NC(0)$ es equivalente a la familia $ANCAG$ y la familia $NC(\infty)$ se corresponde a las GA bien definidas.

Para evaluar las instancias de los atributos en un árbol sintáctico, correspondientes a una instancia de la producción p , se deben tener en cuenta tres clases de dependencias:

- las dependencias directas entre las instancias que ocurren en p .
- las dependencias entre las instancias de los atributos del símbolo de la parte izquierda de p , las cuales están determinadas por su contexto superior.
- las dependencias entre las instancias de atributos de los símbolos no terminales de la parte derecha de p , las cuales están determinadas por sus contextos inferiores (las producciones aplicadas en el subárbol cuya raíz está rotulada por ese no terminal).

3.1. La familia $NC(0)$

La familia $NC(0)$ se corresponde con la familia $ANCAG$, la que está caracterizada por la siguiente definición:

DEFINICIÓN 3.1 Sea p una producción de P de la forma

$$X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 \dots X_k \alpha_k \quad (1)$$

$$IDP(p) = DP(p) \bigcup_{i=1}^k Down(X_i)$$

$$Down(X) = \bigcup_{q \in P | X = lhs(q)} IDP(q) \parallel_{A(X)}$$

donde $IDP(q) \parallel_{A(X)}$ denota la proyección del grafo $IDP(q)$ conteniendo sólo las dependencias (transitivas) de los atributos de X , siendo X el símbolo de la parte izquierda de la producción q ($lhs(q)$).

Los grafos $Down(X)$ son una *aproximación segura* al grafo de dependencias de los atributos de X en el sentido que incluyen todas las posibles dependencias en *cualquier* árbol sintáctico, pero además puede contener algunas *dependencias espúreas* (dependencias que no pueden ocurrir simultáneamente en ningún árbol sintáctico).

Los grafos $IDP(q)$ ³ son los grafos de dependencias inducidas por cada producción.

DEFINICIÓN 3.2 Una GA es $ANCAG$ o ($NC(0)$) si cada grafo $IDP(p)$, $\forall p \in P$ es no circular.

Un evaluador para GA's $NC(0)$ utiliza el orden de evaluación (orden topológico) generado a partir de los grafos $IDP(p)$.

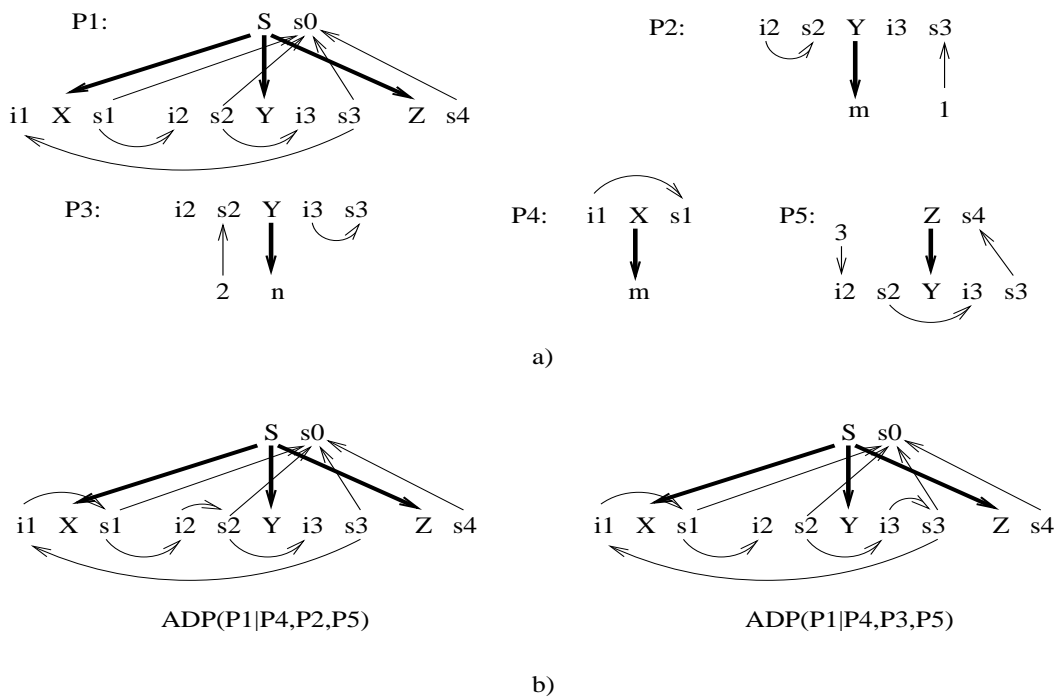


Figura 1: GA NC(1) pero no ANCAG

3.2. La familia NC(1)

En la figura 1 a) ⁴ se muestra una GA que no es ANCAG, ya que $IDP(P1)$ es circular⁵, aunque la GA no lo es. Esto se debe a que los grafos $Down(X)$ representan las posibles dependencias en cualquier árbol sintáctico, pero las producciones P2 y P3 nunca pueden aplicarse simultáneamente y ésta es la causa de la circularidad.

Si se toman en cuenta los posibles diferentes contextos inferiores sería posible diferenciar las dependencias en diferentes grafos y se podrían asociar a cada producción un conjunto de grafos posibles (uno por cada contexto inferior). Si ninguno de esos grafos es circular, la GA no es circular y un evaluador podría seguir el plan de evaluación generado a partir de cada grafo. El evaluador debería seleccionar (en tiempo de ejecución) el plan adecuado para cada producción (inspeccionando su contexto inferior).

DEFINICIÓN 3.3 Sea q una producción como en (1), y sean p_i , ($1 \leq i \leq k$) producciones cuyo lhs(p_i) es X_i ,

$$DCG_X(q) = \bigcup_{q \in P | X = lhs(q)} ADP(q | p_1, p_2, \dots, p_k) \parallel_{A(X)}$$

³Induced Dependencies Graphs

⁴Las flechas gruesas denotan las producciones y las finas las dependencias entre los atributos.

⁵ $IDP(P1) = ADP(P1|P4, P2, P5) \cup ADP(P1|P4, P3, P5)$.

$$ADP(q|p_1, p_2, \dots, p_k) = DP(q) \bigcup_{i=1}^k DCG_{X_i}(p_i)$$

Un grafo $DCG_X(q)$ se denomina el *downward characteristic graph de X en los subárboles derivados a partir de la producción q* y denotan las dependencias (transitivas) entre los atributos de X en algún subárbol derivado a partir de la aplicación de la producción q .

Un grafo $ADP(q|p_1, p_2, \dots, p_k)$ se denomina *augmented dependency graph de la producción q con los subárboles derivados a partir de la aplicación de p_1, p_2, \dots, p_k* .

DEFINICIÓN 3.4 Una GA es $NC(1)$ si cada grafo del conjunto

$$SADP(q) = \{ADP(q|p_1, p_2, \dots, p_k)\}$$

es acíclico, $\forall q \in P$.

La figura 1 b) muestra que la GA es $NC(1)$ (sólo se muestran los grafos de la producción que causaba la circularidad para las $ANCAG$).

Se debe notar que los grafos $DCG_X(q)$ son estimaciones más exactas que los grafos $Down(X)$ ya que es fácil demostrar que

$$\bigcup_{q|X=lhs(q)} DCG_X(q) \subseteq Down(X)$$

3.3. La familia $NC(m)$

Los grafos $ADP(q|\dots)$ tienen en cuenta una generación de descendentes de la producción q . Si extendemos la idea para tener en cuenta más generaciones se puede definir una nueva familia de clases de GA: las GA $NC(m)$: *noncircular AG with m-generation lookahead*.

DEFINICIÓN 3.5 Sea q una producción de la forma (1), $\tau = \langle X, m \rangle$ un árbol de profundidad m , cuya raíz está rotulada X y es un subárbol de algún árbol sintáctico derivado por la GA⁶ y sean $\tau_i = \langle X_i, m \rangle$, para $1 \leq i \leq k$,

$$DCG_X(\tau) = \bigcup ADP_{\langle m \rangle}(q|\tau_1, \tau_2, \dots, \tau_k) \parallel_{A(X)}$$

$$ADP_{\langle m \rangle}(q|\tau_1, \tau_2, \dots, \tau_k) = DP(q) \bigcup_{i=1}^k DCG_{X_i}(\tau_i)$$

DEFINICIÓN 3.6 Una GA es $NC(m)$ si cada grafo del conjunto

$$SADP_{\langle m \rangle}(q) = \{ADP_{\langle m \rangle}(q|\tau_1, \tau_2, \dots, \tau_k)\}$$

es acíclico, $\forall q \in P$ y $\forall \tau_i = \langle X_i, m \rangle$, $1 \leq i \leq k$.

⁶ $\tau = \langle X, m \rangle$ se denomina un **m-phrase tree**.

En [7] se muestra cómo una gramática $NC(m)$ puede ser transformada en $NC(m - 1)$ (y por ende en $NC(1)$). El resultado sólo tiene interés teórico, ya que el número de producciones se puede incrementar en forma exponencial.

3.4. La familia $NC(\infty)$

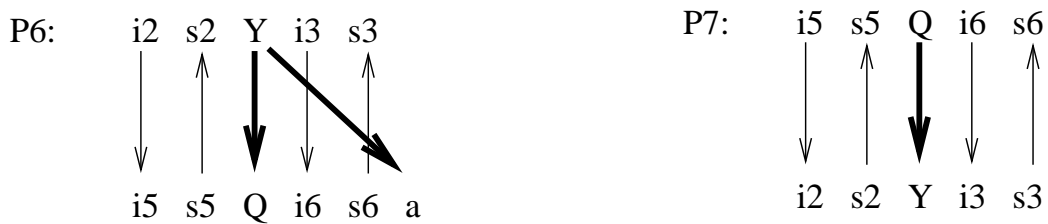


Figura 2: Producciones adicionales que hacen que la GA sea $NC(\infty)$

Existen GAs que no son $NC(m)$, para ningún m . Si a la GA de la figura 1 a) le adicionamos dos producciones como las de la figura 2, ésta última no es $NC(m)$ (para ningún m finito), ya que contiene producciones mutuamente recursivas. Un árbol sintáctico podría contener derivaciones $Y \rightarrow Q \rightarrow Y$ cualquier número de veces. Además, se puede observar que la nueva AG no es circular (es $GABD$).

Un evaluador para las GA $NC(\infty)$ necesita tener en cuenta un número *potencialmente* (no *actualmente*) infinito de descendientes. En términos de un árbol sintáctico, lo anterior significa que el evaluador necesitará “mirar” hasta las hojas para seleccionar un plan para una instancia de una producción.

Para realizar la selección el evaluador necesitará realizar un recorrido ascendente del árbol sintáctico para *marcar* los nodos con la información sobre las dependencias *actuales* transitivas en base al subárbol.

A cada no terminal X se le asocia un conjunto $\Delta(X)$ de posibles grafos de dependencias entre sus atributos. El algoritmo 1 computa todos los posibles grafos. Si algún $\delta \in \Delta(X)$ es cíclico, entonces la GA no es $NC(\infty)$.

Sean p una producción como en (1) el algoritmo *InfiniteLookAhead*, además, construye la función $\Lambda[q|\delta_1, \delta_2, \dots, \delta_k]$, la cual representa el grafo de dependencias entre los atributos de X_0 cuando se aplica la producción p y las dependencias transitivas de los símbolos X_i son δ_i , ($1 \leq i \leq k$), respectivamente.

La información denotada por Λ será utilizada por el evaluador para la selección de planes. El algoritmo 1 es básicamente el *test de circularidad* de Knuth[5] con la adición del cómputo de la función Λ y los conjuntos $\Delta(X)$.

La complejidad del algoritmo es $O(|N||P|(h!)^{l+1}h^3)$, donde h es el número máximo de atributos de los símbolos y l es el número máximo de no terminales de la parte derecha de las producciones. El análisis de complejidad se basa en que $|\Delta(X)| = O(h!)$. Dado que hay $|N|$ no terminales, la suma de todos los $|\Delta(X)|$ es $O(|N|h!)$ (el cual es

algoritmo 1 InfiniteLookAhead

```
 $\Delta(X) \leftarrow \emptyset, \forall X \in V$ 
repeat
   $change \leftarrow false$ 
  for cada  $q: X_0 \rightarrow \alpha_0 X_1 \alpha_1 \dots X_k \alpha_k$  do
    for cada  $\delta_i \in \Delta(X_i), 1 \leq i \leq k$  do
       $G \leftarrow DP(q) \cup_{i=1}^k \delta_i$ 
       $\delta \leftarrow G|_{X_0}$ 
      if  $\delta$  es circular then error("Circular AG")
       $\Lambda[q|\delta_1, \dots, \delta_k] \leftarrow \delta$ 
      if  $\delta \notin \Delta(X_0)$  then
         $changed \leftarrow true$ 
         $\Delta(X_0) \leftarrow \Delta(X_0) \cup \delta$ 
      end if
    end for
  end for
until not changed
```

el número de veces máximo que se ejecuta el ciclo *repeat*, ya que en cada iteración se debe agregar al menos un nuevo δ). la sentencia *for* más externa se ejecuta $|P|$ veces y el *for* más interno se ejecuta a lo sumo $(h!)^l$. La operación de unión $G \leftarrow DP(q) \cup_{i=1}^k \delta_i$ toma $O(l)$ ($k \leq l$) y la operación de proyección ($|_{X_0}$ tiene $O(h^3(l+1)^3)$, ya que se debe realizar la clausura transitiva.

Si bien el orden de complejidad es exponencial con respecto a l , éste valor generalmente no supera a 4 en la práctica (una GA con numerosos no terminales en la parte derecha de las producciones es muy inusual, ya que sería poco legible e indicaría que no se ha modularizado como corresponde). Si bien se pueden llegar a realizar un gran número de operaciones y las funciones computadas pueden tener un número considerable de entradas, el problema es perfectamente tratable con la potencia computacional que hoy brinda cualquier PC de escritorio.

3.4.1. Generación de planes de evaluación

Una vez que se aplica el algoritmo 1 y se ha detectado que la GA es $NC(\infty)$, se generan los planes de evaluación asociados a cada producción.

El algoritmo 2 realiza la generación de planes y es una modificación del algoritmo propuesto en [7]. Una de las modificaciones es la lista de trabajo WL , que contiene pares de símbolos y ordenes en lugar de sólo ordenes como propone Wu Yang y el uso de un *filtro* para evitar la generación de planes espúreos que pueden ser generados (aunque algunos posteriormente descartados) inútilmente por el algoritmo original. El filtro es la función (boolean) *applicable*, la cual se define como la expresión lógica:

$$applicable(q, \omega, \delta_1, \dots, \delta_k) \equiv \Lambda[q|\delta_1, \delta_2, \dots, \delta_k] = \delta'_i$$

algoritmo 2 ComputePlans

$\omega_0 \leftarrow$ cualquier orden de $A(S)$
 $WL \leftarrow \{(S, \omega_0)\}$
repeat
 $(X_0, \omega) \leftarrow$ un elemento de WL ; $WL \leftarrow WL - \{(X_0, \omega)\}$
 sea $q : X_0 \rightarrow \alpha_0 X_1 \alpha_1 \dots X_k \alpha_k$ y *applicable*($q, \omega, \delta_1, \dots, \delta_k$)
 for cada $ADP(q|\delta_1, \dots, \delta_k) \in SADP(q)$ **do**
 $\psi \leftarrow TSort(ADP(q|\delta_1, \dots, \delta_k) \cup \omega)$
 $\Gamma[q, \omega, \delta_1, \dots, \delta_k] \leftarrow \psi$
 for cada $X_i \in rhs(q)$ **do**
 $\omega_i \leftarrow \psi|_{A(X_i)}$
 $\Theta[q, \omega, i, \delta_1, \dots, \delta_k] \leftarrow \omega_i$
 if $\Pi(X_i, \omega_i) = \text{undefined}$ **then**
 $\Pi(X_i, \omega_i) \leftarrow \text{defined}$; $WL \leftarrow WL \cup \{(X_i, \omega_i)\}$
 end if
 end for
 end for
until $WL = \emptyset$

donde δ'_i es tal que $\omega = \Theta[q, \omega', i, \dots, \delta'_i, \dots]$

El uso de la función *applicable* reduce significativamente el número de planes a generar evitando la generación de planes espúreos. En [7] plantea descartar los planes espúreos si es que éstos son circulares (ya que en ésta etapa la AG obviamente no es circular). El uso de ésta función reduce significativamente el caso promedio de complejidad del algoritmo.

Para la gramática de la figura 2 el algoritmo 1 computa⁷:

$\Delta(S) = \{\delta_1 = [s0]\}$, $\Delta(Y) = \{\delta_2 = [i2 \rightarrow s2 \ s3 \ s4], \delta_3 = [i2 \ s2 \ i3 \rightarrow s3]\}$, $\Delta(X) = \{\delta_4 = [i1 \rightarrow s1]\}$,
 $\Delta(Z) = \{\delta_5 = [s4]\}$, $\Delta(Q) = \{\delta_6 = [i5 \rightarrow s5 \ i6 \ s6], \delta_7 = [i5 \ s5 \ i6 \rightarrow s6]\}$.
 $\Lambda[P1] = [s0]$, $\Lambda[P2] = [i2 \rightarrow s2 \ i3 \ s3]$, $\Lambda[P3] = [i2 \ s2 \ i3 \rightarrow s3]$, $\Lambda[P4] = [i1 \rightarrow s1]$, $\Lambda[P5|\delta_2] = \Lambda[P5|\delta_3] = [s4]$,
 $\Lambda[P6|\delta_6] = [i2 \rightarrow s2 \ i3 \ s3]$, $\Lambda[P6|\delta_7] = [i2 \ s2 \ i3 \rightarrow s3]$, $\Lambda[P7|\delta_2] = [i5 \rightarrow s5 \ i6 \ s6]$, $\Lambda[P7|\delta_3] = [i5 \ s5 \ i6 \rightarrow s6]$.

y el algoritmo 2 computa:

$\Gamma[P1, \langle s0 \rangle, \delta_4, \delta_2, \delta_5] = \langle s3, i1, s1, i2, s2, i3, s4, s0 \rangle$, $\Gamma[P1, \langle s0 \rangle, \delta_4, \delta_3, \delta_5] = \langle s2, i3, s3, i1, s1, i2, s4, s0 \rangle$,
 $\Gamma[P2, \langle s3, i2, s2, i3 \rangle] = \langle s3, i2, s2, i3 \rangle$, $\Gamma[P3, \langle s2, i3, s3, i2 \rangle] = \langle s2, i3, s3, i2 \rangle$, $\Gamma[P4, \langle i1, s1 \rangle] = \langle i1, s1 \rangle$,
 $\Gamma[P5, \langle s4 \rangle, \delta_2] = \langle s3, i2, s2, i3, s4 \rangle$, $\Gamma[P5, \langle s4 \rangle, \delta_3] = \langle s2, i3, s3, i2, s4 \rangle$,
 $\Gamma[P6, \langle s3, i2, s2, i3 \rangle, \delta_6] = \langle s6, s3, i2, i5, s5, s2, i3, i6 \rangle$, $\Gamma[P6, \langle s2, i3, s3, i2 \rangle, \delta_7] = \langle s5, s2, i3, i6, s6, s3, i2, i5 \rangle$,
 $\Gamma[P7, \langle s6, i5, s5, i6 \rangle, \delta_2] = \langle s3, s6, i5, i2, s2, s5, i6, i3 \rangle$, $\Gamma[P7, \langle s5, i6, s6, i5 \rangle, \delta_3] = \langle s2, s5, i6, i3, s3, s6, i5, i2 \rangle$.

3.4.2. Un evaluador para la familia $NC(\infty)$

El procedimiento *eval*(T) del algoritmo 3 evalúa los atributos de un árbol sintáctico T . La primer fase (*mark* y *select* selecciona el plan a utilizar en cada nodo del árbol utilizando la información computada estáticamente por los algoritmos anteriores y luego

⁷ $[i2 \rightarrow s2 \dots]$ denota un grafo de dependencias.

algoritmo 3 eval(T): el evaluador de atributos

```
procedure mark( $n$ )
for  $i=1$  to  $k$  do
    mark( $n.child[i]$ )
end for
 $n.mark \leftarrow \Lambda[n.q|n.child[1].mark, \dots, child[k].mark]$ 

procedure select( $n, \omega$ )
 $n.plan \leftarrow \Gamma[n.q, \omega, m_1.mark, \dots, m_k.mark]$ 
for  $i=1$  to  $k$  do
    select_plan( $n.child[i]$ ,  $\Theta[n.q, \omega, i, n.child[1].mark, \dots, n.child[k].mark]$ )
end for

procedure vs_eval( $n$ )
 $r \leftarrow n$ 
while vs[r.plan][r.op]  $\neq$  LEAVE do
    case vs[n.plan][n.op++] do
        VISIT( $i$ ):  $n \leftarrow n.child[i]$ 
        COMPUTE( $j$ ): compute( $j$ )
        LEAVE:  $n \leftarrow n.parent$ 
    end case
end while

procedure eval( $T$ )
mark( $T$ ); select( $T, \mu$ ); vs_eval( $T$ )
```

invoca al evaluador basado en secuencias de visitas.

En realidad los planes de evaluación no se almacenan como una secuencia de atributos sino que se transforman en secuencias de operaciones denominadas *secuencias de visita*. Existen tres operaciones: **visit**(i) que significa visitar al hijo i -ésimo, **compute**(j) que es computar la función semántica j y **leave** que es visitar al nodo padre. Cada secuencia de visita debe finalizar con una operación *leave*.

La transformación de un plan en secuencias de visita es simple: cada subsecuencia de atributos heredados del símbolo de la parte izquierda de la producción se reemplaza por una operación *leave*, cada subsecuencia de atributos sintetizados del símbolo X_i de la parte derecha se reemplaza con una operación *visit*(i) y cada ocurrencia de un atributo sintetizado del símbolo de la parte izquierda o un atributo heredado de un símbolo de la parte derecha, se reemplaza por una operación *compute*(j), donde j es en número (asignado por el generador) de la función semántica que define a dicho atributo.

4. wagcc

El generador **wagcc** toma como input una definición de una gramática de atributos y genera una especificación *lex-yacc* la cual sintetiza un árbol sintáctico T y luego invoca a *eval*(T). La figura 4 muestra gráficamente la generación del evaluador. El producto final es un programa C que contiene el analizador léxico (función *yylex*(\cdot)), el parser

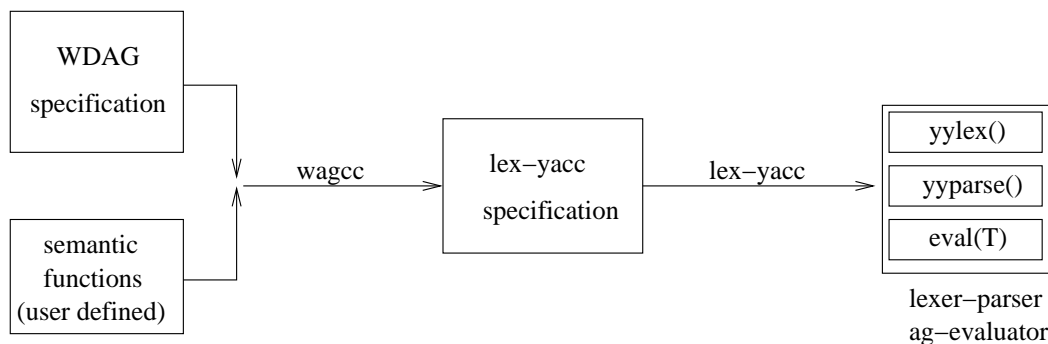


Figura 3: Esquema de uso de wagcc

(*yyparse()*) y la función de evaluación de los atributos (*eval()*).

La gramática subyacente debería ser *LALR(1)*, ya que el parser será generado por *yacc*. La especificación *lex-yacc* generada incluye los algoritmos descritos y las funciones Δ , Λ , Γ y Θ (representadas como arreglos de enteros). Las secuencias de visita generadas están representadas por un arreglo *vs[plan][[]]* donde cada fila representa un plan el cual consiste de un vector de operaciones (codificadas como enteros: *i* para *VISIT(i)*, $-j$ para *COMPUTE(j)* y 0 para *LEAVE*). Además se genera la función *compute(j)* que contiene las invocaciones (en una sentencia **switch(j)**) a las funciones semánticas provistas por el usuario.

Se ha elegido implementar el evaluador basado en secuencias de visitas como un autómata porque es más compacto que las técnicas alternativas propuestas en [3], la cual se basa en la generación de procedimientos recursivos pero no es la técnica más adecuada para un evaluador *multi-plan*.

Los nodos del árbol sintetizado contienen los siguientes campos: (*prod*) (contiene el número de producción que se aplicó en el nodo), *childrens* (contiene el número hijos), *child[]* (vector de punteros a nodos hijos), *op* (número de operación de la secuencia de visita correspondiente al nodo), *plan* (plan seleccionado), *mark* (usado para la selección) y además contiene los campos correspondientes a los atributos del símbolo.

5. Conclusiones y futuras extensiones

Se ha presentado una herramienta de generación de procesadores de lenguajes (generalmente conocidos como compiladores de compiladores) cuyo lenguaje de especificación está basado en gramáticas de atributos bien definidas. Esta familia de GA es la mayor familia para la que es posible generar evaluadores estáticos. Los evaluadores generados son compactos ya que toda la información generada se codifica en arreglos de enteros y no es necesario generar grafos de dependencias como lo requieren los evaluadores dinámicos.

Se ha refinado el algoritmo de computación de planes de evaluación de [7], previniendo la generación de planes espúreos. Hasta el momento no se conoce otra herramienta basada en los conceptos desarrollados en este trabajo.

Es necesario realizar mayores experimentos con gramáticas mas reales (como las de algún lenguaje de programación) para analizar los requerimientos de espacio utilizado por las funciones descriptas y el número de planes generador por producción.

En éstos momentos se está desarrollando una versión para C++ con un diseño orientado a objetos el cual permitirá desarrollar versiones para otros lenguajes comúnmente usados como por ejemplo Java.

Referencias

- [1] M. Arroyo, J. Aguirre, N. Florio. 2002. *NCEval, un generador de evaluadores concurrentes para gramáticas de atributos no circulares*. CACIC 2002. Buenos Aires.
- [2] U. Kastens. 1980. *Ordered Attribute Grammars*. Acta Informatica, vol. 13, pag: 229-256.
- [3] U. Kastens. 1991, *Implementation of Visit-Oriented Attribute Evaluators*. Proc. International Summer School SAGA, vol. 545, pág: 114-139.
- [4] D. Knuth. 1968. *Semantics of context free languages*. Math Systems Theory 2, June 2. Pag: 127-145.
- [5] D. Knuth. 1971. *Semantics of context free languages*. Math Systems Theory Vol 5, no 1. Pag: 95-96. (correction)
- [6] H. Riis, S. Skyum. 1981. *K-Visit Attribute Grammars*. Math Systems Theory, vol. 15, pág: 17-28.
- [7] Wu Yang. March 2002. *A Classification of Non-Circular Attribute Grammars Based on the Look-Ahead Behavior*. IEEE Transactions on Software Engineering, vol. 28, No 3. Pag: 210-227.
- [8] Wu Yang. 1999. *A Finest Partitioning Algorithm for Attribute Grammars*. Second Workshop on Attribute Grammars and their Applications - WAGA 99.
- [9] Wu Yang. 1998. *SSCC: A Software Tool Based on Extended Ordered Attribute Grammars*. Proc. National Science Council (Rep. of China), Parte A, Physical Science and Engineering, vol. 23, no. 1, pág: 85-99.