Gramáticas de atributos *NC(1)* condicionales

Marcelo Arroyo, Jorge Aguire y Nicolás Florio
{marroyo,jaguirre,nflorio}@dc.exa.unrc.edu.ar*
Departamento de Computación
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Universidad Nacional de Río Cuarto - Argentina

Resumen

En este trabajo se presenta una extensión a las gramáticas de atributos (GA) no circulares NC(1) introducidas por Yang [11]. Se introducen las *gramáticas de atributos NC(1) condicionales* al estilo de las gramáticas de atributos condicionales introducido por Boyland en [1]. Las GA NC(1) contienen a la familia ANCAG[1] y permiten generar estáticamente planes de evaluación. Se hace una introducción a las GA NC(1) y se presentan las GA condicionales. Se definen las *GA NC(1) condicionales*, como así también las *secuencias de visita condicionales*, utilizadas para la evaluación. Se presenta un algoritmo para la generación de *planes de evaluación condicionales* y se describe una estrategia de generación de secuencias de visita condicionales a partir de los planes de evaluación computados. Se propone una implementación de las secuencias de visita condicionales basada en una simple adaptación del método propuesto por Kastens en [3].

Palabras clave: gramáticas de atributos condicionales, compiladores, procesadores de lenguajes, planes de evaluación.

1 Introducción

Desde su introducción en 1968[7], las gramáticas de atributos (*GA*) han sido motivo de gran interés tanto en investigación como en la práctica para la generación de procesadores de lenguajes (compiladores y compiladores-compiladores). Las *GA* son un simple y poderoso formalismo para especificar computaciones sobre lenguajes libres de contexto y se han utilizado para describir la semántica de lenguajes de programación. Uno de los principales intereses es el desarrollo de algoritmos de evaluación de las instancias de los atributos en un árbol sintáctico. En una gramática de atributos, hay ecuaciones de atribución que especifican las reglas para computar los valores de las instancias en un árbol sintáctico derivado a partir de una gramática de atributos. Estas ecuaciones de atribución definen dependencias entre las instancias de los atributos de un árbol sintáctico. El orden de evaluación de estas instancias debe ser consistente con sus dependencias, de forma tal que una instancia de un atributo a no pueda ser evaluada hasta que todas las instancias de atributos

*El presente trabajo ha sido realizado en el marco de proyectos subsidiados por SECyT de la UNRC y por la Agencia Córdoba Ciencia.

[1] Gramáticas de atributos absolutamente no circulares.

usadas en las ecuaciones que definen a a hayan sido evaluadas. Si hay una dependencia circular, es decir, una instancia de atributo que dependa transitivamente de sí misma, el árbol sintactactico no podrá ser evaluado. Por esta razón, es importante saber si hay dependencias circulares entre las instancias de los atributos de un árbol sintáctico derivado de una gramática de atributos. Este problema es conocido como *el problema de circularidad de gramáticas de atributos*, el cual constituye un problema NP[5]. Hay varias subclases de las gramáticas bien formadas para las cuales hay algoritmos de decisión de tiempo polinomial.

En este trabajo presentamos un generador de evaluadores de gramáticas de atributos condicionales para la clase de gramaticas *NC(1)*[2]. La principal característica de las *GA NC(1)* es que permiten generar evaluadores basados en *secuencias de visitas*, por lo cual pueden ser vistas como como una generalización de las *ordered attribute grammars (OAG)* definidas por Kastens y las *l-ordered GA*[3]. La diferencia radica en que en una GA NC(1) se pueden asociar más de un plan de evaluación con cada producción. Un plan de evaluación consiste de una secuencia de instrucciones de tres tipos: *eval*, *visit* y *leave*. En cada visita a un nodo se pueden evaluar algunos atributos del nodo, y en cada visita al padre(*leave*) se computan algunos atributos heredados del nodo y en cada visita a alguno de sus hijos (*visit*) se computan algunos atributos sintetizados.

Las *GA* tradicionales permiten una sola ecuacion de atribución para ser asignada a cada atributo de una produccion. Todos los sistemas de gramáticas de atributos tradicionales permiten expresiones condicionales como ecuaciones de atribución. Estas expresiones condicionales son funciones de tres parámetros, donde el valor lógico del primer parámetro selecciona uno de los otros dos. La definición formal del condicional no tiene en cuenta su naturaleza no estricta, por lo tanto una GA puede ser circular (según la definición tradicional) aunque nunca puedan ocurrir dependencias cíclicas en un árbol sintáctico.

La figura 1 muestra una gramática de atributos tradicional (circular).

---

p_0: $S \rightarrow A$
attribution
$A.i := 0$
$S.val := A.s$
end
p_1: $A \rightarrow A\ A$
attribution
$A_0.s := if\ (odd(A_0.i)\ then\ A_1.s\ else\ A_2.s$
$A_1.i := if\ (odd(A_0.i)\ then\ A_2.s\ else\ A_0.i + 1$
$A_2.i := if\ (odd(A_0.i)\ then\ A_0.i + 1\ else\ A_1.s$
end
p_2: $A \rightarrow \lambda$
attribution
$A.s := A.i + 1$
end

Figura 1: Una gramática de atributos circular (tradicional)

---

[2]No Circulares basadas en informacion de sus descendientes inmediatos.
[3]Estas ag permiten que a cada producción se pueda asociar una única secuencia de evaluación.

Las *GA* condicionales introducidas por Boyland[1], son una extensión de las *GA* tradicionales. Estas definen una estructura condicional de la forma ***if*** *cond* ***then*** *eq1; eq2;...eqn* ***else*** *eq'1; eq'2;...eq'n* ***endif.***

Boyland redefine las dependencias entre atributos teniendo en cuenta los condicionales y así logra extender el poder expresivo. La figura 2 muestra una gramática de atributos condicional (no circular, según Boyland) equivalente a la de la figura 1 .

---

p_0: $S \rightarrow A$
attribution
$A.i := 0$
$S.val := A.s$
end
p_1: $A \rightarrow A\ A$
attribution
if c: odd$(A_0.i)$ *then*
$A_0.s := A_1.s$
$A_1.i := A_2.s$
$A_2.i := A_0.i + 1$
else
$A_0.s := A_2.s$
$A_1.i := A_0.i + 1$
$A_2.i := A_1.s$
endif
end
p_2: $A \rightarrow \lambda$
attribution
$A.s := A.i + 1$
end

Figura 2: Una gramática de atributos condicional (no circular)

---

2 Definiciones

2.1 Gramáticas de atributos clásicas

Una grámatica algebraica libre de contexto es una tupla *(N,T,S,P)* donde N es el conjunto de no terminales; T es el conjunto de terminales ($N \cap T = \oslash$); $S \in N$ es el símbolo distinguido o de comienzo; y P es el conjunto de *producciones*. Sea $\sum \equiv N \cap T$ el conjunto de *símbolos* de la grámatica. Cada producción $p \in P$ tiene la forma $p : X_0^p \rightarrow X_1^p, \ldots, X_{n_p}^p$ donde $X_0^p \in N$, $X_i^p \in \sum$, *con* $1 \leq i \leq n_p$, $n_p \geq 0$. Sea $X^p \equiv \{X_i^p | \, 0 \leq i \leq n_p\}$, el conjunto de *ocurrencias de símbolos* de la producción P.

Una gramática de atributos es una tupla *(G,S,I,R)* donde G es una gramática algebraica libre de contexto, y para cada $X \in \sum$, *S(X)* e *I(X)* son respectivamente los conjuntos de *atributos sintetizados* y *atributos heredados* definidos para el simbolo X (con $S(X) \cap I(X) = \oslash$). Por conveniencia, asumimos que cada símbolo no terminal no posee atributos heredados y sólo un atributo sintetizado, *valor* $S(X) = I(X) = \{valor\}$ para todo $X \in T$. Sea $A(X) \equiv S(X) \cup I(X)$ el conjunto de atributos de X; usaremos $S(X_i^p)$, $I(X_i^p)$ y $A(X_i^p)$

para referirnos los conjutos de atributos asociados con la ocurrencia del símbolo X_i^p. Una ocurrencia de un atributo tiene la forma $X_i.a$ donde $a \in A(X_i)$. Para cada producción p de la forma $p: X_0^p \rightarrow X_1^p, \ldots, X_{n_p}^p$, R^p es el conjunto de reglas de atribución que relacionan las ocurrencias de los atributos. Cada regla r tiene la siguiente forma: $X_{i_0}^p.a_0 = f(X_{i_1}^p.a_1, \ldots, X_{i_n}^p.a_k)$ donde $k \geq 0$, donde para cada $0 \leq j \leq k$, $X_{i_j}^p \in X^p$, para cada $a_j \in A(X_{i_j}^p)$, y donde f , si existe, es una función estricta. Sea $DO(r) \equiv \{X_{i_0}^p\}$ las *ocurrencias definidas de r* (llamadas tanbien *ocurrencias de salida*). Sea $UO(r) \equiv \{X_{i_0}^p.a_j | \, 0 < j \leq k\}$ las *ocurrencias usadas de r* (llamadas también *ocurrencias de entrada*).

Las reglas de atribución deben ser bien formadas. Las ocurrencias definidas deben ser un atributo sintetizado de la parte derecha o un atributo heredado de la parte izquierda: $DO(X) \subseteq DO(X)^p$, donde $DO(X)^p$ es el conjunto de todas las ocurrencias de atributos para la producción p, definidos de la siguiente forma: $DO^p \equiv \{X_0^p.a | \, a \in S(X_0^p)\} \cup \{X_i^p.a | \, a \in I(X_i^p),\ 0 < i \leq n_P\}$.

Para cada ocurencia definida debe haber sólo una definición: $\bigcup_{r \in R^p} DO(r) = DO^p$ donde $DO(r) \cap DO(r') = \oslash$, $r \in R^p$, $r \neq r'$.

Sólo los atributos heredados de la parte izquierda y los atributos sintetizados de la parte derecha pueden ser utilizados para para computar el valor de un atributo: $UO(r) \subseteq UO^p$, donde UO^p es el conjunto de toda las ocurrencias de atributos que pueden ser usados: $UO^p \equiv \{X_0^p.a | \, a \in I(X_0^p)\} \cap \{X_i^p.a | \, a \in S(X_i^p),\ 1 < i \leq n_p\}\}$.

Una *ocurrencia definida* $X_i^p.a \in DO^p$ depende de una *referencia usada* $X_j^p.b \in UO^p$ (denotado $X_i^p.b \rightarrow X_j^p.a$) si la regla $r \in R^p$ que define $X_i^p.a$ ($\{X_i^p.a\} = DO(r)$) usa X_j^p ($X_j^p.b \in UO(r)$). D^p denota el *grafo de dependencia* de una producción; los vértices V_{D^p} son las ocurrencias de atributos; y los arcos E_{D^p} son las dependencias entre ocurrencias de atributos: $D^p \equiv (V_{D^p}, E_{D^p})$; $V_{D^p} \equiv DO^p \cup UO^p$; $E_{D^p} \equiv \{X_j^p.b \rightarrow X_j^p.a | \, \exists \in R^p,\ \{X_i^p.a\} = DO(r),\ X_J^p.b \in UO(r)\}$.

Para todo árbol sintáctico de una gramática de atributos, se puede obtener su grafo de dependencias a partir de la unión de los grafos de dependencias de sus producciones. Una gramática de atributos es *circular* si y sólo si existe un árbol sintáctico derivado de su gramática libre de contexto cuyo grafo de dependencias tenga un ciclo.

2.2 Gramáticas de atributos NC(1)

Las dependencias entre las instancias de atributos en un árbol sintáctico pueden inferirse desde las dependencias entre ocurrencias de las producciones analizando su atribución (denominadas *dependencias directas*). Una vez conocidas las dependencias directas, se pueden calcular las dependencias transitivas entre los atributos de un símbolo. Estas se pueden calcular en base a su contexto (superior)[4] o en base a la estructura derivada (hijos) a partir de este símbolo (contexto inferior).

Las dependencias transitivas entre los atributos de un símbolo X debido a su contexto se denominan *dependencias transitivas ascendentes*. Las dependencias transitivas entre los atributos de un símbolo X debido a su estructura derivada se denominan *dependencias transitivas descendentes*.

DEFINICIÓN 2.1: Sea X^p el no terminal de la parte izquierda de una producción p. El *grafo caráctristico de dependencias transitivas descendentes de* X^p en base a los subárboles derivados a partir de la poducción p , denotado *DCG(p)*, es un grafo: $DCG(p) = (V, E)$ donde $V = A_x$ y

$$E = \{X^p.a \rightarrow X^p.b | \, \textit{existe una dependencia transitiva desde } X^p.b \textit{ a } X^p.a$$

[4]Denominamos contexto de un símbolo X al subárbol T tal que X es una hoja de T.

en algún árbol derivado apartir de X^p *aplicando la producción* $p\}$

.

Formalmente: Sea $p: X_0^p \to X_1^p, \ldots, X_{n_p}^p$ una producción, sea p_i una producción cuya parte izquierda es X_i con $0 \leq i \leq k$:

$$DCG_x(p) = \cup\{\prod(DPA(p|\, p_1, \ldots, p_k), A_x)\, tal\, que\, p_1, \ldots, p_k\}$$

donde $X_1, \ldots, X_n$ son los no terminales de la parte izquierda de $p_1, \ldots, p_k$ respectivamente.

$$DPA(q|\, p_1, \ldots, p_k) = D^p \cup DCG_{x_1}(p_1) \cup \ldots \cup DCG_{x_k}(p_k)$$

$\prod(DPA(p|\, p_1, \ldots, p_k), A_x)$ =proyección del grafo $DPA(q|\, p_1, \ldots, p_k)$ respecto a los atributos de X.

El grafo $DPA(p|\, p_1, \ldots, p_k)$ se denomina grafo de dependencias aumentado de la producción p con respecto de los subárboles derivados por aplicar $p_1, \ldots, p_k$. Sea $SDAPA(p)$ el conjunto de todos los $DPA(p|\, p_1, \ldots, p_k)$.

DEFINICIÓN 2.2: Una *GA* es *NC(1)* sii para toda producción p de *GA* si $DPA(p|\, p_1, \ldots, p_k) \in SDAPA(p)$ es acíclico.

Una propiedad fundamental de las *GA NC(1)* es que son *GA bien definidas*, además conforman una familia más grande que *las estrictamente no circulares (ANCAG)*[5].

Si una GA es NC(1), de acuerdo con su definición, cada grafo $DPA(p|\, p_1, \ldots, p_k)$ es acíclico, por lo cual cualquier orden topológico de $DPA(p|\, p_1, \ldots, p_k)$ podría ser una posible secuencia de evaluación para los atributos de p[6]. Se debe notar que cada producción puede tener asociados varios grafos $DPA(p|\ldots)$ (cada uno determina un orden total de evaluación de los atributos que ocurren en p), por lo que las familias de las GA NC(1) se denominan también *multiplan*.

2.3 Gramáticas de atributos condicionales

Una gramática de atributos condicional es una tupla (G, S, I, C, R), donde G, S e I tienen la misma forma que en una gramática de atributos tradicional. Para cada produción p, C^p es un conjunto de condiciones usadas en las regla de atribución R^p. Cada regla r es o bien una *regla primaria*, de la forma clásica, o una *regla condicional* con la siguiente forma: $if\, c_r\ then\ R_r^T\ else\ R_r^F\ endif$ donde R_r^T, $R_r^F \subset R^p$ son conjuntos de reglas y $c_r \in C^p$ es una expresión rotulada de la forma $l_r : f(X_{i_1}^p.a_1, \ldots, X_{i_k}^p.a_k)$ para alguna función predicativa f. Las reglas condicionales pueden estar anidadas. Nos vamos a referir a la expresión lógica c_r como a la *condición* y a los miembros de R_r^T y R_r^F como *reglas de atribución dependientes*. Las reglas que no están anidadas en ningun condicional se llaman *reglas de alto nivel* y se denotan R^∞. El conjunto de *reglas primarias* se denotan R^0 y el conjunto de *reglas condicionales* $R - R^0$ se denota R^c.

Las condiciones son definidas en función de las *ocurrencias definidas* y las *ocurrencias usadas*. Las definiciones de DO^p y UO^p deben ser aumentadas para considerar lascondiciones:

$$DO^p \equiv C^p \cup \{X_0^p.a|\, a \in S(X_0^p)\} \cup \{X_i^p.a|\, a \in I(X_I^p),\ 0 < i \leq n_p\}$$

$$UO^p \equiv C^p \cup \{X_0^p.a|\, a \in I(X_0^p)\} \cup \{X_i^p.a|\, a \in S(X_I^p),\ 0 < i \leq n_p\}$$

[5]Gramáticas de atributos a partir de las cuales no pueden derivarse árboles sintácticos con dependencias circulares.

[6]Una *GA*es *ANCAG* o *NC(1)* si la dependencias directas de los atributos de cada producción no son circulares.

Las definiciones de $DO(r)$ y $UO(r)$ para las reglas primarias se extienden para aplicarlas a las reglas condicionales. Para una regla condicional r , $DO(r)$ se define de forma tal que incluya las condiciones y todos los atributos definidos en las ramas del condicional, mientras que $UO(r)$ contiene sólo la condición y las ocurrencias de atributos usadas en las condiciones:

$$DO(r) \equiv \{\{c_r\} \bigcup_{r' \in R_r^T \cup R_r^F} DO(r'), \quad r \in R^c\}$$

$$UO(r) \equiv \{\{c_r\} \cup \{X_{i_j}^p.a_j | \, 1 \leq j \leq k\}\}, \quad r \in R^c$$

donde c_r es de la forma $l_r : f(X_{i_1}^p.a_1, \ldots, X_{i_k}^p.a_k)$.

Como en la gramáticas de atributos tradicionales, todas las ocurrencias definidas deben estar en DO^p. Todas las referencias usadas deben estar en UO^p:

$$DO(r) \subseteq DO^p, \; r \in R^p$$

$$UO(r) \subseteq UO^p, \; r \in R^p$$

Una regla condicional es *bien formada* sólo si R_r^T y R_r^F definen el mismo conjunto de ocurrencias de atributos y una sola vez cada uno:

$$\left(\bigcup_{r' \in R_r^T} DO(r') \right) - C = \left(\bigcup_{r' \in R_r^F} DO(r') \right) - C, \quad r \in R^c$$

$$\bigwedge_{r \neq r' \in R} DO(r') \cap DO(r'') = \oslash, \; R \in \{R_r^T, R_r^F\}, \; r \in R^c$$

Cada ocurrencia en DO^p debe ser definida por exactamente una regla de alto nivel:

$$\bigcup_{r \in R^p \cap R^\infty} DO(r) = DO^p, \qquad \bigcap_{r \neq r' \in R} DO(r') \cap DO(r') = \oslash$$

Cada regla de atribución $r \in R^p$ tiene cero o más condiciones. Definimos una *restricción* para cada regla, denotada $T(r)$, que expresa cuando una regla es válida: $T(r) \in T^p$, donde T^p es el conjunto de toda las posibles restricciones, y donde cada restricción es un conjunto de *condiciones*, cada una de las cuales es un par (c, b), $c \in C$, $b \in \{T, F\}$:

$$T^p \equiv \wp(C^p \times \{T, F\})$$

Las restricciones $T(r)$ se definen recursivamente. Las reglas de primer nivel no tienen restricciones: $T(r) = \oslash$, $r \in R^\infty$. Las reglas dependientes tienen las restricciones de las reglas condicionales que la contienen junto con las restricciones de la condición. La restricción para la rama verdadera restinge la condición a ser T; la restrición para la rama falsa la restringe a ser F:

$$T(r') \equiv T(r) \cup \{(c_r, T)\},\ r \in R_r^T,\ r \in R^c$$

$$T(r') \equiv T(r) \cup \{(c_r, F)\},\ r \in R_r^F,\ r \in R^c$$

Una restricción t es *inconsistente* si contiene (c, T) y (c, F) para la misma condición c. En caso contrario se denomina *consistente*. Una restricción t está *correctamente estructurada* si es consistente y, para una condición restringida c (es decir, $(c, b) \in t$, para algún b), la restricción para esta regla está incluida en t ($T(c_r) \subset t$, donde r_c es la regla condicional que chequea a t). Intituivamente, las restricciones correctamente estructuradas nos aseguran que una restricción no contiene una condición irrelevante. Dos restricciones correctamente estructuradas t y t' son compatibles ($t \sim t'$), si $t \cup t'$ es consistente. Una restricción correctamente estructurada t es *maximal* si ninguna extensión $t' \supset t$ es correctamente estructurada. Por lo cual, para cada restricción correctamente estructurada t, existe una restricción maximal t' que la incluye ($t' \supseteq t$).

El *grafo de dependencias condicionales* (D'^p) para una producción p se define sobre las ocurrencias de atributos y rotulando los arcos con restriciones:

$$D'^p \equiv (V_{D'^p}, E_{D'^p}) \tag{1}$$

$$V_{D'^p} \equiv UO^p \cup DO^p$$

$$E_{D'^p} \equiv UO^p \times T^p \times DO^p$$

$$E_{D'} \equiv \{u \overset{t}{\rightarrow} d \mid u \in UO(r),\ t = T(r),\ d \in DO(r),\ u \neq d,\ r \in R\}$$

Un conjunto de arcos $\{u_i \overset{t_i}{\rightarrow} d_i\}$ es *válido* si la unión de las restriciones, $\cup_i t_i$, es correctamente estructurada. Decimos que un camino $V_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} V_n$, con $n > 0$, es un *ciclo válido* si $t_1 \cup t_2 \cup \ldots \cup t_n$ es correctamente estructurado.

Una gramática de atributos condicional es *circular* si alguno de los grafos de dependencias obtenidos de algún árbol sintáctico tiene un ciclo válido.

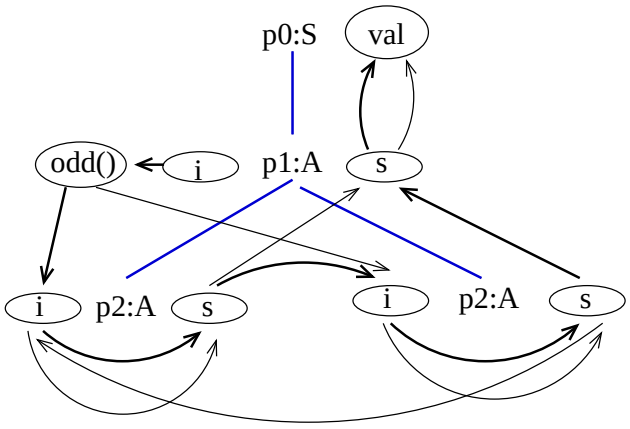

Figura 3

La figura 3 muestra un orden total del grafo de dependencias condicionales de la gramática condicional de la 2. Los arcos en negrita muestran un orden total por la restricción maximal $\{odd(), T\})$. Los arcos comunes muestran el camino por la restricción $\{odd(), F\})$. Se debe notar que el grafo contiene un ciclo, pero no es un ciclo válido.

3 Gramáticas de atributos NC(1) condicionales

Para extender la familia de gramáticas *NC(1)* a *NC(1)* condicionales, debemos redefinir los grafos $DPA(p \mid p_1, \ldots, p_k)$ [11] usando la definición de grafo de dependencias condicional de una producción p (D'^p) dado en (1).

$$DPA'(p \mid p_1, \ldots, p_k) = D'^p \cup DGC_{X_1}(p_1) \cup \ldots \cup DGC_{X_k}(p_k)$$

donde los símbolos $X_1, \ldots, X_k$ aparecen en la parte de $p_1, \ldots, p_k$, respectivamente.
Con esta nueva definición, cada grafo $DPA'(p \mid \ldots)$ contiene arcos con rótulos de la forma (c, T) o (c, F).

DEFINICIÓN 3.1: Una gramática de atributos *NC(1)* es una gramática *NC(1) condicional no circular* si cada grafo $DPA'(p \mid \ldots)$, $\forall p \in P$ no contiene ciclos válidos.

Se debe notar que un grafo de dependencias para una producción puede contener ciclos, pero si el ciclo involucra un camino no válido, o sea un camino de la forma $\ldots \overset{(c_k,T)}{\rightarrow} v_i \overset{(c_j,T)}{\rightarrow} \ldots \overset{(c_k,F)}{\rightarrow} v_n \ldots$, con $i = n$, las dependencias involucradas en el ciclo no ocurrirán nunca simultáneamente en ningún árbol sintáctico ya que la condición para evaluar el atributo v_i es complementaria de la condición para evaluar v_n.

4 Evaluación

Existen muchos métodos de evaluación de gramáticas de atributos y podemos clasificarlos en dos categorías: dinámicos y estáticos.

Los métodos dinámicos se basan básicamente en tres pasos: en primer lugar se construye el grafo de dependencias dado un árbol sintáctico, luego (asumiendo que el grafo es acíclico) se realiza un orden topológico sobre el grafo de dependencias para realizar la evaluación, la que es controlada por el orden total obtenido.

En los métodos estáticos, se han propuesto algoritmos específicos de evaluación para cada familia de GA. Éstos determinan el orden de evaluación haciendo una análisis estático de las producciones.

Si es posible determinar un orden total, luego se puede aplicar el método de evaluación mas adecuado para cada familia.

4.1 Secuencias de visita

Las OAG, ANCAG y NC(1) no pueden ser evaluadas en un número determinado a priori de pasadas sobre el árbol sintáctico.

Para las *OAG,* Kastens propuso un método de evaluación basado en *secuencias de visita.* Las secuencias de visita son secuencias $< v_1, v_2, \ldots, v_n >$ de tres operaciones:

1. *visit(i,j)*: visitar el nodo hijo i-ésimo por j-ésima vez.

2. *comp(r)*: computar la regla de atribución r que define un atributo.

3. *leave(j)*: ir al nodo padre del nodo corriente por j-ésima vez.

Estas secuencias de operaciones se determinan *estáticamente* para cada producción y permiten *caminar* por el árbol sintáctico e ir evaluando atributos en cada nodo. Obviamente una secuencia de visitas para una producción p debe ser consistente con las dependencias de sus atributos.

Las siguientes condiciones se deben mantener para que una $< v_1, v_2, \ldots, v_n >$ sea consistente con las dependencias:

1. Si $v_k = comp(r_n)$ que define al atributo a y $v_l = comp(r_m)$ que define el atributo b y usa a, entonces $k < l$.

2. Si $v_k = comp(r_n)$ que define un atributo a de símbolo X_i en p y $v_l = visit(i,j)$ o $v_l = leave(j)$ (y X_i es el símbolo de la parte izquierda de p) hacen uso de a, entonces $k < l$.

4.2 Secuencias de visita condicionales

La introducción de las reglas de atribución condicionales hace que las secuencias de visita tradicionales no contemplen las condiciones que hay que tener en cuenta para realizar una computación. En esta sección se introduce una extensión a las secuencias de visita que denominamos secuencias de visita condicionales.

DEFINICIÓN 4.1: Una *secuencia de visita condicional* $< cvs_1, cvs_2, \ldots, cvs_n >$ es una secuencia de operaciones de la forma (c, op) , donde $c \in \{T, F, \perp\}$ es un *rótulo* y *op* es una operación *check(c), comp(r), visit(i,j) o leave(j)* .

Cada operación (c, op) en la secuencia contiene un rótulo que identifica si la operación correspode a una evaluación de una regla primaria ($c = \perp$) o una regla condicional. Una operación estará rotulada con T (F) si corresponde a la evaluación de una regla $r \in R_c^T$ $(r \in R_c^F)$ (de alguna regla condicional c).

Para que una secuencia de visita condicional sea consistente con las dependencias entre sus atributos debemos redefinir las condiciones dadas en la sección anterior.

Las siguientes condiciones se deben mantener para que una secuencia $< cvs_1, cvs_2, \ldots, cvs_n >$ sea consistente con las dependencias:

1. Si $v_k = (c, comp(r_n))$ que define al atributo a y $v_l = (c, comp(r_m))$ o $v_l = (c, check(r_m))$ usan a, entonces $k < l$.

2. Si $v_k = (T, op(r_n))$ y $v_l = (c, check(r_m))$ usan un atributo a y $r_n \in R_{r_m}^T$, entonces $l < k$, donde *op* es una operación *comp*, *check*, *visit*, o *leave*.

3. Si $v_k = (F, op(r_n))$ y $v_l = (c, check(r_m))$ usan un atributo a y $r_n \in R_{r_m}^F$, entonces $l < k$, donde *op* es una operación *comp*, *check*, *visit*, o *leave*.

4. Si $v_k = (c, comp(r_n))$ que define un atributo a de símbolo X_i en p y $v_l = (c, visit(i,j))$ o $v_l = (c, leave(j))$ (y X_i es el símbolo de la parte izquierda de p) hacen uso de a, entonces $k < l$.

4.3 Generación de ordenes totales condicionales

Un *orden total condicional* es un grafo que tiene como particularidad que de cada nodo de condición salen dos ramas: una de las ramas contiene los arcos rotulados (c, T), que será utilizada cuando la condición evalúe a verdadero; la otra rama contiene los arcos rotulados (c, F), que será utilizada cuando la condición evalúe a falso.

Para poder generar secuencias de visitas, se debe determinar un orden total sobre las dependencias de los atributos de cada $DPA'(p|\ldots)$. No es posible aplicar un sort topológico directamente por la posible presencia de ciclos no válidos. En [11] se presenta un algoritmo para generar planes de evaluación para las GA NC(1). En [1] se presenta un algoritmo para generar un orden total condicional a partir de un grafo de dependencias condicional. Para generar un orden total condicional para las GA NC(1) condicionales es necesario combinar ambos algoritmos.

El algoritmo 1 computa los planes de evaluación condicionales[7] para las GA NC(1) condicionales. Los planes resultantes quedan almacenados en Θ.

Algorithm 1 Computación de un ordenes totales condicionales

$WL := \emptyset$, $\mu :=$ orden de evaluación arbitrario de los atributos de S
for cada producción p que tenga a S como parte izquierda **do**
  $WL := WL \cup \{p, \mu\}$ no marcado
end for
repeat
  Sea (p, ω) no marcado en W; $WL := WL - (p, \omega)$
  Sea la producción $p : X_0 \rightarrow \alpha_0 X_1 \alpha_1 \ldots X_k \alpha_k$
  for cada ADP'$(p \mid p_1, \ldots, p_k) \in$ SADP'(p) **do**
    $\Gamma[p, \omega, p_1, \ldots, p_k] := \Psi := genOTC(\text{DPA'}(p \mid p_1, \ldots, p_k) \cup \omega, \emptyset)$
    for cada $1 \leq i \leq k$ **do**
      $\Theta[p, \omega, i, p_1, \ldots, p_k] := \omega_i := proyectar(\Psi, A_{X_i})$
      if $(p_i, \omega_i) \notin WL$ **then**
        $WL := WL \cup \{(p_i, \omega_i)\}$ (sin marcar)
      end if
    end for
  end for
until $WL = \emptyset$

El algoritmo 2(extraído de [1]) genera de un orden total condicional. La función recursiva *genOTC(G,t)* genera un orden total condicional para un grafo G de dependencias condicionales y un conjunto de restricciones (o condiciones) t consistente con G.

La función *genOTC(G,t)* es invocada desde el algoritmo 1 con el grafo de dependencias condicionales $DPA'(p|\ldots)$ y con un conjunto de restricciones (inicialmente) vacío. genOTC obtiene un nodo maximal, lo remueve del grafo original (H_{old}) y simultáneamente va generando el orden total condicional para luego aplicarse resursivamente sobre el grafo reducido. En caso que el nodo maximal sea una condición, se generan dos subgrafos: uno por la rama de condiciones por el verdadero (n^T, O^T) y otro por falso (n^T, O^T), aumentando el conjunto de restricciones según corresponda. En caso que el nodo maximal seleccionado no sea una condición se genera un orden total (subgrafo) no condicional. En los retornos de las llamadas recursivas se van uniendo los subordenes (subgrafos) totales computados.

Es importante notar que la sentencia DPA'$(p \mid p_1, \ldots, p_k) \cup \omega$ no introduce ciclos válidos, si lo hiciera no sería NC(1) o estaría mal formada (recordemos que cada atributo se debe definir en cada rama de una regla

[7] U orden total condicional.

Algorithm 2 generación de un orden total condicional

$genOTC(H_old, t)$
if $H_{old} = (\emptyset, \emptyset)$ **then**
  return $(END, (\{END\}, \emptyset))$
else if $\exists n \in V_{H_old}, \nexists n', n' \rightarrow n \in E_{H_old}$ **then**
  $H_{new} := H_{old}/n$
  if $n \in C$ **then**
    $t^T := t \cup \{(n, T)\}$
    $(n^T, O^T) := genOTC(H_{new} \downarrow t^T, t^T)$
    $t^T := t \cup \{n, F\}$
    $(n^F, O^F) := genOTC(H_{new} \downarrow t^F, t^F)$
    return $(n, O^T \cup O^F \cup (\{n, n^T, n^F\}, \{n \xrightarrow{t^T} n^T, n \xrightarrow{t^F} n^F\})$
  else
    $(n', O') := genOTC(H_{new}, t)$
    return $(n, O' \cup (\{n, n'\}, \{n \xrightarrow{t} n'\}$
  end if
end if

condicional) y ω es un orden compatible dado por el contexto superior.

4.4 Generación de secuencias de visita condicionales

El algoritmo 3 genera secuencias de visita condicionales que luego pueden ser implementadas por cualquiera de los métodos propuestos en [2] con pequeñas modificaciones que deben tener en cuenta la computación de operaciones *check(r)* y luego computar las operaciones subsiguientes rotuladas con el resultado de la condición.

Algorithm 3 generación de un orden total condicional

$genSVC(o, l)$
Sea o un orden total condicional particionado en n subordenes s_i de la forma $< a_1, \ldots, a_n >$ (atributos no condicionales) o $< C, sc_T, sc_F >$, donde C denota un nodo de condición y sc_T y sc_F representan sus subordenes por T y F, respectivamente. Sea $l \in \{T, F, \perp\}$
for cada orden s_i **do**
  if s_i es de la forma $< a_1, \ldots, a_n >$ **then**
    $svc :=$ secuencias de visita condicionales para s_i generadas en forma tradicional (según Kastens) con rótulo $\perp$.
  else if s_i es de la forma $< C, sc_T, sc_F >$ **then**
    $svc :=< svc, (\perp, check(C)) >$
    $svc :=< svc, genSVC(sc_T, T) >$
    $svc :=< svc, genSVC(sc_F, F) >$
  end if
end for
return svc

Una implementación basada en subrutinas recursivas sería similar a la propuesta por Kastens con la adaptación que las operaciones $(l, check(C))$ generan sentencias *if (C) then* s_1 *else* s_2*endif* del lenguaje utilizado, donde s_1 corresponde a las operaciones rotuladas por T y s_2 corresponde a las operaciones rotuladas por F (correspondientes a cada rama del nodo condición del orden total condicional) ubicadas apropiadamente en los cuerpos de los procedimientos.

Un evaluador de GA NC(1) condicionales debe primero realizar un recorrido descendente para seleccionar un plan en cada nodo del árbol y luego evaluar utilizando un evaluador de secuencias de visita condicionales como el descripto. En [11] se describe el proceso.

5 Conclusiones

Se ha presentado una extesión a las GA NC(1) aumentando su poder expresivo y se proponen algorimos para la generación de planes de evaluación. Se han introducido las secuencias de visitas condicionales que permiten la evaluación extendiendo de una manera muy simple y obvia los evaluadores tradicionales. Los conceptos analizados se pueden utilizar directamente para implementar evaluadores de atributos para una familia de GA que contiene a la mayoría de las GA utilizables en la práctica. Al momento se continúa investigando sobre diferentes alternativas de implementación de evaluadores basados en secuencias de visita condicionales y sus posibles optimizaciones, como pueden ser detección de subsecuencias comunes, operaciones inalcanzables, etc.

Referencias

[1] *[Boyland J.]Conditional Attribute Grammars*, ACM Transactions on Programming Languages and Systems, vol 18, No. 1. January 1996, pages 73-108.

[2] *[Kastens¸U.] Implementation of visit-oriented attribute evaluators.* In Attribute grammars¸Applications and Systems. SAGA Proceedings˙Eds.˙Lecture Notes in Computer Science¸vol 545˙Springer-Verlag¸114-139.

[3] *[Kastens U.] Ordered Attribute Grammars.* Acta Informática 13, Springer-Verlag, pp. 229-256. 1980.

[4] *[Yang¸W] Multiplan attribute grammars.*

[5] *[Jazayeri M., Ogden W., Rounds W.] The intrinsically exponential complexity of the circularity problem for attribute grammars.* Comm. ACM 18, 12 Dec. pp. 687-706.

[6] *[P. Deransart, M. Jourdan, B. Lorho] Attribute Grammars: Definitions, systems and bibliography.* Lecture Notes in Computer Science 323, Springer-Verlag, New York. 1988.

[7] *[Knuth D.] Semantics of context-free languages.* Mathematical System Theory 2, pp. 127-145. 1968.

[8] *[Yang, W.] A Classification of non-circular attribute grammars.* Computer and Information Science Dept., National Chaio-Tung Univ., Hsinchu, Taiwan. 1997.

[9] *[Kennedy K., Warren S.] Automatic generation of eficient evaluators for attribute grammars*, pp. 32-49 in Conference Record of the Third ACM Symposium on Principles of Programming Languages, Atlanta, ACM, New York. 1976.

[10] *[Paaki J.] Attribute grammar paradigms - A high-level methodology in languaje implementation*, ACM Computing Surveys 27(2), pp. 196-255. 1995.

[11] [Yang, W.] A Classification of Non-Circular Attributte Grammars Based on the Look-Ahead Behavior. Computer and Information Science Department, National Chiao-Tung University, HsinChu, Taiwan. 1999.

[12] *[Arroyo M., Aguirre J., Florio N.] Un generador de Evaluadores de Gramáticas de Atributos NC(1) de máximo paralelismo sin sincronización entre procesos.* Cacic 2000. Ushuaia, Argentina. 2000.