Un generador de evaluadores de gramáticas de atributos NC(1) de máximo paralelismo sin sincronización entre procesos

Marcelo Arroyo[1], Nicolás Florio[2], Jorge Aguirre[3]

Área de Computación, Fac. de Cs. Exactas, Fco−Qcas y Naturales
Universidad Nacional de Río Cuarto.

Palabras clave: Gramáticas de atributos, Generadores de Compiladores, Concurrencia, Paralelismo, Procesadores de Lenguajes.

Resumen

En éste trabajo se presenta el desarrollo de un generador de evaluadores de gramáticas de atributos no circulares −*NC(1)*−. Este generador computa estáticamente toda la información necesaria para que el evaluador generado construya, en tiempo proporcional a la longitud del árbol, procesos concurrentes independientes capaces de producir la evaluación. Esto permite evaluar aprovechando el máximo paralelismo posible sin ningún tipo de sincronización. En éste trabajo se aprovechan dos enfoques: el clásico enfoque secuencial, que determina un orden lineal de evaluación, con el enfoque recientemente propuesto por Wuu Yang para el particionado de mayor refinamiento del grafo de dependencias en regiones disjuntas. El algoritmo desarrollado evita un recorrido adicional sobre el árbol sintáctico y completa el estudio de los detalles para la implementación del evaluador. Este desarrollo se integrará como evaluador alternativo de un generador de procesadores de lenguajes.

Este trabajo se está desarrollando en el marco del proyecto de investigación y desarrollo 189A, subsidiado por la Secretaría de Ciencia y Técnica de la Universidad Nacional de Río Cuarto.

1 marroyo@dc.exa.unrc.edu.ar
2 nflorio@dc.exa.unrc.edu.ar
3 jaguirre@dc.uba.ar

1. Introducción

Desde la introducción de las Gramáticas de Atributos (*GA*) en 1968[1], han sido motivo de un gran interés en investigación. Las *GA* son un formalismo simple y poderoso para especificar computaciones sobre lenguajes libres de contexto. Uno de los principales intereses es el desarrollo de algoritmos de evaluación de las instancias de los atributos en un árbol sintáctico. Existen muchos algoritmos para realizar evaluación secuencial pero últimamente la aparición de arquitecturas paralelas y lenguajes de programación con soporte de concurrencia hacen interesante el desarrollo de algoritmos de evaluación paralela.

Para poder paralelizar la evaluación de las instancias de los atributos en un árbol sintáctico se pueden seguir diferentes estrategias. Generalmente se particiona el árbol sintáctico o su grafo de dependencias para asignar cada partición a un proceso diferente.

El particionado puede basarse en los no terminales o en producciones para obtener los diferentes bloques, pero si no se tienen en cuenta las dependencias entre los atributos, los procesos pueden necesitar sincronización y/o comunicación para intercambiar valores de los atributos.

Si las instancias de los atributos evaluadas por procesos diferentes son independientes, entonces se puede eliminar la comunicación y sincronización entre procesos.

Para el particionado basado en el grafo de dependencias generalmente se siguen los siguientes enfoques: estático, dinámico e híbrido. El particionado dinámico se realiza luego que se haya construido el árbol sintáctico y su grafo de dependencias, lo cual causa una recarga adicional al proceso de evaluación. En el particionado estático se analizan las dependencias entre los atributos de la GA. Los métodos híbridos calculan estáticamente cierta información que luego es tomada por el evaluador para obtener las particiones en tiempo de ejecución.

Aquí se mostrará un algoritmo para calcular particiones estáticamente basado en las dependencias entre los atributos y que simultáneamente genera planes de evaluación de las instancias de los atributos para cada producción. También se presenta un algoritmo evaluador que selecciona la partición del grafo de dependencias (*GD*) correspondiente al árbol sintáctico, selecciona el plan correspondiente para cada instancia de las producciones y dispara procesos evaluadores concurrentes independientes.

El algoritmo permite la evaluación de cualquier *GA* que corresponda a la clase *NC(1)*[4]. La principal característica de las *GA* que pertenecen a la familia *NC*, es que permiten generar estáticamente evaluadores basados en *secuencias de visita*. Por lo anterior, las *GA* de la familia *NC* pueden ser vistas como una generalización de las *ordered attribute grammars* (*OAG*) definidas por Kastens [2] y *l−ordered GA*[5]. La diferencia radica en que en las *NC*, cada producción puede tener más de un plan de evaluación. Un plan de evaluación consiste de una secuencia de instrucciones de tres clases: *eval*, *visit* y *suspend*. En cada visita a un nodo se pueden evaluar algunos atributos del nodo, y en cada visita al padre (*suspend*) se computan algunos atributos heredados y en cada visita a algunos de los hijos (*visit*) se computan algunos atributos sintetizados[3].

2. Gramáticas de atributos

Una *GA* se construye a partir de una gramática libre de contexto (*CFG*) $<N,T,P,S>$, donde N es un conjunto finito de símbolos no terminales, T es un conjunto finito de símbolos terminales, P es un conjunto finito de producciones de la forma $X \rightarrow \alpha$ donde $X \in N$, $\alpha \in (N \cup T)^*$ y $S \in N$ (símbolo de comienzo). Para cada $X \in N$, existe al menos una producción de la forma $X \rightarrow \alpha$. Denotaremos a una producción q de la forma $q: X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 ... X_k \alpha_k$, donde $X_i \in N$ y $\alpha_i \in T^*$, $(0 \leq i \leq k)$. Asumiremos que S no aparece en la parte derecha de ninguna producción.

4 Non−Circular basada en información de descendentes inmediatos.

5 Estas AG permiten que a cada producción se pueda asociar una única secuencia (plan) de evaluación.

Con cada símbolo $X \in N$ se asocia un conjunto de *atributos* denominado $A_X=IA_X \cup SA_X$, donde IA_X son los atributos heredados de X y SA_X los atributos sintetizados de X $(IA_X \cap SA_X=\varnothing)$. Como S no aparece en ninguna parte derecha, los atributos de S son sintetizados ($As=SA_S$).

Las diferentes instancias de los atributos describen propiedades de instancias específicas de los símbolos en un árbol sintáctico. Por simplicidad, asumiremos que los atributos de diferentes símbolos tienen diferentes nombres.

Un atributo a de un símbolo X se denota como $X.a$. Dado que pueden haber muchas ocurrencias de un símbolo en una producción, puede haber muchas *ocurrencias* de un atributo en una producción. Además como una producción puede aparecer aplicada más de una vez en un árbol sintáctico, pueden haber muchas *instancias* de una ocurrencia de un atributo en un árbol sintáctico.

Con cada producción se asocia un conjunto de *atribuciones* de la forma $X_i.a:=f(X_j.b,...,X_m.c)$, con la restricción que $X_i.a$ es un atributo sintetizado de X_i y X_i aparece en la parte izquierda de la producción, o $X_i.a$ es un atributo heredado de X_i y X_i aparece en la parte derecha de la producción[6]. La función f no produce efectos colaterales.

Las ecuaciones en la atribución inducen dependencias entre las ocurrencias de los atributos en una producción q. Estas dependencias se pueden representar en un grafo de dependencias de q, denotado como $DP(q)$, cuyos nodos representan las ocurrencias de los atributos en q y los arcos las dependencias entre las ocurrencias. Un arco $X.a \rightarrow Y.b$ significa que la ocurrencia $X.a$ es un parámetro de la función que evalúa a $Y.b$ (existe una ecuación de la forma $Y.b=f(...,X.a,...)$).

3. Computación de los planes de evaluación

Las instancias de atributos en un árbol sintáctico deben evaluarse en un orden que sea consistente con respecto a sus dependencias. Estas dependencias entre las instancias de atributos en un árbol sintáctico pueden inferirse desde las dependencias entre las ocurrencias en las producciones individuales analizando su atribución (denominadas dependencias directas). Una vez conocidas las dependencias directas se pueden calcular las dependencias transitivas entre los atributos de un símbolo. Las dependencias transitivas entre los atributos de un símbolo se pueden calcular en base a su contexto[7] (superior) o en base la estructura derivada (hijos) a partir de éste símbolo (contexto inferior).

Las dependencias transitivas entre los atributos de un símbolo X debido a su contexto se denominan *dependencias transitivas ascendentes*. De otro modo, las dependencias transitivas entre los atributos de un símbolo debido a su estructura derivada se denominan *dependencias transitivas descendentes*. Estas últimas dependencias transitivas se usan para la construcción de los planes de evaluación, mientras que las primeras serán utilizadas por el evaluador para seleccionar un plan en particular durante la ejecución. El plan seleccionado depende de la instancia particular del árbol sintáctico.

DEFINICIÓN: El *grafo de dependencias transitivas descendentes* de un símbolo X es el grafo $D(X)=<Nodos,Arcos>$ donde:

$Nodos=A_X$

$Arcos=\{X.a \rightarrow X.b$ / *existe una dependencia transitiva desde X.b a X.a en algún subárbol derivado a partir de X*$\}$

La computación del grafo $D(X)$ es costosa, por lo que se utilizan aproximaciones de $D(X)$. Uno de los algoritmos más conocidos es el propuesto por Kennedy y Warren en [4] (algoritmo $Down(X)$) que incluye todas las dependencias transitivas posibles entre los atributos de X en todos los posibles árboles sintácticos[8].

6 Esta notación se llama *forma normal* en la literatura.

7 Denominamos contexto de un símbolo X al subárbol T tal que X es una hoja de T.

8 Además puede incluir algunas falsas dependencias.

Es posible computar grafos de dependencias transitivas descendentes más representativos considerando cada producción individualmente[5].

DEFINICIÓN: Sea *X* el no terminal de la parte izquierda de una producción *q*. El *Grafo Característico de dependencias transitivas descendentes de X* en base a los subárboles derivados vía la producción *q*, denotado *DCGx(q)*, es un grafo:

DCGx(q)=<Nodos,Arcos> donde:

Nodos=A_X

Arcos={X.a→X.b / existe una dependencia transitiva desde X.b a X.a en algún subárbol derivado a partir de X aplicando la producción q}

Formalmente: Sea $q{:}X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 ... X_k \alpha_k$ una producción, sea p_i una producción cuya parte izquierda sea X_i , $(0 \leq i \leq k)$:

$DCGx(q) = \cup$ { ‹*(DPA(q | p₁,p₂,...,pₖ),A_X) tal que p₁,p₂,...,pₖ son producciones cuya parte izquierda son X₁, X₂, ..., Xₖ respectivamente* }

$DPA(q \mid p_1,p_2,...,p_k) = DP(q) \cup DCG_{X1}(p_1) \cup DCG_{X2}(p_2) \cup ... \cup DCG_{Xk}(p_k)$

$\prod(DPA(q \mid p_1,p_2,...,p_k),A_X)$ = *proyección del grafo* $DPA(q \mid p_1,p_2,...,p_k)$ respecto a *los atributos de X.*

El grafo $DPA(q \mid p_1,p_2,...,p_k)$ se denomina el grafo de dependencias aumentado (o extendido) de la producción *q* con respecto a los subárboles derivados por aplicar $p_1,p_2,...,p_k$. Sea $SDPA(q)=\{DPA(q \mid p_1,p_2,...,p_k)\}$.

En [5] se muestra que el los grafos $DCG_X(q)$ están incluídos en los grafos computados por el algoritmo $Down_X(q)$ propuesto en [4] y constituyen mejores aproximaciones.

El siguiente algoritmo computa los grafos *DCGx(q).*

Algoritmo: ComputaDCG
/* Inicialmente, DCG(X,q) no tiene definido ningún arco */
repetir
    Cambios:=Falso
    para cada producción $q{:}X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 ... X_k \alpha_k$
        para cada combinación de $(p_1,p_2,...,p_k)$, tal que p_i tiene a X_i en su parte izquierda
            $G{:=}DP(q) \cup DCG(X_1, p_1) \cup DCG(X_2, p_2) \cup ... \cup DCG(X_k, p_k)$
            $NDGC(X_0 , q){:=}$proyección(G,A_{X0}) /* proyección de G en los atributos de X_0 */
            si $NDGC(X_0 , q) \not\subseteq DCG(X_0 , q)$ **entonces**
                $DCG(X_0 , q){:=}DCG(X_0 , q) \cup NDGC(X_0 , q)$
                Cambios:=Verdadero
            fin si
        fin para
    fin para
hasta Cambios = Falso

DEFINICIÓN: Una GA es NC(1) ⇔ ∀q de GA tal que si $DPA(q \mid p_1,p_2,...,p_k) \in SDPA(q)$ es acíclico.

Una propiedad fundamental es que cada GA NC(1) es una GA bien definida[5], además las GA NC(1) son una familia más grande que las estrictamente no circulares (ANCAG)[9].

Si una GA es NC(1), de acuerdo a su definición, cada grafo $DPA(q \mid p_1,p_2,...,p_k)$ es acíclico, por lo tanto, cualquier orden topológico de $DPA(q \mid p_1,p_2,...,p_k)$ podría ser una posible secuencia de

9 Una GA es ANCAG o NC(0) si las dependencias directas de los atributos en cada producción no son circulares.

evaluación para los atributos de *q*. Sin embargo, el contexto de una instancia individual de *q* en algún árbol sintáctico podría imponer restricciones en en el orden de evaluación de los atributos del símbolo de la parte izquierda de *q*. Esto determina que cada producción puede tener varios planes posibles de evaluación.

3.1 Algoritmo de generación de planes de evaluación

Cada plan de evaluación define escencialmente un orden de evaluación de las ocurrencias de los atributos. Cada orden de evaluación debe ser consistente con las dependencias entre las ocurrencias de los atributos. Existen tres clases de dependencias para una ocurrencia de un atributo en una producción:

a. las dependencias directas en base a las ecuaciones de la atribución de la producción.
b. las dependencias transitivas ascendentes entre las ocurrencias de los atributos del símbolo de la parte izquierda de la producción.
c. las dependencias transitivas descendentes entre las ocurrencias de los atributos de los símbolos no terminales de la parte derecha de la producción.

El siguiente algoritmo computa los planes de evaluación para cada producción[5].

Algoritmo: ComputarPlanes
L=$\varnothing$
Def:=**falso**
μ := OrdenArbitrario(A_S) /* Orden total arbitrario de los atributos de S */
para cada producción q cuya parte izquierda es S
    Def(q,μ):=**verdadero**
    L:=L$\cup${(q,μ)}
fin para
repetir
    (q,ω):=SacarElemento(L) /* Saca un elemento de L */
    sea q:$X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 ... X_k \alpha_k$
    para cada DPA(q | $p_1,p_2,...,p_k$) $\in$ SDPA(q)
        ψ:=ComputarOrden(DPA(q | $p_1,p_2,...,p_k$), ω)
        Γ[q,ω, $p_1,p_2,...,p_k$]:= ψ
        para cada X_i de la parte derecha de q
            ω_i:=proyectar(ψ,A_{Xi})
            Δ [q,ω,i,$p_1,p_2,...,p_k$]:= ω_i
            si no Def(p_i,ω_i) entonces
                Def(p_i,ω_i):=**verdadero**
                L:=L$\cup${(p_i,ω_i)}
            fin si
        fin para
    fin para
hasta L=$\varnothing$

función ComputarOrden(G,ω)
    sea ω : $a_1 \rightarrow a_2 .. \rightarrow a_m$
    para cada i desde 1 hasta m−1
        G:=G$\cup${$a_i \rightarrow a_{i+1}$}
    fin para
retornar un orden total compatible con el orden parcial representado por el grafo G.

El algoritmo utiliza una lista de trabajo (L) que almacena los elemento aún por tratar. El proceso termina cuando se han tratado todos sus elementos. La función *Def* se utiliza para evitar el procesamiento repetido de tuplas. Al comienzo, cualquier orden arbitrario de los atributos sintetizados de S, no causa ninguna circularidad en la evaluación de los atributos en el árbol sintáctico. ψ es un orden total de evaluación de los atributos del grafo *DPA(q | $p_1,p_2,...,p_k$)* (un

posible orden de evaluación de los atributos de q), ω es la proyección de ψ con respecto a los atributos del símbolo no terminal de la parte izquierda de q (restricciones impuestas por el contexto). El algoritmo *ComputarPlanes* computa dos funciones, Γ y Δ. La función Γ será usada para seleccionar un plan de evaluación de cada producción entre todos los posibles por el evaluador. $\Gamma[q,\omega, p_1,p_2,...,p_k]$ es el orden de evaluación de las ocurrencias de los atributos en la producción q, cuando los atributos del no terminal de la parte izquierda de q deben ser evaluados en el orden ω (restricción impuesta por el contexto) y las producciones aplicadas a los símbolos de la parte derecha de q son $p_1,p_2,...,p_k$. La función $\Delta[q,\omega,\boldsymbol{i},p_1,p_2,...,p_k]$ es una proyección de $\Gamma[q,\omega, p_1,p_2,...,p_k]$ con respecto a los atributos del $\boldsymbol{i}$−ésimo no terminal de la parte derecha de q.

Debemos notar que la función *ComputarOrden* determina un orden total a partir de un orden parcial. Como pueden existir varios posibles órdenes totales, *ComputarOrden* selecciona uno de ellos.

La función *proyectar*(ψ,A_X) elimina de la secuencia ψ las ocurrencias de los atributos que no están en A_X. La complejidad de este algoritmo se muestra en [6].

4. Particionado de los atributos

A continuacion, se presenta un algoritmo estático de partición de grafos de dependencias de atributos para gramáticas de atributos *NC* desarrollado en [5]. Este algoritmo particiona el grafo de dependencias de atributos en regiones disjuntas. Los atributos de diferentes regiones son independientes. El algoritmo construye el conjunto de todas las particiones viables para cada producción analizando la gramática. Estas particiones serán utilizadas para la generación de procesos concurrentes de evaluación.

Para particionar el grafo de dependencias de una producción se deben considerar nuevamente las tres tipos de dependencias entre atributos: las dependencias base o directas, las dependencias transitivas ascendentes y las dependencias descendientes transitivas.

Diferentes instancias de una producción p en un árbol sintáctico tienen las mismas dependencias base, pero posiblemente diferentes dependencias descendientes transitivas y ascendentes transitivas. Es por esta razón que una producción puede tener más de una partición viable sobre sus ocurrencias de atributos. Es importante destacar que en este algoritmo las dependencias descendientes transitivas y ascendentes transitivas de una instancia de una producción son independientes (ya que el contexto y los subárboles no se superponen).

DEFINICIÓN: una ***partición viable de un grafo*** de dependencias de atributos es una partición de los nodos del grafo de dependencias de atributos que satisface la siguiente condición: Sean $X.a \rightarrow X.b$ dos instancias de atributos en el grafo de dependencias, si hay una relación de dependencia $X.a \rightarrow X.b$ en el grafo, los dos nodos $X.a$ y $X.b$ están en el mismo bloque.

DEFINICIÓN: sea π una partición viable de un grafo de dependencias de un árbol sintáctico atribuido T, sea X una instancia de un no terminal en T (es decir un nodo interno). La proyección de π sobre las instancias de los atributos de X es una ***partición viable de los atributos de X***.

DEFINICIÓN: sea π una partición viable de un grafo de dependencias de un árbol sintáctico atribuido T, sea q una instancia de una producción en T. La proyección de π sobre la s instancias de los atributos de q es una ***partición viable de las ocurrencias de los atributos de la producción q***.

DEFINICIÓN: sea X un nodo correspondiente a un no terminal en un árbol sintáctico T, sea G_x el grafo de dependencias de atributos del árbol T. La partición basada en la proyección G_X sobre los atributos de X es llamada una ***partición admisible de los atributos de X***.

DEFINICIÓN: sea X un nodo correspondiente a un no terminal en un árbol sintáctico T, sea q una producción aplicada a X en T, sea T_X el subárbol de T con raíz en X. La partición basada en la proyección de G_x sobre la ocurrencia de q es llamada una ***partición admisible de los atributos de q***.

A continuación se presenta el algoritmo de partición:

Algoritmo: ParticiónAG

para cada no terminal X **hacer**
  $\Xi(X):=\emptyset$; $\Theta(X):=\emptyset$
fin para
para cada producción q **hacer**
  π_q:=la partición de los atributos de q en base a *DP(q)*
fin para
repetir /* pasada ascendente */
  cambio:= **falso**
  para cada producción q: $X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 \ldots X_k \alpha_k$ **hacer**
    para cada $\sigma_1 \in \Xi(X_1)$, $\sigma_2 \in \Xi(X_2)$,..., $\sigma_k \in \Xi(X_k)$ **hacer**
      π:= mezclar(...mezclar(mezclar(π_q, σ_1), $\sigma_{2),\ldots,}$ σ_k)
      σ:= proyectar(π,{atributos de X_0})
      $\sum(q \mid \sigma_1, \sigma_2, \ldots, \sigma_k) := \sigma$
      si $\sigma \notin \Xi(X_i)$
          cambio:= **verdadero**
          $\Xi(X_0) := \Xi(X_0) \cup \{\sigma\}$
      fin si
    fin para
  fin para
hasta cambio=**falso**

/* pasada descendente */
$\Theta(S) := \Xi(S)$, donde *S* es el símbolo distinguido de la gramática
repetir
  cambio:= **falso**
  para cada producción q: $X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 \ldots X_k \alpha_k$ **hacer**
    para cada $\sigma_0 \in \Theta(X_0)$, $\sigma_1 \in \Xi(X_1)$, , $\sigma_2 \in \Xi(X_2)$..., $\sigma_k \in \Xi(X_k)$ **hacer**
      si Π (q $\mid \sigma_1, \sigma_2, \ldots, \sigma_k$) aún no ha sido definido
          π:= mezclar(...mezclar(mezclar(π_q, σ_1), $\sigma_{2),\ldots,}$ σ_k)
          Π (q $\mid \sigma_1, \sigma_2, \ldots, \sigma_k$) := σ
          para *i*:=1 **hasta** k **hacer**
            σ:= proyectar(π,{atributos de X_0})
            Φ(q, π, *i*):= σ
            si $\sigma \notin \Theta(X_i)$
                cambio:=**verdadero**
                $\Theta(X_0) := \Theta(X_0) \cup \{\sigma\}$
            fin si
          fin para
      fin si
    fin para
  fin para
hasta cambio=**falso**

función proyectar(π,V) $\rightarrow$ partición
/* π es una partición. V es un conjunto de ocurrencias de atributos */
/* el resultado es la partición más fina de los atributos de V que es consistente con π */
  para cada bloque β de π **hacer**
    para cada elemento t de β **hacer**
      si t $\notin$ V **entonces** $\beta := \beta - \{t\}$ **fin si**
    fin para
    si $\beta = \emptyset$ **entonces** $\pi := \pi - \{\beta\}$ **fin si**
  fin para
  retornar π
fin función

```
función mezclar(π,σ) → partición
/* π y σ son particiones de una producción y de un no terminal, respectivamente */
/*  mezclar encuentra la partición más fina consistente con π y σ */
 para cada bloque γ de σ hacer
     β:=∅
     para cada elemento t de γ hacer
          si t ∉ γ
               δ:= el bloque en γ que contiene a t
               π:= π − {δ}
               β:= β ∪ δ
          fin si
       fin para
       π:= π − {β}
 fin para
 retornar π
fin función
```

El algoritmo ParticiónAG calcula todas las particiones viables de las ocurrencias de atributos en las producciones de la gramática. El algoritmo realiza dos recorridos sobre las producciones: la primer pasada (ascendente) calcula todas las particiones admisibles de los atributos de los no terminales. Durante la segunda pasada (descendente) se computan todas las particiones viables de las ocurrencias de atributos en las producciones. En la primera pasada, se consideran las dependencias base y las dependencias transitivas descendentes. Para cada producción q: $X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 ... X_k \alpha$, se determina una partición admisible de los atributos de X_0 mezclándolos con la partición base de q y una partición admisible de los atributos de X_i (con i=1,2..,k), para luego proyectar la partición resultante sobre los a los atributos de X_0 . Las particiones admisibles calculadas durante la primera pasada son almacenadas en la función $\sum(q \mid \sigma_1, \sigma_2, ..., \sigma_k)$, la cual asocia una combinación de particiones admisibles de atributos $(\sigma_1, \sigma_2, ..., \sigma_k)$ de X_i (i=1,2..,k) con atributos de X_0. Todas las particiones admisibles de los atributos de los no terminales son almacenadas en la función Ξ.

Al final de la primera pasada, Ξ(*S*), donde *S* es el símbolo distinguido de la gramática, es el conjunto de las particiones admisibles de los atributos de *S*.

En la segunda pasada, se consideran los tres tipos de dependencias mencionadas. Para cada producción *q*: $X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 \alpha_2 ... X_k \alpha$, se determina una partición viable de las ocurrencias de los atributos en la producción q mezclando la partición base de *q*, una partición viable de los atributos de X_0 y una partición admisible de los atributos de X_i (σ_i) (con i=1,2..,k). La partición resultante, llamada σ, es una partición viable de las ocurrencias de los atributos de la producción *q* y es almacenada como $\Pi\ (q \mid \sigma_1, \sigma_2, ..., \sigma_k)$. La información almacenada en Π será utilizada para la definición de los planes de visita. Cuando se encuentra una nueva partición viable de atributos de un no terminal, supongamos *X*, ésta se utiliza para calcular nuevas particiones viables de las ocurrencias de atributos de producciones cuya parte izquierda es *X* (contexto inferior). El algoritmo finaliza cuando no se pueden encontrar más particiones viables. La complejidad de este algoritmo se muestra en [5].

El generador del evaluador de atributos computa los planes posibles y las particiones viables que serán usadas por el algoritmo de evaluación.

```
Algoritmo: GenerarEvaluador
para cada producción q hacer
        DP(q):=ComputarDP(q)   /* Computación de las dependencias directas de q */
fin para
GenerarPlanes    /* Se computan los planes posibles de evaluación para cada producción */
ParticiónAG      /* Se computan las particiones viables para cada producción */
```

5. El algoritmo de evaluación de atributos

El evaluador de atributos deberá, en tiempo de ejecución, seleccionar para cada nodo del árbol sintáctico un plan de evaluación y una de las particiones viables.

Algoritmo: EvaluarAtributos (T)
/* T es un árbol sintáctico */
/* PT será la partición de T */
SeleciónAscendente(raíz de T)
σ:=partición seleccionada para la raíz de T
PT:=σ
SelecciónDescendente(raíz de T,σ)
/* Se disparan procesos concurrentes evaluadores para cada partición de T */
para cada elemento *e* de la partición PT **hacer**
    CrearProceso(VS(*e*))
fin para

procedimiento SelecciónAscendente(n)
    /* Este procedimiento selecciona una partición para cada instancia de cada producción en T */
    sea q la producción aplicada en el nodo n del árbol
    sean $m_1, m_2, ..., m_k$ los nodos hijos no terminales de n
    para cada m_i **hacer**
        SelecciónAscendente(m_i)
    fin para
    sean $\sigma_1, \sigma_2, ..., \sigma_k$ las particiones seleccionadas en los nodos $m_1, m_2, ..., m_k$, respectivamente
    seleccionar la partición $\sum(q \mid \sigma_1, \sigma_2, ..., \sigma_k)$ para el nodo n.
fin procedimiento

procedimiento SelecciónDescendente(n,σ,ω)
    sea q la producción aplicada en el nodo n del árbol
    sean $m_1, m_2, ..., m_k$ los nodos hijos no terminales de n
    sean $\sigma_1, \sigma_2, ..., \sigma_k$ las particiones seleccionadas en los nodos $m_1, m_2, ..., m_k$, respectivamente
    π:=Π $(q \mid \sigma_1, \sigma_2, ..., \sigma_k)$
    seleccionar la partición π para la producción q
    /* selección del plan para la produción q */
    sean $p_1,p_2,...,p_k$ las producciones aplicadas en $m_1, m_2, ..., m_k$
    Plan[q]:=proyección(Γ[q,ω, $p_1,p_2,...,p_k$],π)
    PT:=NuevoPT(PT,π) /* se aumenta la partición inducida de T */
    para cada m_i **hacer**
        SelecciónDescendente(mi,Φ(q, π, i), Δ [q,ω,i,$p_1,p_2,...,p_k$])
    fin para
fin procedimiento

El algoritmo presentado necesita realizar dos recorridos sobre el árbol sintáctico. En el primer recorrido (ascendente) se selecciona una de las particiones viables para los atributos de la raíz en base a las instancias de las producciones aplicadas en el árbol. Una vez definida la partición de los atributos de la raíz del árbol, durante el recorrido descendente, para cada nodo correspondiente a una instancia de una producción, se selecciona una de las particiones viables (π:=Π $(q \mid \sigma_1, \sigma_2, ..., \sigma_k)$) que se utiliza para obtener la proyección del plan de evaluación seleccionado (Γ[q,ω, $p_1,p_2,...,p_k$])[10]. La función *proyección(Γ[q,ω, $p_1,p_2,...,p_k$],π)* divide el plan seleccionado para *Γ[q,ω, $p_1,p_2,...,p_k$]* en planes inducidos por cada elemento de la partición π. Cada plan resultante de la proyección podría ser evaluado en forma independiente ya que fue escogido en base a la partición viable seleccionada. Se define en *PT* la partición del árbol sintáctico T inducida por las particiones seleccionadas en cada nodo de T. *PT* se utiliza para asignar procesos de evaluación para

10 Plan[q] es un conjunto de planes (secuencias de visita) de evaluación.

cada partición. VS(e) es un evaluador secuencial basado en secuencias de visitas.

7. Un ejemplo

P0: *S→X*	p1: *X→1Y*	P2: *X→2Y*	P3: *Y→3*	P4: *Y→*4
S.j:=X.b	*Y.e:=X.a*	*Y.e:=X.a*	*Y.f:=Y.e*	*Y.h:=Y.e*
S.j:=X.d	*X.b:=Y.f*	*X.b:=Y.f*	*Y.h:=Y.g*	*Y.f:=Y.g*
X.a:=0	*Y.g:=X.c*	*Y.g:=X.c*		
X.c:=1	*X.d:=Y.h*	*X.d:=Y.h + Y.f*		

La figura 1 muestra los grafos de dependencias de cada producción (DP(q)) y la figura 2 muestra dos árboles sintácticos posibles de la gramática de atributos dada. La tabla 1 muestra las funciones computadas por el algoritmo ParticiónAG y a continuación se describen los planes seleccionados para cada producción.

Las producciones del ejemplo 1) se particionan en π_1, π_4 y π_7, que inducen una partición del grafo de dependecias *PT = {<j,a,b,e,f>,<k,c,d,g,h>}*.

Las producciones del ejemplo 2) se particionan en π_3, π_5 y π_8 que inducen una partición del grafo de dependencias *PT = {<j,a,b,e,f,k,c,d,g,h>}*.

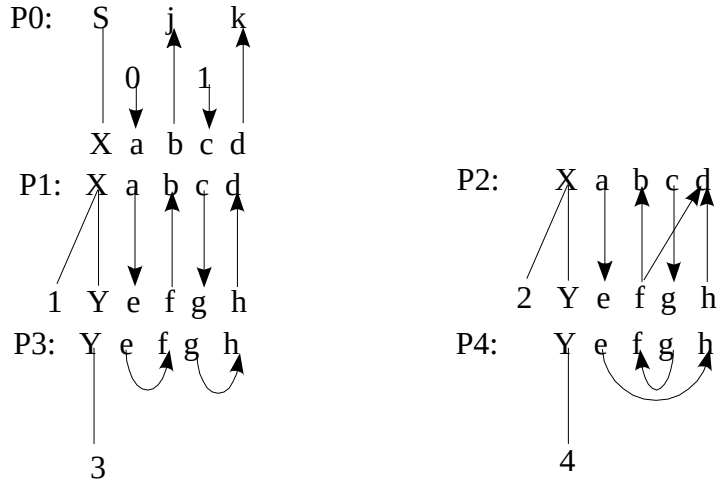

Figura 1

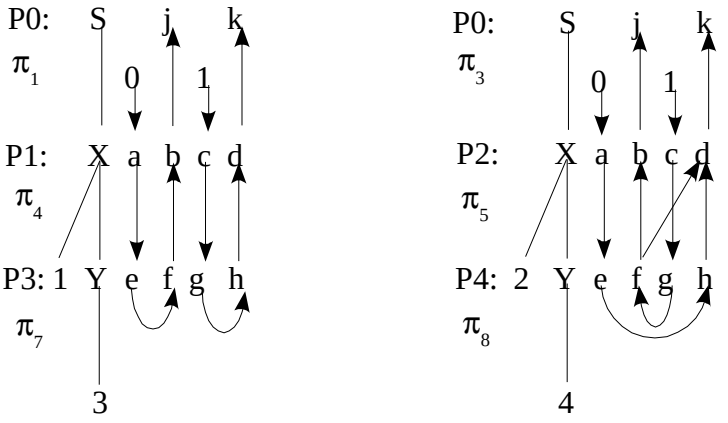

Figura 2

función Σ	función Π	función Φ	tablas Ξ y Θ
$\Sigma(P0 \mid \alpha_3)= \alpha_6$ $\Sigma(P0 \mid \alpha_4)= \alpha_6$ $\Sigma(P0 \mid \alpha_5)= \alpha_7$	$\Pi(P0\|\alpha_6,\alpha_3)=\pi_1$, $\Pi(P0\|\alpha_6,\alpha_4)=\pi_2$ $\Pi(P0\|\alpha_6,\alpha_5)=\pi_3$, $\Pi(P0\|\alpha_7,\alpha_3)=\pi_3$ $\Pi(P0\|\alpha_7,\alpha_4)=\pi_3$, $\Pi(P0\|\alpha_7,\alpha_5)=\pi_3$	Φ (P0,π_1,1)=α_3 Φ (P0,π_2,1)=α_4 Φ (P0,π_3,1)=α_5	$\Xi(S)=\{ \alpha_6,\alpha_7\}$ $\Xi(X)=\{ \alpha_3,\alpha_4, \alpha_5\}$ $\Xi(Y)=\{ \alpha_1,\alpha_2\}$ $\Theta(S)=\{ \alpha_6,\alpha_7\}$ $\Theta(X)=\{\alpha_3,\alpha_4, \alpha_5\}$ $\Theta(Y)=\{\alpha_1,\alpha_3, \alpha_8\}$
$\Sigma(P1 \mid \alpha_1)= \alpha_3$ $\Sigma(P1 \mid \alpha_2)= \alpha_4$	$\Pi(P1\|\alpha_3,\alpha_1)=\pi_4$, $\Pi(P1\|\alpha_3,\alpha_2)=\pi_5$ $\Pi(P1\|\alpha_4,\alpha_1)=\pi_5$, $\Pi(P1\|\alpha_4,\alpha_2)=\pi_6$ $\Pi(P1\|\alpha_5,\alpha_1)=\pi_5$, $\Pi(P1\|\alpha_5,\alpha_2)=\pi_5$	Φ (P1,π_4,1)=α_1 Φ (P1,π_5,1)=α_8 Φ (P1,π_6,1)=α_2	
$\Sigma(P2 \mid \alpha_1)= \alpha_5$ $\Sigma(P2 \mid \alpha_2)= \alpha_5$	$\Pi(P2\|\alpha_3,\alpha_1)=\pi_5$, $\Pi(P2\|\alpha_3,\alpha_2)=\pi_5$ $\Pi(P2\|\alpha_4,\alpha_1)=\pi_5$, $\Pi(P2\|\alpha_4,\alpha_2)=\pi_5$ $\Pi(P2\|\alpha_5,\alpha_1)=\pi_5$, $\Pi(P2\|\alpha_5,\alpha_2)=\pi_5$	Φ (P2,π_5,1)=α_8	
$\Sigma(P3)= \alpha_1$	$\Pi(P3\|\alpha_1)=\pi_7$, $\Pi(P3\|\alpha_2)=\pi_8$ $\Pi(P3\|\alpha_8)=\pi_8$		
$\Sigma(P4)= \alpha_2$	$\Pi(P2\|\alpha_1)=\pi_8$, $\Pi(P2\|\alpha_2)=\pi_8$ $\Pi(P2\|\alpha_8)=\pi_8$		

donde:

$\pi1$= {<a,b,,j>, <c,d,k>}, $\pi2$= {<a,d,k>,<b,c,j>}, $\pi3$= {<a,b,c,d,j,k>}, $\pi4$= {<a,b,e,f>, <c,d,g,h>}, $\pi5$={<a,b,c,d,e,f,g,h>}, $\pi6$={<a,d,e,h>, <b,c,f,g>}, $\pi7$= {<e,f>, <g,h>}, $\pi8$={<e,f,g,h>}.

$\alpha1$= {<e,f>, <g,h>}, $\alpha2$= {<e,h>, <f,g>}, $\alpha3$= {<a,b>, <c,d>}, $\alpha4$= {<a,d>, <c,b>}, $\alpha5$={<a,b,c,d>}, $\alpha6$={<j>, <k>}, α={<j,k>} , $\alpha8$= {<j>, <k>}

planes seleccionados para el ejemplo 1) (suponiendo ω=<j,k>)	planes seleccionados para el ejemplo 2) (suponiendo ω=<j,k>)
Plan[P0]=proy(<a,b,j,c,d,k>,π_1)={<a,b,j>,<c,d,k>} Plan[P1]=proy(<a,e,f,b,c,g,h,d>,π_4)={<a,e,f,b>, <c,g,h,d>} Plan[P3]=proy(<e,f,g,h>,π_7)={<e,f>,<g,h>}	Plan[P0]=proy(<c,b,j,a,d,k>,π_3)={<c,b,j,a,d,k>} Plan[P1]=proy(<c,g,f,b,a,e,g,h,d>,π_4)={<c,g,f,b,a,e,g,h,d>} Plan[P3]=proy(<g,f,e,h>,π_7)={<g,f,e,h>}

8. Conclusiones

Se ha presentado un trabajo que utiliza y combina algoritmos descriptos en [5] y [6] para la generación (estática) de evaluadores concurrentes de atributos para la clase de gramáticas de atributos NC(1). El particionado se realiza en base al análisis estático de las dependencias entre los atributos. Se presentan, además, detalles no desarrollados en [5], como la combinación de la selección de planes de evaluación en base a la particiones viables seleccionadas para cada producción. El evaluador tiene como costo adicional dos recorridos del árbol sintáctico (orden lineal con respecto al tamaño del árbol) pero con la ventaja que no hay ningún tipo de sobrecarga para los procesos evaluadores ya que no necesitan ningún tipo de sincronización y/o comunicación.

El enfoque de la combinación del algoritmo de particionado con el de generación de planes de secuencias de visitas puede ser fácilmente extendido a las gramáticas de atributos NC(∞) como se describe en [5].

Referencias

[1]Knuth, D.E., "Semantics of context−free languages", *Mathematical System Theory*, Vol. 2. No. 2. pp. 127−145, June 1968.

[2]U. Kastens, "Ordered attribute grammars", *Bericht No. 7/78, Universität Karlsruhe*, 1978.

[3]U.Kastens, "Implementation of visit−oriented attribute evaluators", *Procceedings of the International Summer School on Attribute Grammars, Applications and Systems,* Lecture Notes in Computer Science, 545:114−139. Springer−Verlag, 1991.

[4]K.Kennedy and S.Warren, "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, Ga., Jan. 19−21, 1976), ACM, New York, January 1976.

[5]Wuu Yang, "A finest partitioning algorithm for attribute grammars", *Second Workshop on Attribute Grammars an their Applications − WAGA99*, March 1999.

[6]Wuu Yang, "A Classification of Non−Circular Attribute Grammars Based on the Look−Ahead Behavior", *National Chiao−Tung University*, 1999.

[7]Uwe Kastens, "Implementation of Visit−Oriented Attribute Evaluators", *University of Paderborn*.

[8]Jukka Paakki, "Attribute Gramars Paradigms − A High−Level Methodology in Language Implementation", *ACM Computing Surveys*, Vol. 27, Nro 2, pp. 196−255. June 1995.

[9]Martin Jourdan, "A Survey of Parallel Attribute Evaluation Methods", Attibute Grammars, Aplications and Systems, June 1991, pp. 234−255 in *Lecture Notes in Computer Science*, Springer−Verlag, 1991.

[10]J. Saraiva, P. Henriques, "Concurrent Attribute Evaluation", *Dto. Informática, Universidade do Minho,* Braga, Portugal. April 1993.