magGen: Un Generador de Evaluadores de Gramáticas de Atributos Multiplanes

Marcelo Arroyo; Gerardo Kilmurray y Gonzalo Picco*

Resumen

Las *Gramática de Atributos* (GA) son un formalismo que poseen el poder descriptivo de las Gramáticas Libres de Contexto (CFG) y la expresividad de los lenguajes funcionales, para definir la semántica de un lenguaje.

Las ecuaciones de una GA inducen dependencias entre los atributos que ocurren en la producción. Si una GA contiene dependencias circulares no podrá ser evaluada, ya que no existirá un orden de evaluación consistente. Esto se conoce como el *Problema de Circularidad*.

El test de circularidad es intrínsecamente exponencial, lo que dificulta la implementación de evaluadores eficientes, por lo que comúnmente las herramientas imponen restricciones en las dependencias.

En 1998, Wuu Yang caracteriza una nueva familia de GA, denominada *Gramáticas de Atributos Multiplanes* (MAG) o NC(1), que permiten una mayor expresividad, presentando un algoritmo eficiente para su evaluación.

Este trabajo describe **magGen**: un generador de evaluadores estáticos para la familia MAG.

1. Introducción

Desde que Knuth introdujo, en 1966, las Gramáticas de Atributos (GA)[9], éstas se han utilizado ampliamente para el desarrollo de herramientas de procesamiento de lenguajes formales, como compiladores, intérpretes, traductores, así como también para especificar la semántica de lenguajes de programación.

Actualmente, existen muchas herramientas basadas en gramáticas de atributos pero generalmente se basan en familias más restrictivas como las absolutamente no circulares (ANCAG) o las ordenadas definidas por Uke Kastens (OAG)[6]. Estas familias se caracterizan por requerir que exista un único orden (plan) de evaluación por cada producción.

Las GA multiplan pueden tener varios planes asociados a una producción ya que las discrimina en base a sus contextos de aplicación. El plan correspondiente de una instancia de una producción en un árbol sintáctico se debe seleccionar dinámicamente en base a su contexto.

En este trabajo se describe una implementación de **magGen**(**m**ultiplan **a**ttribute **g**rammar **gen**erator), las optimizaciones desarrolladas sobre las ideas

*{marroyo,gkilmurray,gpicco}@dc.exa.unrc.edu.ar. Departamento de Computación, FCFQyN, U.N.R.C.

propuestas por Wuu Yang[1] y algunas medidas de desempeño del generador y de los evaluadores generados.

En las primeras secciones se introducen los conceptos fundamentales sobre las gramáticas de atributos (GAs) y métodos de evaluación. En las secciones subsiguientes se describe el desarrollo de **magGen**, los detalles más relevantes de su implementación, algunas medidas de rendimiento obtenidas y finalmente las conclusiones y trabajo futuro.

2. Gramáticas de atributos

En una gramática de atributos, se relaciona con cada símbolo de una gramática libre de contexto con un conjunto de atributos y con cada poducción un conjunto *de reglas semánticas* o *ecuaciones*, las cuales definen valores de los atributos en el contexto de la producción.

Una GA permite definir la sintaxis y semántica de un lenguaje de una manera declarativa y se usan ampliamente para el desarrollo de procesadores de lenguajes, como compiladores, intérpretes y herramientas de análisis de código.

Definición 2.1 *Una gramática libre de contexto es una tupla* $CFG =< V, T, S, P >$ *donde* V *es el conjunto de no terminales,* T *el conjunto de terminales,* $\Sigma = V \cup T$*, es el conjunto de símbolos,* $S \in V$ *es el símbolo inicial y* P *es el conjunto de producciones de la forma* $(X_0, X_1, \ldots, X_k)$*, donde* $X_0 \in V$ *se denomina la parte izquierda (left hand side, lhs) y* $X_1, \ldots, X_k$ *(*$X_i, 1 \leq i \leq k$*) es la parte derecha (rhs). Las produciones se denotan comúnmente de la forma* $X_0 \rightarrow X_1, \ldots, X_k$

Definición 2.2 *Dada una gramática* $G =< V, T, S, P >$*, se define* $ST = (K, D)$ *como un* árbol de derivación*, o Parse Tree, donde* K *es un conjunto de nodos y* D *es una relación no reflexiva sobre* K*, si y sólo si cumple las siguientes condiciones:*

- $K \subseteq (V \cup T \cup \lambda)$
- $k_0 = S$
- $k_i \in K$ *y* $(k_0, k_i) \in D$*, si* $S \rightarrow k_1 \ldots k_n \in P$
- *Si un nodo* $k \in T \cup \{\lambda\}$*, entonces* k *es una* ***hoja*** *de* ST
- *Si* $k_i \in V$*,* $(1 \leq i \leq n)$*, entonces* k_i *es la raíz del sub-árbol sintáctico para la* $CFG < V_i, T_i, k_I, P_i >$*, donde* V_i*,* T_i *y* P_i *son los símbolos terminales, no-terminales y producciones alcanzables desde las producciones* p *cuyo* $lhs(p) = k_i$*. En este caso,* k_i *se denomina* ***nodo interior****.*

Definición 2.3 *Una gramática de atributos es una tupla* $GA =< G, A, V, Dom, F, R >$ *donde:*

- $G =< V_N, V_T, S, P >$ *es una CFG reducida y no ambigua.*
- $A = \bigcup_{X \in \Sigma} A(X)$*, es el conjunto finito de atributos (*$A(X)$ *es el conjunto de atributos asociados al símbolo* X*).*

$A = S \cup H$, $S \cap H = \emptyset$. *S es el conjunto de atributos* ***sintetizados*** *y H es el conjunto de atributos* ***heredados***[1].

- *V es el conjunto finito de dominios de valores de los atributos.*
- $Dom : A \rightarrow V$ *asocia a cada atributo un dominio o conjunto de valores* $d \in V$.
- *F es un conjunto finito de funciones semánticas de la forma:*

$$f \subseteq (\bigotimes_{j=0}^{k} Dom(a_j)) \rightarrow Dom(a_0) \tag{1}$$

- $R = \bigcup_{p \in P} R^p$ *es el conjunto finito de reglas de atribución o ecuaciones asociadas a cada producción p de P, donde*

$$R^p = \bigcup_{j=0}^{m^p} \{r_j^p\} \tag{2}$$

y cada regla $r_j^p \in R^p$, *con* $0 \leq j \leq m^p$ *es de la forma:*

$$r_j^p : X_0.a_0 = f(X_1.a_1, \ldots, X_k.a_k) \tag{3}$$

donde cada X_i *es un símbolo que ocurre en la producción p ,* $a_i \in A(X_i)$, $(0 \leqslant i \leqslant k)$ *y* $f \in F$.

En una ecuación como (3), $X_i.a_i \rightarrow X_0.a_0$, *es decir,* $X_0.a_0$ ***depende*** *de* $X_i.a_i$ $(1 \leq i \leq k)$.

Se utilizará la notación $X.a$ para denotar que $a \in A(X)$.

Definición 2.4 *Una GA es* ***bien definida*** *si cada producción* $p : X_0 \rightarrow X_1 \ldots X_k$, R^p *contiene ecuaciones que definen los atributos* $a \in S(X_0)$ *y todos los atributos* $b \in H(X_i)$, $(i > 0)$.

Definición 2.5 *Un árbol sintáctico en el cual a cada nodo se ha extendido para incluir los valores de los atributos del símbolo correspondiente, se denomina* ***árbol atribuído***.

Definición 2.6 *El árbol atribuido, sobre el cual los valores de los atributos en cada nodo ya se han definido mediante un proceso de evaluación, se denomina* ***árbol decorado***.

Definición 2.7 *El grafo de dependencias directas de una producción p,*
$DP(p) = \{(X_i.a, X_j.b) \mid X_j.b \rightarrow X_i.a \ (X_i.a \text{ depende de } X_j.b) \text{ en } R^p\}$

Definición 2.8 *Para un árbol sintáctico* $T(GA)$, *su grafo de dependencias* $GD(T)$ *es el grafo construido a partir de la composición de los grafos de dependencias* $GD(p)$ *de cada instancia de producción en T.*

[1]Intituivamente, los atributos heredados transmiten valores en forma descendente en un árbol atribuido, mientras que los sintetizados lo hacen en forma ascendente.

2.1. Evaluación de gramáticas de atributos

Cualquier evaluador de GAs deberá respetar el orden de evaluación de los atributos impuesto por las dependencias.

Definición 2.9 *Una secuencia (o plan) de evaluación de atributos es consistente con las dependencias de los atributos de la GA si y sólo si para cada ecuación* $r_j^p : X_0.a = f(\ldots, X_i.b, \ldots)$ *en una producción* $p \in P$*, la definición de* $X_i.bi$ *deberá preceder a* $X_0.a$*.*

Definición 2.10 *Una gramática de atributos GA es "**circular**" si y sólo si existe un árbol sintáctico atribuido* $T(GA)$*, tal que su grafo de dependencias* $GD(T)$ *contiene al menos un ciclo.*

Si una GA contiene dependencias circulares no será posible encontrar un orden de evaluación consistente. Esto se conoce como el "*problema de la circularidad*", el cual se ha demostrado ser intrínsecamente exponencial[8]. El propósito de muchas investigaciones, ha sido descubrir nuevas familias o subgrupos de GAs, para las cuales puedan detectarse circularidades con algoritmos de menor complejidad.

En 1998, Wuu Yang en [2] caracteriza una nueva jerarquía de la GAs, denominadas **NC(n)**[2].

magGensoporta la familia MAG NC(1), la cual contiene a la familia de las OAG presentadas por Uwe Kastens[7] en 1980, presentando un algoritmo de evaluación basado en secuencias de visita. El algoritmo es exponencial en el numero de símbolos y producciones, lo cual lo hace un problema perfectamente tratable.

Los métodos de evaluación estáticos deben tener en cuenta todos los árboles sintácticos posibles a ser generados por la gramática y calcular las dependencias entre las instancias de los atributos, para cada uno de ellos. Esta información, calculada en tiempo de generación del evaluador, permite que la evaluación sea muy eficiente ya que no es necesario detectar circularidades ni generar los órdenes de evaluación en tiempo de ejecución.

Un árbol sintáctico se construye a partir de la aplicación sucesiva de producciones de la gramática. Una instancia de una producción en un árbol sintáctico tiene como *contexto inferior* a las instancias de las producciones aplicadas a los símbolos no terminales de su parte derecha y, como *contexto superior* a la instancia de la producción a partir del cual se ha construido dicho árbol.

Dada una producción p se deben considerar tres tipos de dependencias que definen el contexto de la misma: Las directas, obtenidas por las ecuaciones de p y las de los contextos superior e inferiores.

Instancias diferentes de una producción p tendrán las mismas dependencias directas, pero podrán tener diferentes dependencias impuestas por el contexto. Es necesario entonces, determinar todas las *dependencias posibles* entre los atributos que ocurren en una producción para luego poder determinar todos los posibles *planes o secuencias de evaluación posibles* entre los atributos que ocurren en una producción.

[2]Non Circular AG basadas en Lookahead behaviour.

Una vez determinadas las secuencias de evaluación, hay que considerar que las instancias de atributos se evalúen en el contexto de su definición.

Para eso las secuencias se traducen a *secuencias de visita*, las cuales son tres operaciones introducidas por Uwe Kastens en [7]:

visit(child,i) indica que el evaluador debe moverse (visitar) el nodo hijo *child* de n y corresponde a la *i-ésima* visita al nodo hijo.

compute(at) indica que debe evaluarse la ecuación que define at en la producción p aplicada correspondiente al nodo n.

leave indica que deberá retornar al nodo padre, finalizando la visita corriente.

Aquí se presenta una leve modificación de las secuencias de visita en [7], principalmente en las operaciones *visit* y *leave*. Si el evaluador recuerda en cada nodo del árbol atribuído el punto de la secuencia hasta donde de evaluó en la visita anterior, se puede prescindir del parámetro en estas dos operaciones.

La evaluación termina cuando se ejecutaron todas las operaciones de la secuencia de visita del nodo raíz del árbol sintáctico.

Estos evaluadores pertenecen a una familia denominada *Evaluadores Multivisita*, ya que el proceso de evaluación puede requerir múltiples visitas a un nodo.

2.2. Gramáticas de Atributos Multiplanes

Para abordar la presentación y definición de las Gramáticas de Atributos Multiplanes es necesario introducir algunos conceptos previamente.

Definición 2.11 *Las dependencias transitivas entre los atributos del símbolo X en subárboles a partir de la producción p de la forma $X \rightarrow \ldots$ se denomina* ***Downward Characteristic Graph****, denotado como $DCG_X(p)$.*

Definición 2.12 *El* ***conjunto de dependencias aumentadas de una producción q con contexto inferior*** *$p_1, \ldots, p_k$:*

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

tal que $q \rightarrow p_1 \ldots p_k \in P$.

Cada grafo $ADP(q|p_1, p_2, \ldots, p_k)$ representa las dependencias entre los atributos de los símbolos que aparecen en un árbol que tenga como raíz una instancia del símbolo de la parte izquierda de la producción $q \rightarrow \alpha X_1 \ldots X_k \beta$ en que se habrían aplicado las producciones $p_1 : X_1 \rightarrow \psi_1$, ..., $p_k : X_K \rightarrow \psi_k$ en su contexto (subárbol) inferior inmediato.

El conjunto de todas las posibles dependencias aumentadas para una producción q se define como:

$$SADP(q) = \{ADP(q|p_1, p_2, \ldots, p_k) \mid \forall p \rightarrow p_1, \ldots, p_k \in P\}$$

A modo de ejemplo, en la GA de la figura 1, los grafos

$$DP(p_1) = \{(s1, i2), (s2, i3), (s3, i1), (s0, s1), (s0, s2), (s0, s3), (s0, s4)\}$$
$$DCG_X(p4) = \{(i1, s1)\},\ (X.s1 \text{ depende de } X.i1)$$
$$ADP(p_1 \mid p_4, p_2, p_5) = DP(p_1) \cup DCG_X(p_4) \cup DCG_Y(p_2) \cup DCG_Z(p_5)$$

Definición 2.13 *Una gramática de atributos G es una gramática de atributos multiplanes si y solo si* $\forall q \in P : (\forall g :\in SADP(q) : g \text{ es no circular})$

Esta definición tiene en cuenta contextos inferiores (subárboles) de sólo un nivel, lo cual caracteriza a la familia NC(1) de Wuu Yang. En [2], Wuu Yang extiende la idea para considerar hasta m contextos inferiores, denominadas $NC(m)$ y hasta todos los posibles contextos ($NC(\infty)$). Esta última familia coincide con las GA no circulares.

3. Desarrollo de magGen

El desarrollo de **magGen** se realizó en 4 etapas:

a) definición del Lenguaje especificación de MAG, b) desarrollo del parser del lenguaje, representación interna y chequeos, c) construcción de grafos y aplicación de algoritmos de cómputo de planes y secuencias de visita y d) generación de código del evaluador.

3.1. El lenguaje de especificación de magGen

El lenguaje define secciones para la declaración del dominio semántico, donde se definen los tipos y perfiles de las funciones y operadores a usar, la definición de los atributos de cada símbolo y el bloque de definición de las producción de la gramática con sus ecuaciones correspondientes.

```
1  semantic domain
2    op infix (10, left) +: int, int -> int;
3  attributes
4    s0: syn <int> of {S};
5    s1: syn <int> of {X};
6    s2: syn <int> of {Y};
7    s3: syn <int> of {Y};
8    s4: syn <int> of {Z};
9    i1: inh <int> of {X};
10   i2: inh <int> of {Y};
11   i3: inh <int> of {Y};
12
13 rules
14   p1: S ::= X Y Z
15       compute
16         S[0].s0 = X[0].s1+Y[0].s2+Y[0].s3+Z[0].s4;
17         X[0].i1 = Y[0].s3;
18         Y[0].i2 = X[0].s1;
19         Y[0].i3 = Y[0].s2;
20       end;
21   p2: Y ::= 'm'
22       compute
23         Y[0].s2 = Y[0].i2;
24         Y[0].s3 = 1;
25       end;
26   p3: Y ::= 'n'
27       compute
28         Y[0].s2 = 2;
29         Y[0].s3 = Y[0].i3;
30       end;
31   p4: X ::= 'm'
32       compute
33         X[0].s1 = X[0].i1;
34       end;
35   p5: Z ::= Y
36       compute
37         Z[0].s4 = Y[0].s3;
38         Y[0].i2 = 3;
39         Y[0].i3 = Y[0].s2;
40       end;
```

Figura 1: MAG en lenguaje de especificación de **magGen**.

La figura 1 muestra un ejemplo de especificación de una GA.

4. Detalles de implementación

El parser del lenguaje se apoya en la utilización de un framework reconocido mundialmente, denominado ***Spirit***, perteneciente a la biblioteca de C++ llamada ***Boost***[10]. Esta decisión trajo dos grandes beneficios; la confiabilidad del parser obtenido y la rápida obtención del mismo, ya que, la gran ventaja de ***Spirit*** es permitir escribir la definición de la gramática en **C++** directamente

sin tener que recurrir a herramientas externas.

La herramienta verifica que la GA de entrada esté bien definida y genera el evaluador, los grafos de dependencias (para depuración) y los planes generados.

El chequeo de ciclicidad sobre los grafos ADP, se implementó utilizando el algoritmo de búsqueda en profundidad (Depth-first search) de ***Boost*** combinado con la creación de un "visitador" especializado, el cual a medida que recorre el grafo, va recordando los nodos y aristas visitadas mientras no se halla detectado un ciclo. Esos subgrafo se generan (en forma de archivos gráficos) al usuario en caso de error, para que pueda corregirse el error.

4.1. Construcción de planes

Los planes de evaluación son órdenes parciales computados a partir del orden topológico de cada $ADP(q \mid p_1, \ldots, p_k)$ y a partir de un orden de evaluación requerido para los atributos del símbolo de la parte izquierda de q.[3]

Para el ejemplo presentado en la figura 1 se deben computar 7 planes. Cada plan es una secuencia de enteros representando números de ecuaciones. A continuación se muestran algunos de los planes computados.

1. $< 6, 2, 9, 3, 5, 4, 10, 1 >$ = gen_plan$(\lambda |p_1| < p_4, p_2, p_5 >)$
2. $< 7, 4, 8, 2, 9, 3, 10, 1 >$ = gen_plan$(\lambda |p_1| < p_4, p_3, p_5 >)$
3. $< 7, 4, 8, 3 >$ = gen_plan$(proj(2, lhs(p_3))|p_3| <>)$
4. $< 2, 9 >$ = gen_plan$(proj(1, lhs(p_4))|p_4| <>)$
5. ...

La función gen_plan$(s|P|i)$ toma las restricciones del contexto superior (s), la regla (P) y el contexto inferior (secuencia de producciones), i.

La función $proj(p, s)$ proyecta un orden consistente de los atributos del símbolo s (parte izquierda de una producción), en base al plan p (plan de su contexto superior).

Se puede notar que los dos primeros planes corresponden a la producción p_1 (cada uno se corresponde con un diferente contexto inferior), el tercero a la producción p_3 en el contexto del plan 2 y el cuarto a la producción p_4 en el contexto del plan 1.

4.2. Construcción Secuencias de Visita

Cada plan se transforma en una secuencia de visitas, es decir una secuencia de operaciones *eval(eq)*, *visit(child)* y *leave*.

El algoritmo de generación de secuencias de visita es simple y lineal a partir de cada producción, un plan y los tipos (heredados o sintetizados) de los atributos de los símbolos que ocurren en la producción.

Para cada elemento i del plan:

[3] Lo que constituye una restricción en el orden de los atributos del $lhs(q)$ impuesto por el contexto superior de q.

- si la ecuación i define un atributo sintetizado del símbolo de la parte izquierda o un atributo heredado de un símbolo de la parte derecha de la producción, generar un $compute(i)$,
- si la ecuación i define un atributo heredado del símbolo de la parte izquierda de la producción, generar un $leave$,
- si la ecuación i define un atributo sintetizado del símbolo de la parte derecha de la producción, generar un $visit(j)$, donde j es el índice del nodo hijo correspondiente.

A modo de ejemplo, se muestra una secuencia de visita generada por **magGen**para el ejemplo de la figura 1:

el plan $< 7, 4, 8, 3 >$ se traduce en $< compute(7), leave, compute(8), leave >$, lo que hará que el evaluador realice la evaluación de la ecuación 7, luego retorne al nodo padre, ejecutará las secuencias de visita correspondientes para luego visitar nuevamente al primer nodo, computará la ecuación 8 y finalmente retornará al nodo padre. Se debe notar que antes de evaluar la ecuación 8 se require computar el valor de $Y.i3$, el cual se hará en el contexto superior.

El evaluador de atributos lanza la ejecución de las secuencias de visita asociados a la producción de comienzo, lo que demandará la evaluación de todos los atributos del árbol.

4.3. Generación de código

La etapa final de **magGen** esta dada por la generación de código del evaluador. Toda la lógica (plan seleccionado, ecuaciones) e información (atributos) del evaluador está codificada en la clase *Node*, el cual representa un nodo (polimórfico) de un árbol atribuído.

Su declaración e implementación está en los archivos `node.hpp` y `node.cpp`. Los planes se representan en la clase *plan* (plan.{hpp,cpp}).

4.4. Algoritmo de evaluación

El evaluador de un AST se basa en dos etapas:

1. `traverse`: selección del plan de evaluación para cada nodo del árbol en base a la instancia de la producción correspondiente al nodo y su contexto (superior e inferior). Este primer recorrido del árbol es descendente.
2. `evaluate`: es el intérprete de las secuencias de visitas. Inicia ejecutando la secuencia de visitas de la raíz, la cual conducirá las visitas y ejecución de las ecuaciones en el árbol.

5. Desempeño

En la etapa de prueba de **magGen** se han analizado una serie de medidas performance que permitieron observar números concretos sobre el funcionamiento de la herramienta. En la figura 3 se muestran los tiempos en la compilación

de la herramienta, teniendo en cuenta dos versiones de Spirit Boost[4]. Se detecta un incremento de performance con el uso de la Spirit Boost actual.

```
M ::= E
    M.valor = E.valor
E ::= E '+' E
    E.valor = E.valor + E.valor
    | E '-' E
      E.valor = E.valor − E.valor
    | E '*' E
      E.valor = E.valor * E.valor
    | E '/' E
      E.valor = E.valor / E.valor
    | '(' E ')'
      E.valor = E.valor
    | '-'E
      E.valor = E.valor * (−1.0)
    | num
      E.valor = num.valor
num ::= digit num
          num.valor = (digit.valor * 10.0) + num.valor
    | real
          num.valor = real.valor
     | digit
          num.valor = digit.valor
```

```
digit ::= '1'
          digit.valor = 1.0
        | '2'
          digit.valor = 2.0
        | '3'
          digit.valor = 3.0
        | '4'
          digit.valor = 4.0
        | '5'
          digit.valor = 5.0
        | '6'
          digit.valor = 6.0
        | '7'
          digit.valor = 7.0
        | '8'
          digit.valor = 8.0
        | '9'
          digit.valor = 9.0
        | '0'
          digit.valor = 0.0
real ::= digit '.' digit
          real.valor = digit.valor + (digit.valor / 10.0)
```

Figura 2: GA de Expresiones.

En las figuras 4 se presentan medidas sobre los tiempos y tamaños logrados en el binario obtenido del evaluador generado por **magGen**.

Input / Spirit	**1.8.X**	**2.3**
`MAG Wuu Yang`	∼0.087 sec	∼0.084 sec
`MAG Aritmetica`	∼0.590 sec	∼0.450 sec

Figura 3: Medidas de performance: Versión de Spirit Boost

Input / g++	**tiempo**
`Wuu Yang`	1.51 s
`Aritmetica`	1,56 m

(a) Tiempos de compilación

Input / g++	**Kb**
`Wuu Yang`	48
`Aritmetica`	384

(b) Tamaños del ejecutable

Figura 4: Medidas para el evaluador generado

Estos resultados se desprenden de los ejemplos de las figuras 1 y 2.

Las pruebas realizadas también muestran que, en la práctica, a pesar que una producción pueda aparecer en varios diferentes contextos en un árbol, los planes de evaluación de diferentes contextos coinciden, por lo cual no se espera que una producción tenga asociado un gran número de planes.

Por ejemplo, para el ejemplo de la figura 2 se tienen en cuenta 371 contextos posibles mientras que se generan sólo 22 secuencias de visita, lo que muestra que los evaluadores generados en la práctica serán pequeños y eficientes.

6. Conclusiones y trabajo futuro

Se logró obtener una herramienta modularizada, eficiente y completamente desarrollada en C++, lo cual muestra que es posible implementar evaluadores

[4]Se uso la mínima versión compatible (Boost 1.37) y la actual versión (Boost 1.43).

estáticos compactos, legibles con un muy buen rendimiento para familias menos restrictivas de GAs que las usadas comúnmente en otras herramientas.

Se está trabajando en extensiones como:

- Desarrollo de bibliotecas de funciones de utilidad de uso común en especificaciones, como manejo de ambientes, algoritmos de análisis estático (interpretación abstracta) y generación de código.
- Extensión de especificaciones, al estilo de las GA orientadas a objetos con mecanismos como redefinición o delegación (forwarding) de reglas para permitir un lenguaje de especificación más modular.
- Introducción de sentencias de acceso no local de atributos para simplificar las especificaciones.

magGen se encuentra disponible en http://genevalmag.googlecode.com/

Referencias

[1] Wuu Yang. 1998. *Multi-Plan Attribute Grammars*. Department of Computer Information Science. National Chiao-Tung University, Hsin-Chu, Taiwan. 2

[2] Wuu Yang. 1999. *A Classification of Non Circular Attribute Grammars based on Lookahead behavior*. Department of Computer and Information Science. National Chiao-Tung University, Hsin-Chu, Taiwan. 4, 6

[3] Wuu Yang, W. C. Cheng. *A Polynomial Time Extension to Ordered Attribute Grammars*. Department of Computer and Information Science.National Chiao-Tung University, Hsin-Chu, Taiwan.

[4] John E. Hopcroft, Rajeev Montwani, Jefrey D. Ullman.*Introduction to Automata theory, languajes, and computation*. Addison-Wesley (2001) second edition.

[5] Alfred V. Aho, Ravi Sethi, Jeffrey D. ullman.*Compilers: Principles, Techniques and Tools*. Addison-Wesley (1985) Iberoamericana, S.A. Wilmington, Delaware, E.U.A.

[6] Arroyo, Marcelo Daniel. *Gramáticas de atributos, clasificación y aportes en técnicas de evaluación*. Tesis de carrera de Magister en Ciencias de la Computación. Universidad Nacional del Sur. Bahía Blanca - Argentina. 2008. 1

[7] U. Kastens. 1980. *Ordered Attribute Grammars*. Acta Informatica. Vol. 13, pp. 229-256. 4, 5

[8] Jazayeri, Ogden and Rounds. 1975. *The intrinsically exponential complexity of the circularity problem for attribute grammars*. Comm. ACM 18. December 2, Pag: 697-706. 4

[9] D. Knuth. 1968. *Semantics of context free languages*. Math Systems Theory 2.June 2.Pag: 127-145. 1

[10] *Boost C++ Libraries*. URL: http://www.boost.org/ 6