

magGen

Generador de Evaluadores Estáticos para MAG

Gerardo Luis Kilmurray - Gonzalo Martín Picco

Departamento de Computación
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Universidad Nacional de Río Cuarto

5 de septiembre de 2010



Temario

Temario

- 1 ¿Qué es magGen?
 - Motivación
 - Objetivos
- 2 Preliminares
 - Gramáticas de Atributos Multiplanos (MAG)
 - Evaluador estático
 - Planes de evaluación
 - Secuencias de visita
- 3 magGen
 - Herramientas usadas
 - Funcionamiento
 - Lenguaje de Especificación
 - Análisis estático
 - Generación de código
 - Usando magGen
- 4 Evaluador generado
- 5 Performance
- 6 Comentarios Finales



Temario

¿Qué es magGen?

- 1 ¿Qué es magGen?
 - Motivación
 - Objetivos
- 2 Preliminares
 - Gramáticas de Atributos Multiplanos (MAG)
 - Evaluador estático
 - Planes de evaluación
 - Secuencias de visita
- 3 magGen
 - Herramientas usadas
 - Funcionamiento
 - Lenguaje de Especificación
 - Análisis estático
 - Generación de código
 - Usando magGen
- 4 Evaluador generado
- 5 Performance
- 6 Comentarios Finales



¿Qué es magGen?

¿Qué es magGen?

magGen es una herramienta generadora de evaluadores estáticos para la familia MAG.

mag + Gen

Multi-plans Attribute Grammar - Generator

Características destacables:

- Funciona sobre la familia MAG.
- La construcción de planes mantiene la propiedad de “unicidad de planes”.
- Construcción de planes solo para reglas alcanzables.
- Generación de Evaluadores estáticos.
- La generación de código produce una representación compacta de cada estructura.



Motivación

En las ciencias de la computación los lenguajes juegan un rol muy importante en muchas disciplinas. Desde los comienzos se buscaron *mecanismos* para describirlos y manejarlos.

D. Knuth introdujo en 1966 las **gramáticas de atributos** (GA), estas se han utilizado ampliamente para el desarrollo de herramientas de procesamiento de lenguajes formales y, además, para especificar la semántica de lenguajes de programación.

En 1998, Wu Yang caracteriza una nueva familia denominada **Gramática de Atributos Multiplanos**.

Al momento no existían herramientas basadas en esta familia de Gramáticas de Atributos



Objetivos

Estudio de la familia MAG.

Objetivo Principal

Desarrollar una herramienta en C++ que genere evaluadores estáticos.



Temario

- 1 ¿Qué es magGen?
 - Motivación
 - Objetivos
- 2 Preliminares
 - Gramáticas de Atributos Multiplanos (MAG)
 - Evaluador estático
 - Planes de evaluación
 - Secuencias de visita
- 3 magGen
 - Herramientas usadas
 - Funcionamiento
 - Lenguaje de Especificación
 - Análisis estático
 - Generación de código
 - Usando magGen
- 4 Evaluador generado
- 5 Performance
- 6 Comentarios Finales



Notación

Gramática de atributos (Informal)

En una **Gramática de Atributos** (GA), se relaciona cada símbolo de una *Gramática Libre de Contexto* con un conjunto de atributos. Cada regla o producción tiene asociado un *conjunto de reglas semánticas*, denominadas también *ecuaciones*.

Instancia: Ocurrencia de un símbolo de la gramática asociado a un atributo.



Ejemplo GA

```

(R1)  S → XYZ
      S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
      X.i1 := Y.s3
      Y.i2 := X.s1
      Y.i3 := Y.s2

(R2)  Y → m
      Y.s2 := Y.i2
      Y.s3 := 1

(R3)  Y → n
      Y.s2 := 2
      Y.s3 := Y.i3

(R4)  X → m
      X.s1 := X.i1

(R5)  Z → Y
      Z.s4 := Y.s3
      Y.i2 := 3
      Y.i3 := Y.s2

```

Ejemplo de instancia

Y.i3 \Leftrightarrow Ocurrencia del símbolo Y asociado al atributo i3.



Gramáticas de Atributos Multiplanos

Dependencias directas de una producción

Se denota como $DP(p)$

$$DP(p) = \{(X_i.a, X_j.b) | X_i.a \rightarrow Y_j.b \in R^p\}$$

Dado un símbolo X de una gramática de atributos

$$\text{Down}(X) = \{(a, b) | a \rightarrow b\} \text{ con } a, b \in A(X)\}$$



Gramáticas de Atributos Multiplanos (cont.)

Conjunto de dependencias aumentadas

Sea q una producción de la forma $X_0 \rightarrow \alpha_0 X_1 \alpha_1 X_2 \dots X_k \alpha_k$. Sea p_i una producción cuya parte izquierda es X_i ($1 \leq i \leq k$).

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

se denota como $ADP(q | p_1, p_2, \dots, p_k)$

$DCG_X(p)$ contiene las dependencias, entre las instancias de la gramática, para el símbolo X, acotando el análisis para la producción p y los posibles contextos inferiores.

$$\bigcup_{\text{todo } p} DCG_X(p) = \text{Down}(X)$$



Gramáticas de Atributos Multiplanos (cont.)

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

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

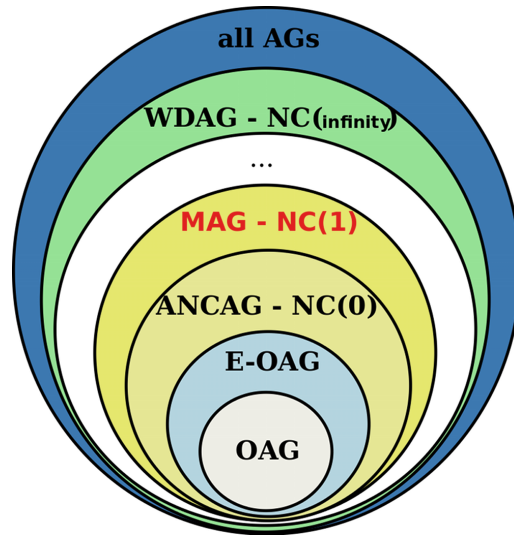
Gramática de Atributos Multiplanos (MAG)

Una gramática G es una *Gramática de Atributos Multiplanos* si y solo si

$$\forall q : q \in P : (\forall g : g \text{ es un grafo de } q \wedge g \in SADP(q) : g \text{ es no circular})$$



Familias de gramáticas de atributos



Evaluador estático

Los métodos 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.

Contexto de una producción

Tres tipos de dependencias definen el *contexto* de una producción:

- Directas obtenidas por las ecuaciones de p .
- Impuestas por el contexto superior.
- Impuestas en el contexto inferior.

Instancias diferentes de una producción p tendrán las mismas dependencias directas, pero podrán tener diferentes dependencias impuestas por los contextos inferiores y superiores.



Planes de evaluación

Luego del análisis de las dependencias, los grafos ADP acíclicos inducen un orden entre las ecuaciones que deben computarse.

El proceso de construcción de planes, considera el tres factores:

- Orden inducido por las dependencias.
- Orden superior para la evaluación de las instancias.
- El contexto de aplicación.

Plan de Evaluación

Un plan de evaluación esta dado por un orden consistente con las dependencias para la computación de las ecuaciones que definen a cada instancia de la gramática.



Secuencias de Visita

El concepto de *Secuencias de Visita* fue presentado en 1980, por **Uwe Kastens** como un método de evaluación polinomial para la familia OAG.

Su funcionamiento se basa en la visita de los nodos de los árboles de derivación para lograr la evaluación del árbol. Este método se apoya en los planes de evaluación de la gramática.

Los evaluadores basados en secuencias de visita pertenecen a una familia denominada **Evaluadores Multivisita**, ya que el proceso de evaluación puede requerir múltiples visitas a cada nodo para evaluarlo.



Secuencia de Visita (cont.)

Definición

Sea n un nodo de un árbol atribuido T . Una secuencia de visita en el nodo n es **una secuencia de tres operaciones o acciones**: $visit(child, i)$, $compute(at)$ y $leave(i)$.

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(i) indica que ha finalizado la visita i -ésima en el nodo corriente y que se debe visitar al nodo padre.



Temario

1 ¿Qué es magGen?

- Motivación
- Objetivos

2 Preliminares

- Gramáticas de Atributos Multiplanos (MAG)
- Evaluador estático
- Planes de evaluación
- Secuencias de visita

3 magGen

- Herramientas usadas
- Funcionamiento
- Lenguaje de Especificación
- Análisis estático
- Generación de código
- Usando magGen

4 Evaluador generado

5 Performance

6 Comentarios Finales



magGen: Herramientas usadas

- C++
- Boost C++ Libraries
 - Spirit Parser Framework
 - The Boost Graph Library (BGL)
- Subversion
- Brouml
- Cmake
- Eclipse
- kile
- \LaTeX 2_ε
- GraphViz
- Dia
- Nemiver + gbd
- Análisis estático de código:
 - CCCC
 - gcov



magGen: Funcionamiento

El funcionamiento de **magGen** esta dado por la integración de 4 etapas, consideradas principales, que marcaron el proceso de desarrollo de la herramienta:

- Lenguaje especificación de MAG.
- Parser del lenguaje, representación interna y chequeos.
- Construcción de grafos y aplicación de algoritmos de cómputo de planes y secuencias de visita.
- Generación de código.

El cómputo de **magGen** se realiza atravesando cada una de estas etapas secuencialmente, es decir, la terminación exitosa de una, habilita la siguiente; por lo tanto cada etapa mantiene su salida de errores de manera independiente.



magGen: Funcionamiento (cont.)

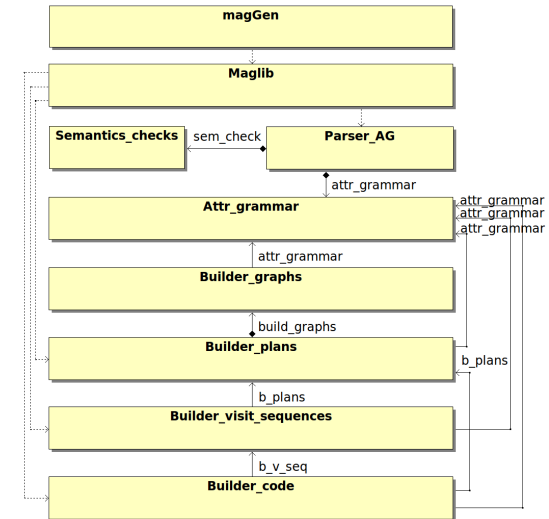
La salida normal de **magGen** que indica que se han realizado todas las etapas correctamente, es la siguiente:

```
* Parsing grammar      [ OK ]
* Generate graphs     [ OK ]
* Build plans         [ OK ]
* Build visit sequence [ OK ]
* Generation code     [ OK ]

Generation complete in: 0.372814 seconds.
```



Arquitectura



Lenguaje especificación de MAG

Especificación

Las secciones que conforman la descripción de una MAG, se corresponden con las características de una gramática de atributos.

```
semantic domain
  <sorts>
  <operators>
  <functions>
attributes
rules
  <rule0>
    <equations>
  ...
  <ruleN>
    <equations>
```



Gramática MAG presentada por Wu Yang

```

(R1)  S → XYZ
      S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
      X.i1 := Y.s3
      Y.i2 := X.s1
      Y.i3 := Y.s2
(R2)  Y → m
      Y.s2 := Y.i2
      Y.s3 := 1
(R3)  Y → n
      Y.s2 := 2
      Y.s3 := Y.i3
(R4)  X → m
      X.s1 := X.i1
(R5)  Z → Y
      Z.s4 := Y.s3
      Y.i2 := 3
      Y.i3 := Y.s2
```



Especificación en lenguaje de magGen

```

semantic domain
  op infix (10, left) +: int, int -> int;

attributes
  s0: syn <int> of {S};
  s1: syn <int> of {X};
  s2: syn <int> of {Y};
  s3: syn <int> of {Y};
  s4: syn <int> of {Z};
  i1: inh <int> of {X};
  i2: inh <int> of {Y};
  i3: inh <int> of {Y};

rules
  S ::= X Y Z
    compute
      S[0].s0 = X[0].s1 + Y[0].s2 + Y[0].s3 + Z[0].s4;
      X[0].i1 = Y[0].s3;
      Y[0].i2 = X[0].s1;
      Y[0].i3 = Y[0].s2;
    end;

```



Especificación en lenguaje de magGen (cont.)

```

Y ::= 'm'
  compute
    Y[0].s2 = Y[0].i2;
    Y[0].s3 = 1;
  end;
Y ::= 'n'
  compute
    Y[0].s2 = 2;
    Y[0].s3 = Y[0].i3;
  end;
X ::= 'm'
  compute
    X[0].s1 = X[0].i1;
  end;
Z ::= Y
  compute
    Z[0].s4 = Y[0].s3;
    Y[0].i2 = 3;
    Y[0].i3 = Y[0].s2;
  end;

```



magGen: Chequeos

Sobre la MAG de entrada se verifican:

- Precedencia y Asociatividad de operadores dentro de expresiones.
- Alcanzabilidad de símbolos y reglas.
- Consistencia de bloque de atributos y bloque de reglas.
- Gramática bien definida
 - Atributos sintetizados del símbolo de la parte izquierda.
 - Atributos heredados de símbolos de la parte derecha.
 - Ecuaciones sólo de los atributos de los símbolos de la regla.
 - Inexistencia de ecuaciones para atributos heredados del símbolo de la izquierda y para atributos sintetizados de símbolos de la parte derecha.
 - Múltiples ecuaciones para un atributo.
 - Consistencia de los índices de las instancias que aparecen en las ecuaciones.
 - Condiciones de una gramática extendida.



magGen: Generación de grafos

Luego de los chequeos semánticos sobre la gramática de entrada, se construyen los siguientes grafos:

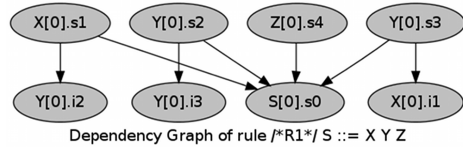
- Grafos **DP**: Dependencias directas de cada producción.
- Grafos **Down**: Dependencias entre atributos de cada símbolo.
- Grafos **DCG**: Dependencias entre atributos de cada símbolos acotado a una producción.
- Grafos **ADP**: Grafos de dependencias aumentadas.

La construcción de los grafos permite realizar chequeos de ciclicidad y servir de base (grafos ADP) para la construcción de planes de evaluación.



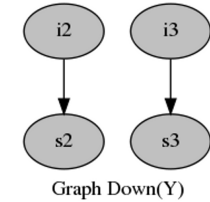
magGen: Grafo DP

```
(R1) S → XYZ
      S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
      X.i1 := Y.s3
      Y.i2 := X.s1
      Y.i3 := Y.s2
```



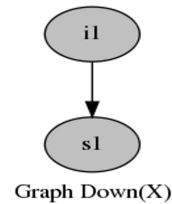
magGen: Grafo Down

```
(R1) S → XYZ
      S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
      X.i1 := Y.s3
      Y.i2 := X.s1
      Y.i3 := Y.s2
(R2) Y → m
      Y.s2 := Y.i2
      Y.s3 := 1
(R3) Y → n
      Y.s2 := 2
      Y.s3 := Y.i3
(R4) X → m
      X.s1 := X.i1
      → Y
(R5) Z → Y
      Z.s4 := Y.s3
      Y.i2 := 3
      Y.i3 := Y.s2
```



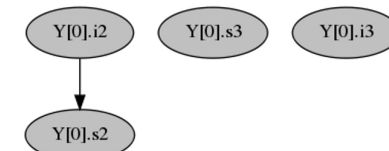
magGen: Grafo Down

```
(R1) S → XYZ
      S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
      X.i1 := Y.s3
      Y.i2 := X.s1
      Y.i3 := Y.s2
(R2) Y → m
      Y.s2 := Y.i2
      Y.s3 := 1
(R3) Y → n
      Y.s2 := 2
      Y.s3 := Y.i3
(R4) X → m
      X.s1 := X.i1
      → Y
(R5) Z → Y
      Z.s4 := Y.s3
      Y.i2 := 3
      Y.i3 := Y.s2
```



magGen: Grafo DCG

```
...
(R2) Y → m
      Y.s2 := Y.i2
      Y.s3 := 1
(R3) Y → n
      Y.s2 := 2
      Y.s3 := Y.i3
...
```

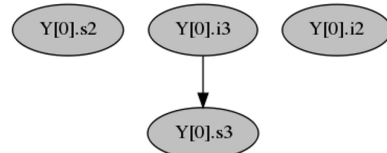


magGen: Grafo DCG

```

...
(R2) Y → m
      Y.s2 := Y.i2
      Y.s3 := 1
(R3) Y → n
      Y.s2 := 2
      Y.s3 := Y.i3
...

```



DCG Graph of rule /*R3*/ Y ::= 'n' with symbol Y

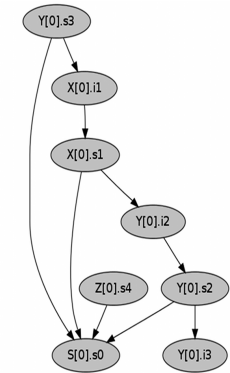


magGen: Grafo ADP

```

(R1) S → XYZ
      S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
      X.i1 := Y.s3
      Y.i2 := X.s1
      Y.i3 := Y.s2
(R2) Y → m
      Y.s2 := Y.i2
      Y.s3 := 1
(R3) Y → n
      Y.s2 := 2
      Y.s3 := Y.i3
(R4) X → m
      X.s1 := X.i1
(R5) Z → Y
      Z.s4 := Y.s3
      Y.i2 := 3
      Y.i3 := Y.s2

```



ADP Graph of rule /*R1*/ S ::= X Y Z with inferior context: R4, R2, R5.

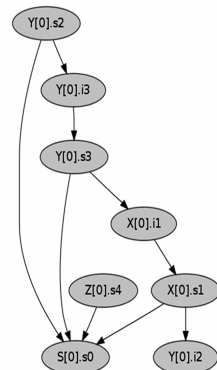


magGen: Grafo ADP

```

(R1) S → XYZ
      S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
      X.i1 := Y.s3
      Y.i2 := X.s1
      Y.i3 := Y.s2
(R2) Y → m
      Y.s2 := Y.i2
      Y.s3 := 1
(R3) Y → n
      Y.s2 := 2
      Y.s3 := Y.i3
(R4) X → m
      X.s1 := X.i1
(R5) Z → Y
      Z.s4 := Y.s3
      Y.i2 := 3
      Y.i3 := Y.s2

```



ADP Graph of rule /*R1*/ S ::= X Y Z with inferior context: R4, R3, R5.



Ciclicidad

- El chequeo de ciclicidad es realizado sobre los grafos ADP.
- Esta basado en el algoritmo de búsqueda en profundidad (Depth-first search) de **Boost** combinado con la creación de un "visitador" especializado.
- El subgrafo es mostrado al usuario en caso de detección de ciclo, sino es descartado.

La salida de **magGen** cuando se detecta ciclicidad, es:

```

* Parsing grammar _____ [ OK ]
* Generate graphs _____ [ OK ]
* Build plans _____ [ ABORT ]

```

ERROR: One o more graph ADP has an cycle in its dependencies↵
 . Look the folder ./out_maggen/graphs/CYCLIC_graphs/ for ↵
 more details.



magGen: Plan de evaluación

Los grafos ADP, generados, son la entrada para la construcción de **planes de evaluación**.

Unicidad

- Se mantiene un repositorio indexado para impedir elementos repetidos.
- Cada índice de plan referencia a una posición de un *vector de planes únicos*.
- Optimización notable tanto en tiempo como en espacio.

Algoritmo `compute_plans`

- Propuesto por Wu Yang en [1].
- Generación de planes y planes proyectados de evaluación.



Algoritmo `compute_plans`

- Se mantiene una cola de trabajos con pares de identificador de producción y un orden de evaluación de sus atributos.
- Cada vez que se saca un elemento de la lista, se marca como realizado.
- Se toma cada uno de los distintos grafos ADP asociados a la producción.
- Se combina la dependencia inducida por el orden con el grafo ADP.
- Se calcula la clausura transitiva del nuevo grafo.
- Se computa el orden topológico del grafo.
- Se guarda el plan si es diferentes a los ya generados.
- Se generan los planes proyectados para cada símbolo de la parte derecha.
- Se agregan a la cola de trabajos, el par con el símbolo y su plan proyectado.



magGen: Plan de evaluación (cont.)

Plan de evaluación

Lista de números de ecuaciones a computar, dirigiendo la evaluación de izquierda a derecha.

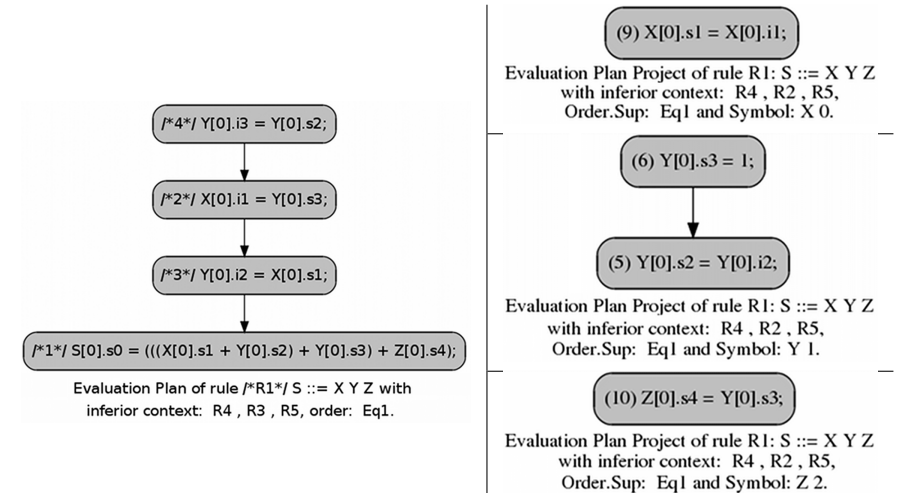
Plan de evaluación proyectado

Plan de evaluación considerando sólo los números de ecuaciones para los atributos de un símbolo particular.

“Los planes proyectados juegan un rol restrictivo sobre el orden de evaluación impuesto a contexto inferior.”



magGen: Ejemplo de plan de evaluación



magGen: Secuencia de visita

- Traducción de planes de evaluación a secuencia de visita: `compute`, `visit`, `leave`.
- Se basa en la aplicación de un recorrido sobre los planes de evaluación resolviendo las dependencias para evaluar los atributos.

Plan de evaluación \Rightarrow Acciones capaces de moverse sobre un AST; `{visit(node1), compute(E1), ..., compute(En)}`



Heurística del algoritmo

- 1 Se recorre el plan tomando en cada paso una de las ecuaciones.
- 2 Dada la ecuación *i*, en el plan, se debe computar todo el *L.value* de la ecuación. Para ello se deben resolver todas las dependencias dadas por *r.value*.
- 3 Dada una dependencia de la ecuación, primero se chequea si la misma fue computada en algún paso anterior, sino, se analiza:
 - **Símbolo de la parte izquierda de la regla** asociado a un **atributo heredado**, realizar un "leave".
 - **Símbolo de la parte derecha de la regla** y además contiene un **atributo sintetizado**, realizar un "visit". **Elección de plan**.
- 4 Luego de la obtención de todas las dependencias, se realiza un `compute` del *L.value* y se marca a este como evaluado.



Generación de código

La etapa final de **magGen** esta dada por la generación de código para el evaluador estático. Esta, produce dos archivos: *interface* (.hpp) e *implementación* (.cpp). Los archivos generados se vinculan con dos módulos estáticos, *Node.hpp* y *Node.cpp*.

Los módulos generados contienen:

- Los planes de evaluación.
- Las secuencias de visita.
- Las reglas de la gramática.
- Cada símbolo de la gramática es implementado mediante structs de C++.
- Conjunto de métodos para la evaluación.



```
(R1) S → XYZ
(1) S.s0 := X.s1 + Y.s2 + Y.s3 + Z.s4
(2) X.i1 := Y.s3
(3) Y.i2 := X.s1
(4) Y.i3 := Y.s2
...
```

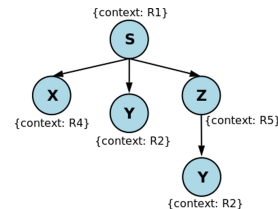
(R1) Rule: S := X Y Z;

{2, 3, 4, 1}

{2, -2, 1, -3, 2, -4, 3, -1}

Visit a nodo hijo Y en busca de Y(0).s3
Visit a nodo hijo X en busca de X(0).s1
Visit a nodo hijo Y en busca de Y(0).s2
Visit a nodo hijo Z en busca de Z(0).s4

Contexto: X->R4, Y->R2, Z->R5



Algoritmo de evaluación

Se basa en dos etapas bien definidas:

Primera recorrida del AST se invoca a la función

- `traverse`: responsable de seleccionar el plan de evaluación para cada nodo no terminal.

Segunda recorrida del AST se invoca a la función

- `eval_visitor`: es un evaluador orientado a visitas, que debe computar los valores de los atributos de los nodos.



Usando magGen

Luego de la instalación, **magGen** puede ser invocado desde la línea de comandos.

Argumentos de magGen

- `-f file` Entrada de **magGen** como `file`. Por defecto, `(cin)` hasta EOF (End Of File).
- `-i header` Incluir header en la generación de código.
- `-fo folder` Directorio de salida para **magGen**. Por defecto `"/out_maggen/"`.
- `-o name` Define a `name` como el nombre de la clase y del archivo generado por **magGen**. Por defecto el nombre `"mag_eval"`.
- `-h` Muestra mensaje de ayuda.



Usando magGen

- Se especifica en un archivo la MAG deseada.
- Se invoca a **magGen** con los parámetros adecuados desde la línea de comandos

Ejemplo: `./maggen -f ./mag.input -fo ./out_folder -o nameEval`

- Se construye el AST de entrada a evaluar en un archivo C++, junto con la función `main` invocando la evaluación
- Se ejecuta el binario para lograr los resultados.



Temario

- 1 ¿Qué es magGen?
 - Motivación
 - Objetivos
- 2 Preliminares
 - Gramáticas de Atributos Multiplanos (MAG)
 - Evaluador estático
 - Planes de evaluación
 - Secuencias de visita
- 3 magGen
 - Herramientas usadas
 - Funcionamiento
 - Lenguaje de Especificación
 - Análisis estático
 - Generación de código
 - Usando magGen
- 4 Evaluador generado
- 5 Performance
- 6 Comentarios Finales



Evaluador generado

Para el uso del evaluador generado por **magGen** se necesitan los 4 archivos generados en el directorio de salida:

- `maggen.hpp`.
- `maggen.cpp`.
- `Plan.hpp`.
- `Node.hpp`.

AST \Rightarrow **magGen** \Rightarrow AST decorado.



Salida completa de magGen

Al ejecutar **magGen** se generan, en la carpeta de salida, lo siguiente:

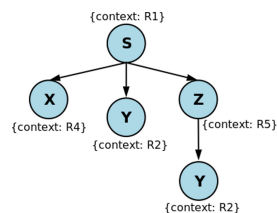
- Archivos **Node.hpp**, **Node.cpp**, **(eval).cpp**, **(eval).hpp**
- Archivo **grammar.log** con la gramática parseada.
- Carpeta **plan_project** con los planes proyectados.
- Carpeta **plans**, con los planes calculados.
- Carpeta **graph**, que contiene subcarpetas con los tipos de grafos construidos.

Tanto los planes como los grafos son generados en formato dot. Los mismo pueden ser convertido a imagen (png) utilizando el script que se encuentra en el paquete de la herramienta.



Construcción de AST de entrada

```
...
maggen eval_mag;
S node_s1(1);
X node_s1_x(4);
Y node_s1_y2(2);
Z node_s1_z(5);
Y node_s1_z_y3(2);
node_s1.add(&node_s1_x).add(&node_s1_y2).add(&(node_s1_z.add(&node_s1_z_y3)));
eval_mag.evaluador_mag(&node_s1);
cout << " After evaluation." << endl;
cout << node_s1.to_string() << endl;
...
```



Temario

- 1 ¿Qué es magGen?
 - Motivación
 - Objetivos
- 2 Preliminares
 - Gramáticas de Atributos Multiplanes (MAG)
 - Evaluador estático
 - Planes de evaluación
 - Secuencias de visita
- 3 magGen
 - Herramientas usadas
 - Funcionamiento
 - Lenguaje de Especificación
 - Análisis estático
 - Generación de código
 - Usando magGen
- 4 Evaluador generado
- 5 Performance
- 6 Comentarios Finales



Medidas de performance: Unicidad de planes

En un principio cada plan contenía el contexto y la lista de ecuaciones que lo definían. Esto llevo a una **saturación en la cantidad de planes**.

En la mayoría de los casos la explosión de planes se producía por la cantidad de contextos: **Planes repetidos**

- La unicidad de los planes permitió una **reducción notable** en la cantidad de planes.
- Optimización destacable en la generación de código del evaluador.



Medidas de performance: Unicidad de planes (cont.)

Ejemplo Aritmética de expresiones

Variable/Unicidad	Sin	Con
Cant. de planes	371	371
Cant. de planes proy.	687	687
Cant. de sec. de visita	371	22
Generación del evaluador	~1.44 sec	~0.47 sec
Cant. de líneas (eval.)	17464	5974



Medidas de performance: Versión de Spirit Boost

Computadora: Microprocesador Intel Core 2 Duo T5550 1.83 GHz, 2 MB memoria caché L2, 3 GB memoria RAM a 667 MHz.

Sistema operativo: GNU/Linux Ubuntu 10.04, Kernel 2.6.32-23, GNU g++ 4.4.3.

Tiempos de generación de evaluadores de magGen

Input / Spirit	1.8.X	2.3
MAG Wu Yang	~0.087 sec	~0.084 sec
MAG Aritmética	~0.590 sec	~0.450 sec



Medidas de performance: Compilación evaluador

Tiempos de compilación del evaluador generado

Input / g++	sin -O3	con -O3
Wuu Yang	~1.51 sec	~2.55 sec
Aritmética	~5.98 sec	~1 min 5.6 sec

Tamaños del ejecutable del evaluador generado

Input / g++	sin -O3	con -O3
Wuu Yang	104 Kb	48 Kb
Aritmética	516 Kb	384 Kb



Temario

- 1 ¿Qué es magGen?
 - Motivación
 - Objetivos
- 2 Preliminares
 - Gramáticas de Atributos Multiplanos (MAG)
 - Evaluador estático
 - Planes de evaluación
 - Secuencias de visita
- 3 magGen
 - Herramientas usadas
 - Funcionamiento
 - Lenguaje de Especificación
 - Análisis estático
 - Generación de código
 - Usando magGen
- 4 Evaluador generado
- 5 Performance
- 6 Comentarios Finales



Trabajos Futuros

- Implementar la generación de código al estilo “*plugins*”. Lo que permitiría extensiones varias, transparentes y elegantes para generar código en diferentes lenguajes. Esto implicaría la definición de una API del motor de generación de código.
- Definir una API para la construcción de los AST de entrada al evaluador generado, lo que permitiría que herramientas externas puedan generar los AST entrada.
- Pruebas de rendimiento como comparaciones con otras herramientas similares.
- Permitir definir atributos de **alto orden**, es decir, atributos que pueden ser un árbol. Para los cuales, su evaluación supondría un nuevo proceso de igual complejidad que el necesario para decorar al AST de entrada.
- Implementar como especies de templates para los atributos.



Extensiones

- Extender **magGen** para gramáticas dentro de la familia $NC(K)$.
- Extensiones al lenguaje de especificación, como módulos a través de `#include`. Esto engloba los puntos siguientes:
 - Disponer de espacios de nombres, y operadores que permitan referirse a estos **espacios de nombres**.
 - Redefinición o extensión de reglas incluidas, un estilo de **sobrecarga**.
 - Permitir el **adicionado de atributos** a símbolos pertenecientes a otros módulos.














Bibliografía I

- Wuu Yang. 1998. *Multi-Plan Attribute Grammars*. Department of Computer Information Science. National Chiao-Tung University, Hsin-Chu, Taiwan.
- 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.
- Wuu Yang. 1998. *Conditional Evaluation in Simple Multi-Visit Attribute Grammar Evaluators*. Department of Computer and Information Science. National Chiao-Tung University, Hsin-Chu, Taiwan.
- 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.
- John E. Hopcroft, Rajeev Montwani, Jeffrey D. Ullman. *Introduction to Automata theory, languages, and computation*. Addison-Wesley (2001) second edition.
- Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley (1985) Iberoamericana, S.A. Wilmington, Delaware, E.U.A.
- Arroyo, Marcelo Daniel. *Gramáticas de atributos, clasificación y aportes en técnicas de evaluación*. Tesis de carrera de Magíster en Ciencias de la Computación. Universidad Nacional del Sur. Bahía Blanca - Argentina.
- U. Kastens. 1980. *Ordered Attribute Grammars*. Acta Informática. Vol. 13, pp. 229-256.



Bibliografía II

-  Jazayeri, Ogden and Rounds. 1975. *The intrinsically exponential complexity of the circularity problem for attribute grammars*. Comm. ACM 18. December 2, Pag: 697-706.
-  Bertrand, Meyer (1997). *Object-Oriented Software Construction*. Segunda edición. Prentice Hall Professional Technical Reference. Santa Barbara (California).
-  Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley publishing Company, Reading, Massachusetts, E. U. A.(1983).
-  Valentin David. *Attribute Grammars for C++ Disambiguation*. LRDE, 2004.
-  D. Knuth. 1968. *Semantics of context free languages*. Math Systems Theory 2.June 2.Pag: 127-145.
-  **eclipse**. *Multi-language software development environment*. URL:<http://www.eclipse.org/>.
-  **doxygen**. *A documentation generator for C++, C, Java, Objective-C, Python, IDL (CORBA and Microsoft flavors), Fortran, VHDL, PHP and C#*. URL: www.doxygen.org.
-  **C++**. *C++ Annotations Version 8.2.0*. URL: <http://www.icce.rug.nl/documents/cplusplus/>.
-  **C++**. *C++ reference* URL: <http://www.cppreference.com/wiki/es/start>.
-  **TEX**. *A document preparation system* URL: <http://www.latex-project.org/>.
-  **Boost**. *Boost C++ Libraries*. URL: <http://www.boost.org/>.



Gracias



¿Preguntas?

