Generador de Evaluador estático

Tesis de la carrera Licenciatura en Ciencias de la Computación.

Autores:
KILMURRAY, Gerardo Luis
PICCO, Gonzalo Martin

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

Director:
Mg. Arroyo Marcelo.

Agradecimientos

A toda la familia.

Índice general

1 Introducción

Contenido

Desde que 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 como compiladores, intérpretes, traductores como también para especificar la semántica de lenguajes de programación. Las gramáticas de atributos son un formalismo simple para la especificación de la semántica de lenguajes formales, como los lenguajes de programación o de especificación. Integran la modularidad que brindan las gramáticas libres de contexto y la expresividad de un lenguaje funcional.

1.1. Gramática de Atributos

En una gramática de atributos, se relaciona con cada símbolo de una gramática libre de contexto un conjunto de atributos. Cada regla o producción tiene asociados un conjunto de reglas semánticas que toman la forma de asignación a atributos de valores denotados por la aplicación de una función, la cual puede tomar como argumentos intancias de atributos pertenecientes a los símbolos que aparecen en la producción. Las reglas semánticas inducen dependencias entre los atributos que ocurren en la producción. El orden de evaluación es implícito (si existe) y queda determinado por las dependencias entre las instancias de los atributos. Una regla semántica se podrá evaluar cuando las instancias de los atributos que aparecen como sus argumentos estén evaluadas. Un evaluador de gramáticas de atributos debe tener en cuenta las dependencias entre las instancias de atributos para seguir un

orden consistente de evaluación de los mismos. Si una GA contiene dependencias circulares no podría ser evaluada ya que no podría encontrarse un orden de evaluación. Existen numerosas herramientas basadas en este formalismo o en alguna de sus ex-tensiones, entre las cuales podemos mencionar yacc, Yet Another Compiler-Compiler,desarrollado por AT&T, AntLR, JavaCC, JavaCUP, ELI y muchas otras.

1.2. Arbol sintactico atribuido

1.3. Métodos de Evaluación

Los métodos estáticos deben tener en cuenta todos los posibles árboles sintácticos posibles a ser generados por la gramática y calcular todas las posibles dependencias entre las instancias de los atributos. Además, se deberán detectar las posibles depen-dencias circulares, para informar la viabilidad de su evaluación. Esto se conoce como el problema de la circularidad, el cual se ha demostrado ser intrínsecamente exponencial [20]. El problema de la circularidad ha motivado que muchos investigadores hayan rea-lizado esfuerzos en la búsqueda e identificación de familias o subgrupos de gramáticas de atributos, para las cuales puedan detectarse circularidades con algoritmos de menor complejidad (polinomial o lineal). Estas familias imponen restricciones sobre la gramática de atributos o sobre las dependencias entre sus atributos para garantizar que una GA no sea circular, con el costo de restringir su poder expresivo. Las clases de familias de gramáticas de atributos que se han utilizado para el desarrollo de herramientas eficientes y que se encuentran ampliamente analizadas en la bibliografía especializada, encontramos las s-atribuidas2 , l-atribuidas, las gramáticas de atributos ordenadas (OAG) y las absolutamente no circulares (ANCAG)[2]. En 1980, Uwe Kastens[23] caracterizó las gramáticas de atributos ordenadas y propuso un método para su evaluación, denominado secuencias de visita. Estas son secuencias de operaciones que conducen el recorrido del árbol sintáctico atribuido y realizan la evaluación de las instancias de los atributos. Kastens propone un método para generar las secuencias de visita en tiempo poli-nomial para la familia OAG. Mas recientemente, en 1999, se han propuesto nuevas familias de GA para las que se pueden implementar evaluadores eficientes basado en métodos estáticos y con un mayor poder expresivo que las utilizadas tradicionalmente[44].

1.3.1. Evalución dinámica

bla bla

1.3.2. Evaluación estática

1.3.3. Evaluacion de la familia ANCAG

1.4. Secuencia de visita

1.5. Generación de evaluadores para GA bien definidas

1.6. Evaluación durante el parsing

bla bla

2
Clasificación de AG

Contenido

bla

2.1. Clasificación basada en la estrategia de evalución

bla bla

2.2. Clasificación basada en dependecias

bla bla

2.3. Clasificación de Knuth

cla cla

2.3.1. Árbol sintáctico atribuído

3
Multi-plans attribute grammar

Contenido

3.1. Clasificación de Wuu Yang

3.1.1. Gramaticas no circulates(NC)

bla bla

3.1.2. ANCAG

4
Evaluacion estatica de MAG

Contenido

bla bla

4.1. Algoritmos para grafos: DP, DOWN, DCG y ADP

bla bla

4.2. Algoritmo de computo de planes

bla bla

4.3. Algoritmo de computo de secuencia de visita

5
Metodología de trabajo

Contenido

5.1. Practicas de software

El análisis y especificación de requerimientos puede parecer una tarea relativamente sencilla, pero la realidad es que el proceso de escribir un software requiere de un marco de trabajo para estructurar, planificar y controlar el desarrollo del sistema. Al mismo tiempo, el uso de herramientas en cada etapa del ciclo de vida (análisis, diseño, implementación y prueba), permite recorrer un camino de creación incremental del sistema, donde cada estadio del proceso refina el modelo. La importancia del uso de herramientas, modelos y métodos para asistir el proceso radica en visualizar y garantizar cualidades del producto desarrollado en practicas de software comprobadas teóricamente.

Algunas de las prácticas de software se tratan en la siguiente sección:

Análisis-Diseño Esta etapa se baso en el estudio del marco teórico compuesto por papers y libros propuestos por el director de tesis. Edemas, se concretaron reuniones frecuentes para evacuar dudas y tomar decisiones respecto a objetivos y aspectos a considerar en el modelo. Esta fase, también, se utilizo para familiarizarse y solidificar el manejo de herramientas empleadas en los distintos estadios del proceso.

Implementación En la etapa de implementación se invirtió una gran porción del tiempo total del proyecto. Esta fase, se dividió principalmente, en abordar los distintos estadio considerados en el análisis-diseño, pero, también, el refinamiento del diseño era una tarea que jugaba un papel importante.

Prueba Esta etapa esta íntimamente relacionada con la etapa anterior (implementación), debido a que fueron realizadas en conjunto. Es decir, las pruebas eran abordadas luego de la implementación de cada fase distinguida en análisis-diseño. Para ello se planteaba casos de prueba específicos para cada fase, basándose en casos abordados en el marco teórico soporte del proyecto.

Documentación La documentación ,a nivel de código, fue abordada desde la etapa de implementación hasta el fin del proceso. Esta etapa prioriza en hecho de clarificar detalles de implementación hacia la comunicación entre los desarrolladores, como así también para desarrolladores o posibles colaboradores externos al proyectos. Edemas, en la parte final se utilizo full-time a la elaboración del informe, presentación y demas, que hacen el desarrollo de una tesina de materia de grado.

5.2. Lenguaje de programación C++

C++ es un lenguaje de programación con tipado estático, multi-paradigma, compilado y de propósito general. Fue desarrollado por Bjarne Stroustrup en el año 1979 en los laboratorios Bell, como una mejora al lenguaje de programación C, y fue originalmente llamado "C con clases".

El lenguaje ha evolucionado, ha sido estandarizado y aún continúa evolucionando. Actualmente, C++ soporta varios conceptos que permiten escribir programas con diferentes estilos: imperativo (procedural), orientado a objetos (herencia, polimorfismo, programación genérica, metaprogramación, etc).

La elección de C++ como lenguaje a utilizar en el desarrollo de **magGen** surgió despues de reuniones con el director de tesis, en las cuales de evaluaron lenguajes y se analizaron parámetros como lo son:

eficiencia en cuanto a tiempo de ejecución, posibilidad de redefinir de los operadores en un contexto dado, mediante la *sobre carga de operadores*, utilización de librerías maduras como soporte de componentes necesarios y extras al objetivo de la tesis, etc.

Esta ultima, permitió la disponibilidad de bibliotecas genéricas, como la STL (Standard Template Library) y Boost Library que fueron utilizadas para disponer de funcionalidades y estructuras con solidas referencias.

5.3. Herramientas

Las lista de herramientas que se detallan a continuación fueron utilizadas con resultados muy positivos en cada una de las etapas del desarrollo de sistema. Es de destacar que las herramientas son "free software".

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE). La versión utilizada fue "Galileo" (lanzada el 24 de junio del 2009). [1]

[1] `http://www.eclipse.org/`

Subversion es un software de sistema de control de versiones. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como *svn* por ser ese el nombre del comando que se utiliza. Esta herramienta fue de vital importancia para llevar a cabo la coordinación, comunicación y elaboración controlada entre los desarrolladores autores del trabajo.[2]

LaTeX 2$_\varepsilon$ es una herramienta para sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas. Este documento en su totalidad se escribió utilizando LaTeX 2$_\varepsilon$.[3]

kile es un editor de Tex/LaTeX. Funciona conjuntamente con KDE en varios sistemas operativos.[4]

Graphviz es una herramienta de visualización de grafos de código abierto. Genera una gran variedad de formatos de salida [5]

Nemiver es una herramienta de debugger que se integra perfectamente en el entorno de escritorio GNOME. En la actualidad cuenta con un motor que utiliza el conocido GNU gdb debugger para depurar programas C/C++. [6]

Dia es una herramienta para la creación de cualquier tipo de diagrama.[7]

Bouml es una herramienta para la creación de diagramas UML.[8]

Análisis estático de código entre los que se encuentran:

Cloc **C**ounter **l**ines **o**f **c**ode. contador de lineas en blanco, lineas de comentario y lineas de código reales en muchos lenguajes de programación. Cloc esta escrito en "Perl" sin dependencias externas fuera del estándar de la distribución "Perl v5.6".[9]

CCCC **C** and **C++** **C**ode **C**ounter. Fue desarrollado como un campo de pruebas para una serie de ideas relacionadas con las métricas de software en un proyecto de Maestría [10]

Gcov es un test de cubrimiento o cobertura de código. Es una forma de probar partes del programa no incluidas en los casos de prueba. Se utiliza en conjunto con GCC. [11]

[2] http://subversion.apache.org/
[3] http://www.latex-project.org/
[4] http://kile.sourceforge.net/
[5] http://www.graphviz.org/
[6] http://projects.gnome.org/nemiver/
[7] http://live.gnome.org/Dia
[8] http://bouml.free.fr/
[9] http://cloc.sourceforge.net/
[10] http://cccc.sourceforge.net/
[11] http://gcc.gnu.org/onlinedocs/gcc/Gcov.html

6
Acerca de magGen

Contenido

6.1. Qué es magGen

6.2. Lenguaje de especificación de las MAG

El lenguaje de especificación utilizado para la descripción de una Gramática de atributos (MAG) fue definido en el marco de este proyecto. Esto permite definir una gramática de atributos como input de **magGen** . La secciones que conforman la descripción de una gramática de atributos se corresponden con las características que definen a una gramática de atributos como tal (ver capitulo de GA). Informalmente, los bloques que conforman la especificación son:

Bloque Dominio Semántico Destinando a la declaración de sort, operadores y funciones que se utilizaran en el bloque de ecuaciones. Este bloque es denominado “semantic domain”.

Bloque de Atributos Destinando a la declaración y definición de los atributos asociados a cada símbolo. Este bloque es denominado “attributes”.

Bloque de Reglas Destinado a la declaración y definición de las reglas sintácticas de la gramática con sus correspondientes ecuaciones semánticas para cada atributo asociado a cada símbolo. Este bloque es denominado “rules”.

De manera formal, se define una gramática G: CFG que define el lenguaje de especificación para el archivo de entrada aceptado por **magGen** .

```
S   =   semantic domain decl_Sd
    |   attributes decl_attr
    |   rules  decl_rules
```

En las secciones posteriores se presentaran los símbolos con mas detalle $< decl\_Sd >$, $< decl\_attr >$ y $< decl\_rules >$

6.2.1. Bloque Dominio semántico

En esta sección presentaremos el bloque "semantic domain" en detalle. Este bloque esta subdividido en 3 secciones, que se corresponden con la definición de los sort o tipos, los operadores y las funciones.

```
decl_Sd = decl_sorts
        | decl_operators
        | decl_functions
```

6.2.1.1. Declaración de sort

La declaración de "sort" esta dada por la siguiente sintaxis:

```
decl_sorts  =   'sort' NAME_SORT ';'
```

6.2.1.2. Declaración de operadores

En esta sección presentaremos la seccion de declaración de operaciones en detalle, la cual esta definida de la siguiente manera:

```
decl_operators =
              'op' infix mode_op NAME_OP ':' NAME_SORT ',' NAME_SORT '->' NAME_SORT ';'
            | 'op' prefix mode_op NAME_OP ':' NAME_SORT '->' NAME_SORT ';'
            | 'op' posfix mode_op NAME_OP ':' NAME_SORT '->' NAME_SORT ';'

mode_op         = '(' NUM_PRECEDENCE ','assoc ')'

assoc           = (left | right | non_assoc)
```

6.2.1.3. Declaración de funciones

```
decl_functions =  'function'    NAME_FUNC':' domain '->' NAME_SORT ';'
domain  = NAME_SORT
        | NAME_SORT ',' domain
```

6.2.2. Bloque de Atributos

En esta sección presentaremos el bloque "attributes" en detalle. El bloque de atributos esta definida de la siguiente manera.

```
decl_attr   = (d_attr)+
d_attr      = NAMA_ATTR : '<' t_attr '>' type_attr 'of' '{'list_symbol'}';
t_attr      = inh
            | syn
list_symbol = SYMB_NON_TERMINAL
            | SYMB_NON_TERMINAL ',' list_symbol
```

6.2.3. Bloque de reglas

Por ultimo el bloque de reglas. Esta dado por el siguiente sintaxis:

```
decl_rules   = (d_rule)+
d_rule       = SYMB_NON_TERMINAL '::=' rigft_symb decl_eqs
rigft_symb   = ( SYMB_NON_TERMINAL | SYMB_TERMINAL)+

decl_eqs     = 'compute' d_eqs  'end;'
             | ';'
d_eqs        = instance '=' right_eq ';'

right_eq     = leaf
             | leaf OP_INFIX leaf right_eq
             | (OP_PREFIX)+ leaf
             | leaf (OP_POSTFIX)+
             | 'NAME_FUNC' '(' right_eq ')'

leaf         = instance
             | LITERAL

instance     = 'symb_non_terminal' '[' 'num_ins' ']' '.' 'name_attr'
```

6.2.4. Comentarios

La especificación permite agregar comentarios. Por cuestiones de simplicidad se han utilizado las mismas reglas sintácticas que C y C++ para el adicionado de lineas o bloques de comentario. Los cuales se detallan a continuación:

/* **comment** */ es la forma de inserción de bloques de comentarios.

// **line commet** es comentario de una linea.

6.2.5. Ejemplo

El ejemplo presentado en la figura 6.1 es uno de los casos de test desarrollado para la construcción de **magGen** . La importancia de este radica en que, el mismo, es un caso de estudio dado en una de las principales bases teóricas que han sido usadas para el sistema.

6.3. Estrucutas internas

bla bla

```
/**
  *  \file              Mag.txt
  *  \brief             Attribute Grammar example.
  *  \date              15/02/2010
  *  \author            Kilmurray, Gerardo Luis <gerakilmurray@gmail.com>
  *  \author            Picco, Gonzalo Martin <gonzalopicco@gmail.com>
  */
/*****************************
 * Block of Semantic Domain *
 *****************************/
semantic domain
    /*********************
     * List of Operators *
     *********************/
    op infix    (10, left) +: int, int -> int;
/************************
 * Block of Attributes *
 ************************/
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};
/******************
 * Block of Rules *
 ******************/
rules
            // P1
    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;
    // P2
    Y ::= 'm'
                compute
                        Y[0].s2 = Y[0].i2;
                        Y[0].s3 = 1;
                end;
            // P3
    Y ::= 'n'
                compute
                        Y[0].s2 = 2;
                        Y[0].s3 = Y[0].i3;
                end;
            // P4
            X ::= 'm'
                compute
                        X[0].s1 = X[0].i1;
                end;

            // P5
            Z ::= Y
                compute
                        Z[0].s4 = Y[0].s3;
                        Y[0].i2 = 3;
                        Y[0].i3 = Y[0].s2;
                end;
```

Figura 6.1: Ejemplo ag_wuu_yang.input

6.4. Disenõ del evaluador estático generado

bla bla

7
Diseño e Implementación de magGen

Contenido

bla bla

7.1. GNU/linux

bla bla

7.2. Algoritmo de generación de secuencia de visita

bla bla

7.3. Algoritmo de generación de código

bla bla

8
Usos

Contenido

8.1. Uso de MagGen

bla bla

8.2. Uso del evaluador generado

bla bla

9
Conclusión

Contenido

bla bla.

9.1. Conclusión

bla bla

9.2. Trabajos futuros

bla bla

A
Appendix Example

A.1. Appendix Example section

And I cite myself to show by bibtex style file (two authors) [**?**].
This for other bibtex stye file : only one author [**?**] and many authors [**?**].

Design and Use of Numerical Anatomical Atlases for Radiotherapy

Resumen: El tratamientos de lenguajes es uno de los temas mas estudiados en los ultimos años. Las gramaticas de atributos son un formalismo que permite utilizar el poder descriptivo de las gramaticas libres de contexto y la expresividad de los lenguajes funcionales.

Keywords: Atlas-based Segmentation, non rigid registration, radiotherapy, atlas creation