Grupo de Procesadores de Lenguajes

Línea:Herramientas de generación de procesadores de lenguajes basados en Gramáticas de Atributos

Marcelo Arroyo, Jorge Aguirre, Nicolás Florio, Román Alarcón [*]
Dpto de Computación - FCEFQyN
Universidad Nacional de Río Cuarto

Resumen

En el presente trabajo se describen las dos útimas herramientas de generación de procesadores de lenguajes basados en Gramáticas de Atributos desarrolladas por el grupo. Se presentan dos herramientas: **nceval** y **agcc**. La primera se basa en la familia NC(1) y genera evaluadores concurrentes. La última es un generador integrado de analizadores léxicos, sintácticos con soporte para múltiples evaluadores.

1. Introducción

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 ser lenguajes de programación o lenguajes de especificación. Integran la modularidad que brindan las gramáticas libres de contexto y la expresividad de un lenguaje funcional. Con cada símbolo de una gramática libre de contexto se asocia un conjunto de *atributos*. Las reglas o producciones tiene asociados un conjunto de *reglas semánticas* que toman la forma de una asignación a una instancia de un atributo el resultado de la aplicación de una función. Dichas funciones pueden toman como argumentos instancias de atributos que *ocurren* en la producción (o valores constantes).

Estas reglas introducen *dependencias* entre los atributos que ocurren en la producción. El orden de evaluación es implícito y un evaluador debe tener en cuenta las dependencias entre los atributos para definir un orden correcto de evaluación de los mismos.

Existen numerosas herramientas basadas en este formalismo y/o sus extensiones, entre las cuales podemos mencionar *yacc, Yet Another Compiler-Compiler*, desarrollado por AT&T, *AntLR*, *JavaCC*, *JavaCUP* y *ELI*.

[*] {marroyo,jaguirre,florio,ralarcon}@dc.exa.unrc.edu.ar

Uno de los principales problemas a resolver estáticamente, es la detección de dependencias circulares, conocido como el *problema de la circularidad* entre los atributos de una GA, ya que es un problema intrínsecamente exponencial[8]. Si una GA contiene dependencias circulares no podría ser evaluada[1] por un evaluador tradicional.

Se pueden utilizar dos enfoques para la evaluación: dinámica y estática. Los evaluadores dinámicos pueden realizar la evaluación bajo demanda o construyen un grafo de dependencias y luego evalúan las instancias en un orden determinado por la topología del grafo (topological-sort). Si el grafo es circular, producen un error. Los generadores estáticos generan evaluadores que siguen un orden de evaluación generado analizando las dependencias de los atributos. Como el análisis estático puede tener costo exponencial, generalmente restringen la familia de AG, imponiendo restricciones en las dependencias, que permiten computar los planes en tiempo polinomial, como las OAG (ordered AG).

2. Productos obtenidos

El grupo ha desarrollado dos herramientas de procesamiento de lenguajes basados en GAs. El primero de ellos, denominado *japlage* y descripto en [1], se basa en un generador basado en GAs y esquemas de traducción, el cual genera evaluadores concurrentes dinámicos que usan una estrategia bajo demanda.

A continuación se describe otra herramienta desarrolladas por el grupo para una familia específica de GAs.

```
Attributes
...
a,b:T1 syntetized of X,Y;
h:T2 inherited of Y;
...
Rules
...
X -> X Y    attribution
              X.a = f(Y.b)
              ...
              Y.h = g(X[1].a)
              ...
            end
...
```

Figura 1: formato de una especificación

[1]Podría ser evaluada si existe un punto fijo sobre la relación de dependencia utilizando evaluación *normal*.

2.1. El generador de evaluadores *nceval*

Esta herramienta[2] genera estáticamente evaluadores concurrentes para gramáticas de atributos $NC(1)$. El usuario especifica una GA como la del ejemplo de la figura 1 y genera un evaluador en lenguaje *java* (si es que pertenece a la familia).

La familia de AGs $NC(1)$ incluye a las *Absolutamente No Circulares* o *ANCAG*. Una GA de esta familia puede evaluarse con un evaluador basado en secuencias de visita. El evaluador asocia un conjunto de planes (secuencia de instancias de atributos) a cada producción[3]. Antes de iniciar el proceso de evaluación de las instancias de los atributos, el evaluador deberá seleccionar uno de los planes posibles en cada nodo del árbol sintáctico.

Los evaluadores generados por *nceval* son evaluadores concurrentes. Las instancias de los atributos en un árbol sintáctico se particionan en regiones disjuntas e independientes. Toda la información sobre las particiones se genera estáticamente (del mismo modo que los planes de evaluación) y se basa en un particionado basado en las dependencias como el propuesto por Wuu Yang en [15].

Los procesos evaluadores no necesitan sincronización ni comunicación entre sí, ya que actúan sobre conjuntos de instancias de atributos independientes. Hasta el momento no se conocen otras herramientas con estas características.

3. Herramienta en desarrollo: *agcc*

Los principales objetivos en el desarrollo de *agcc* son obtener un generador de procesamiento de lenguajes basado en GA que generara un sistema integrado y modular con todos los componentes necesarios: analizador léxico, analizador sintáctico y evaluador de atributos. Para las dos primeros componentes, la idea es reutilizar otras herramientas existentes.

El usuario escribe la especificación de una GA (similar a la de la figura 1) y *agcc* genera un parser que construye el árbol sintáctico que luego es tomado por el evaluador de atributos.

Es posible generar dos tipos de evaluadores: un evaluador dinámico bajo demanda (con detección de circularidad) y otro evaluador basado en secuencias de visita (con soporte para AGs multiplan).

El evaluador dinámico (bajo demanda) implementa un autómata pila que comienza intentando evaluar los atributos del nodo raíz y se van apilando las instancias de los atributos necesarios a partir de ellos. Cada instancia de un atributo en un nodo del árbol sintáctico tiene una *marca* (un bit) sobre su estado (evaluado o no-evaluado). Se utiliza un bit adicional por cada instancia de un atributo para la detección de posibles circularidades.

En el caso que se desee utilizar un evaluador basado en secuencias de visita, cada nodo del árbol sintáctico construido por un parser, tiene asociado un plan de evaluación

[2]Para más información sobre el diseño de *nceval*, ver [4]

[3]Denominados planes *posibles* la producción.

(computado estáticamente por el generador de planes) de los atributos correspondiente a la instancia de la producción aplicada en el nodo. Cada plan de evaluación está representado como una secuencia de operaciones de secuencias de visita[9]. Una secuencia de visitas contiene tres tipos de operaciones: *visit(i)*, *compute(j)* y *leave*. Una operación *visit(i)* (ejecutándose en el contexto de un nodo n), implica que el evaluador debe moverse al conexto inferior i (i-esimo nodo hijo). Una operación *compute(j)* establece que debe computarse la j-ésima función semántica. Una operación *leave* representa un movimiento al nodo padre (o finalizar si se ejecuta en el nodo raíz).

El evaluador selecciona los planes requeridos para cada nodo en la primer visita al nodo (en el caso que la AG sea multiplan).

La figura 2 muestra los componentes de un procesador de lenguajes generado por *agcc*.

Los analizadores sintácticos y léxicos generados se obtienen mediante herramientas existentes (*lex* y *yacc* en la versión corriente) y el evaluador de atributos está formado por un conjunto de clases C++.

agcc es modular en el sentido que su diseño permite incorporar múltiples generadores de planes de evaluación para soportar otras familias de GAs, pudiéndose utilizar el mismo evaluador. Los generadores de código también se pueden extender para generar código en otros lenguajes.

4. Conclusiones y trabajo futuro

En este trabajo se han presentado dos herramientas que permiten generar procesadores de lenguajes basados en GA. Actualmente se están realizando pruebas de ambos productos descriptos y se están estudiando algunas extensiones al lenguaje de especificación como soporte para GA condicionales al estilo Boyland[5] y para GA de alto orden.

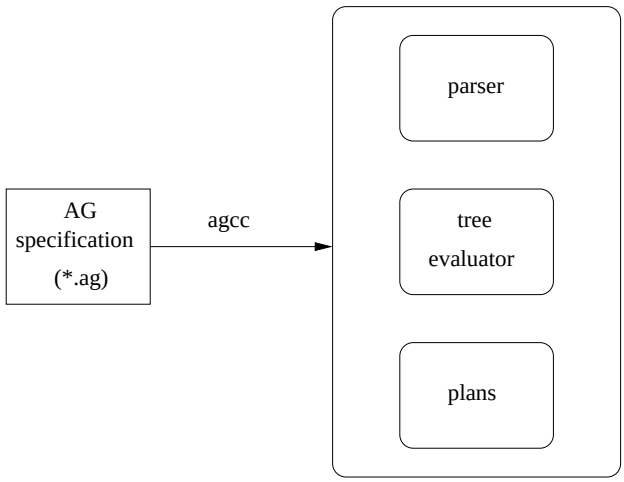

Figura 2: esquema de un procesador generado por *agcc*

En el caso particular de *agcc*, esta herramienta ha sido implementada en C++ y genera especificaciones *lex* y *yacc* y el código generado es C++, pero el diseño contempla generar código para otras especificaciones basadas en otras herramientas y otros lenguajes de programación, como podría ser JavaCup para lenguaje Java.

Referencias

[1] Aguirre, Arroyo, Grinspan. 1999. *Un ambiente de ejecución de procesadores de lenguajes orientado a objetos basado en gramáticas de atributos.*

[2] Aho, Ullman. *Compilers, Concepts, Principles and Tools.*

[3] Arroyo M., Aguirre J., Florio N. 2003. *Generación de evaluadores estáticos de gramáticas de atributos bien definidas.* CACIC 2003.

[4] Arroyo M., Aguirre J. Florio N. 2002. *Generación estática de evaluadores concurrentes para gramáticas de atributos NC(1).* CACIC 2002.

[5] Boyland J.T. 1996. *Conditional Attribute Grammars.* ACM Transactions on Programming Languages and Systems 18. pag. 73-108.

[6] Engelfriet J., Filé. 1982. *Simple multi-visit attribute grammars.* J. Comput. Syst. Sci. 24, 3, 283-314.

[7] Grosch J. 1992. *Efficient Evaluation of Well Formed Attribute Grammars and Beyond.* GMD Forschungsstelle an der Universität Karlsruhe. Germany.

[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.

[9] Uwe Kastens. 1980. *Ordered Attribute Grammars.* Acta Informática 13,3. March. Pag: 229-256.

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

[11] Mason T., Levine J., Brown D. *Lex & yacc.* O'Reilly. ISBN: 1-56592-000-7.

[12] 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, R.O.C.

[13] Wuu-Yang. 1998. *Multi-Plan Attribute Grammars.* Department of Computer and Information Science. National Chiao-Tung University, Hsin-Chu, Taiwan, R.O.C.

[14] 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, R.O.C.

[15] Wuu Yang. 1999. *A finest partitioning algorithm for attribute grammars.* 1999. Second Workshop on Attribute Grammars and their Applications - WAGA99. pp. 77-92.