Grupo de Procesadores de Lenguajes - Linea: Entorno de Generación de Procesadores de Lenguajes

Johanna Mussolini, Cecilia Kilmurray, Jorge Aguirre, Marcelo Arroyo *
UNRC 54+358-676235/(530 FAX){johanna, ckilmurray, jaguirre, marroyo}@dc.exa.unrc.edu.ar

2004

Resumen

En el presente trabajo se expone una de las líneas del grupo de investigación: "Procesadores de Lenguajes"perteneciente al Departamento de Computación de la UNRC, que se ocupa de los modelos teóricos, técnicas y herramientas para la generación de Procesadores de Lenguajes. Los trabajos previos del grupo en esta línea han permitido definir un modelo teórico de definiciones guiadas por sintaxis que brinda una integración más general de las Gramáticas de Atributos y Esquemas de Traducción que los generadores existentes. Sobre dicho modelo se construyó un entorno de generación de Procesadores de Lenguajes llamado Japlage y se implementó su primera versión. También se definió un nuevo formalismo, las Expresiones Regulares Traductoras (ERT). Ellas permiten definir patrones regulares conjuntamente con su traducción. Basado en las ERT también se construyó un generador de analizadores lexicográficos, JTLex. Aquí se describe la nueva versión (V 1.1) de Japlage, que incorpora a JTLex. Esta versión permite utilizar especificaciones lexicográficas basadas en ERT. Se incluye un ejemplo que muestra las ventajas que presenta el uso de este tipo de especificaciones. Finalmente, se analizan los trabajos futuros, en los cuales se intentará obtener una tercera versión de Japlage optimizada en tiempo de ejecución.

1. Introducción

Un Esquema de Traducción con Atributos (ETA) es una Gramática de Atributos (GA) extendida con los elementos de un Esquema de Traducción (ET). Además de un alfabeto de entrada $\sum$ existe un alfabeto de salida Γ. Cada producción de la gramática libre de contexto subyacente en la GA, tiene asociado, además de un conjunto de reglas de evaluación de atributos, una traducción sobre Γ. Los ETs también tienen un alfabeto de salida Γ y están integrados por una gramática libre de contexto G cuyas reglas tienen traducciones asociadas. En ambos casos -ETA y ET- Γ suele considerarse integrado por acciones ó procedimientos de un lenguaje imperativo. Las acciones que componen la salida se ejecutan secuencialmente. En los ETs los procedimientos se comunican mediante atributos asociados a los símbolos, como en una GA pero su valores son computados por acciones de la traducción, por lo cual la evaluación sólo se realiza en el orden prestablecido de ejecución;

*Este trabajo ha sido realizado en el marco de proyectos subsidiados por la Agencia Córdoba Ciencia y la SECyT de la Universidad Nacional de Río Cuarto.

mientras que un ETA permite definir cualquier evaluación bien definida, sin que se especifique el orden de evaluación de los atributos involucrados. Esta ampliación del campo de definición de los ETAs tiene importantes implicaciones prácticas, por ejemplo el sistema de tipos de las expresiones ADA puede definirse directamente con un ETA y no con un ET.
Japlage, es un entorno de generación de procesadores de lenguajes. Él permite generar un procesador de lenguajes, traductor o compilador, a partir de su especificación por medio de un ETA y de la especificación del analizador léxico, que construya sus tokens a partir de la cadena de entrada. El generador de procesadores de lenguajes es, en esencia, un compilador que a partir de las especificaciones de entrada obtiene un procesador del lenguaje especificado. Este procesador de lenguaje podrá ser, por ejemplo, un compilador, que produzca programas ejecutables a partir de programas fuentes o un intérprete, una interface de usuario que permite interactuar con un sistema de información mediante un lenguaje ad hoc, etc. La salida del generador provee el sistema de tiempo de ejecución que ejecuta al ETA especificado en la entrada, denominado Run Time System (RTS).
Una Expresión Regular Traductora (ERT) es una expresión regular en la cual los términos están constituidos por pares carácter-acción [Agu99]. Dentro de las ERT podemos caracterizar las Expresiones Regulares con Traducción Unica (ERTU), aquellas a las que a cada cadena de entrada corresponde una única cadena de salida. Por último, se encuentran las ERTL (Expresiones Regulares Traductoras Lineales) que constituyen una subclase de las ERTU. Las ERTLs son definidas por razones de eficiencia. Las ERTLs se caracterizan porque en ellas deben ser únicas las acciones asociadas a ocurrencias de símbolos que finalicen la generación de un mismo prefijo. Las ERTLs se corresponden con las ERTUs que pueden ser implementadas por Autómatas Finitos Traductores o máquinas de Moore, cuyo tiempo de ejecución es lineal respecto de la cadena de entrada. Por esta razón han elegido a las ERTLs como el formalismo sobre el cual se basa JTLex [Bav02], un generador de Analizadores previamente desarrollado dentro del grupo de trabajo. Facilidad que ha quedado incorporada a japlage, como aquí se describe.

2. Resultados Previos

2.1. JTLex un generador de Analizadores Léxicos Traductores

Fue diseñado e implementado JTLex [Bav02],un generador de analizadores léxicos, que al contrario de los generadores existentes, permite la especificación conjunta de la sintaxis y la semántica de los componentes léxicos, siguiendo el estilo de los esquemas de traducción. Para ello se basa en un nuevo formalismo, las Expresiones Regulares Traductoras (ERTs), [Agu99].Esta herramienta genera automáticamente un analizador léxico a partir de la especificación provista por el usuario. Los analizadores-traductores generados por JTLex procesan una cadena α en tiempo $O(|\alpha|)$ y el tiempo de generación de esta herramienta es del mismo orden que el requerido por los algoritmos utilizados tradicionalmente. Tanto su diseño como la especificación de sus procedimientos con que el usuario implementa la semántica asociada a los símbolos son Orientados a Objetos. El lenguaje de implementación de JTLex es JAVA, como así también, el del código que genera y el que usa el usuario para definir la semántica. El lenguaje de especificación brindado por JTLex sigue el estilo de Lex - que es prácticamente un estándar -.

2.2. Japlage un entorno de Generación de Procesadores de Lenguajes, versión 1.0

Fue diseñado e implementado Japlage,[Fel02] un entorno de generación de procesadores de lenguajes o de compilación de compiladores, basado en un formalismo que brinda la posibilidad de utilizar todas las variantes existentes de la evaluación de Gramática de Atributos (GA), conjuntamente con la potencia de los esquemas de traducción, este nuevo formalismo ha sido llamado Esquema de Traducción con Atributos (ETA) .
En un ETA se declaran separadamente las acciones de traducción y las reglas de evaluación de atributos. Las acciones deben ser ejecutadas en el mismo orden relativo que en los esquemas de traducción. En tanto que las reglas de evaluación de atributos, como en las GAs, pueden ejecutarse en cualquier orden relativo, que respete la dependencia entre los atributos. Finalmente para que una acción sea ejecutada, es necesario que se hayan computado todos los atributos de los que hace uso y todas las acciones que la preceden visitando el árbol primero en profundidad .
Japlage implementa a un ETA mediante un conjunto de procesos concurrentes que se sincronizan por la disponibilidad de atributos evaluados.
Japlage fue diseñado utilizando la tecnología de Orientación a Objetos, como así también el lenguaje de especificación que debe utilizar el usuario, este sigue el estilo yacc, que prácticamente es un estándar. Dicho diseño fue realizado siguiendo el proceso unificado de desarrollo de software - UML - [Han98], siguiendo diversos patrones de diseño [Gam95]. Japlage soporta la facilidad de las Gramáticas de atributos condicionales [Boy96]. Japlage está implementado sobre JAVA.

3. Resultados del último año

3.1. Japlage un entorno de Generación de Procesadores de Lenguajes, versión 1.1

Durante el ultimo año fue desarrollada una nueva versión de Japlage que incorpora a la versión anterior un módulo alternativo -seleccionable por el usuario- de generación de analizadores lexicográficos basado en Expresiones Regulares Traductoras Lineales (JTLex). Para llevar a cabo la incorporación JTLex se desarrolló un módulo que actúa en tiempo de generación del compilador y es el encargado de seleccionar entre un generador de analizadores clásicos, provisto por Japlage, y el generador de analizadores léxicos traductores JTLex; también fue desarrollada una interface para unificar las incompatibilidades presentes entre las clases del procesador de lenguajes generado por Japlage y las clases generadas por JTLex. Dicha interface actúa en tiempo de ejecución del procesador generado; para el diseño de la misma se siguió el *pattern adapter* [Gam95],el cual permite que ciertas clases, que resultan incompatibles, puedan trabajar conjuntamente mediante el desarrollo de un módulo que actúa como intermediario.
El desarrollo de esta nueva versión está basado en la tecnología Orientada a Objetos y se ha utilizado el mismo lenguaje de implementación que el de la versión 1.0 de Japlage (JAVA).

Esta nueva versión de Japlage, al incorporar el generador de analizadores lexicográficos JTLex, permite que pueda ser utilizada la potencia expresiva de las ERT para definir la especificación de un analizador léxico. Por ejemplo, si observamos la Figura 1 en donde

```
%{
int intval;float realval1;float realval2;float real;int cantreal;
%}
%init{
%init}
%%
(\ |\t)+ {break;}
\n       {break;}
"+"     {return TokenDef.MAS;}
"*"{return TokenDef.POR;}
"−"{return TokenDef.MENOS;}
"/"{return TokenDef.DIVIDE;}
"("{return TokenDef.PAREN_A;}
")"{return TokenDef.PAREN_C;}

        INIT{intval=0;}
([0−9]
        ACTION{Integer aux=new Integer(yytextchar()+"");
       intval=intval*10 + aux.intValue();}
)+      {System.out.println("Natural: "+intval);
         yylval_update(new Symbol(TokenDef.ENTERO,new ent()));
        ((ent)(yylval().value)).valor.put(Integer.parseInt(yytext(),10));
        return TokenDef.ENTERO;}

        INIT{realval1=0;realval2=0;cantreal=1;}
([0−9]
        ACTION{Integer aux=new Integer(yytextchar()+"");
       realval1=realval1*10 + aux.intValue();}
) + \. ([0−9]
       ACTION{Integer aux=new Integer(yytextchar()+"");
       cantreal=cantreal*10;
       realval2=realval2*10 + aux.intValue();}
)+
     {real=realval1+(realval2/cantreal); System.out.println("Real: "+real);
      yylval_update(new Symbol(TokenDef.REAL,new real()));
      ((real)(yylval().value)).valor.put(new Float(yytext()).floatValue());
      return TokenDef.REAL;}
%%
import output.TokenDef;
import java.lang.Integer;
import java.lang.Float;
import java.io.*;
import java.lang.System;
```

```
%{
class GLOBAL {}
%}
class ent  {int valor;}
class real {float valor;}
class exp  {int valor_i; float valor;}

%token MAS, MENOS, POR, DIVIDE, PAREN_A, PAREN_C
%token <ent> ENTERO
%token <real> REAL
%type <exp> Expr, Factor, Term, Numero
%start S
%%
S : Expr {System.out.println(" El valor es: " + $1.valor);}
  ;
Expr : Expr MAS Term
        ATTRIBUTION { $$.valor = $1.valor + $3.valor; }
  | Expr MENOS Term
     ATTRIBUTION    { $$.valor = $1.valor − $3.valor; }
  | Term
     ATTRIBUTION    { $$.valor = $1.valor ; }
  ;
Term : Term POR Factor
       ATTRIBUTION   { $$.valor = $1.valor * $3.valor; }
  | Term DIVIDE Factor
     ATTRIBUTION    { $$.valor = $1.valor / $3.valor; }
  | Factor
     ATTRIBUTION    { $$.valor = $1.valor ; }
  ;
Factor :  PAREN_A Expr PAREN_C
          ATTRIBUTION { $$.valor = $2.valor ; }
  | Numero
     ATTRIBUTION      { $$.valor = $1.valor ; }
  ;
Numero : ENTERO
    ATTRIBUTION{ $$.valor = $1.valor ;}
  | REAL
     ATTRIBUTION       { $$.valor = $1.valor ;}
  ;
```

Figura 1: Especificación de un analizador lexicográfico (izquierda) y de un Analizador Sintáctico (derecha).

se tiene la definición de la especificación del analizador lexicográfico y sintáctico de un intérprete que reconoce y evalúa expresiones sobre enteros y reales, se puede ver el uso de las ERT y sus ventajas. En el mismo se puede apreciar la utilidad de poder agregar a las expresiones regulares que definen los tokens, traducciones asociadas a cada carácter de entrada. En el ejemplo dichas traducciones permiten evaluar de manera simple y clara el valor de un entero y un real.

4. Trabajos Futuros

Se intentará mejorar el tiempo de ejecución de los procesadores de lenguajes generados por Japlage, único aspecto en que es superado otras herramientas existentes [Agu03]. Para encarar esta mejora se comenzará por analizar la influencia que pueda tener sobre el tiempo de ejecución la cantidad de threads disparado y en caso que esta sea importante se implementarán estrategias que disminuyan la cantidad de threads que ejecuten concurrentemente.

Si fuera necesario se estudiará reimplementar el código de Japlage para hacerlo más eficiente. También se analizará la posibilidad de incorporar al RTS alguno de los generadores de evaluadores de GAs desarrollados en el grupo.
Se han analizado alternativas al mecanismo de scheduling de procesos de JAVA para mejorar el tiempo insumido por la ejecución concurrente de los procesos generados para realizar una mejora de los traductores producidos por Japlage. A través del análisis de varios casos de estudio se descubrió que en el caso de Japlage no hay mejoras relevantes modificando las políticas de scheduling de los procesos de JAVA, sino que el problema radica en el número de threads generados por cada regla de evaluación. En base a estos estudios se planea desarrollar alguna alternativa que disminuya de manera significativa el número de threads y de esta manera obtener una tercera versión optimizada de Japlage.
Otra aspiración es incorporar a Japlage facilidades para generar Proof-Carrying Code (PCC), cuando los avances en esa línea lo permitan [Nec98] [Col00].

Referencias

[Agu99] J. Aguirre, G. Maidana, M. Arroyo. *"Incorporando Traducción a las Expresiones Regulares"*. Anales del WAIT 99 JAIIO, pp 139 - 154, 1999.

[Bav02] F. Bavera, D. Nordio, M. Arroyo y J. Aguirre. *"JTLex: Un Generador de analizadores Léxicos Traductores"*. Anales del CACIC 2002. Univesidad de Buenos Aires - 2002, pp 124 - 134.

[Boy96] J. T. Boyland. *"Conditional Attribute Grammars"*. ACM Transactions on Programming Languajes and Systems, Vol. 18, No. 1, January 1996, pages 73 - 108.

[Fel02] J. Felippa, G. Gomez, V. Grinspan, J. Aguirre, M. Arroyo. *"japlage un entorno de generación de Procesadores de Lenguajes"*. Universidad Nacional de Río Cuarto, 2002.

[Gam95] Gamma, E. et al. *"Design Patterns elements of Reusable Object Oriented Software"*. Addison Wesley, 1995.

[Han98] E. Hans-Erik, M. Penker. *"UML Toolkit"*. John Wiley & Sons, Inc. 1998.

[Agu03] J. Aguirre, R. Medel, M. Arroyo, N. Florio, F. Bavera, P. Caymes Scutari, D. Nordio. *"Avances en Procesadores de Lenguajes y Proof-Carrying Code"*. Wicc 2003.

[Nec98] G. Necula. *"Compiling with Proof"*. Ph.D. Thesis School of Computer Science, Carnegie Mellon Unversity CMU-CS-98-154. 1998.

[Col00] C. Colby, P. Lee, G. Necula, F. Blau, M. Plesko, K. Cline. *"A certifiying Compiler for Java"*. en Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'00) pp. 95-105, ACM Press, Vancouver (Canadá), Junio 2000.