Universidad de La Laguna

Grado en Ingeniería Informática

Computabilidad y Algoritmia

PRÁCTICA 8

Gramáticas en Forma Normal de Chomsky

Factor de ponderación: 9

1. Objetivos

El objetivo de la práctica es consolidar los conocimientos adquiridos sobre Gramáticas [1] al mismo tiempo que se continúan desarrollando capacidades para diseñar y desarrollar programas orientados a objetos en C++. El programa leerá desde un fichero la especificación de una gramática independiente del contexto y a partir de dicha definición generará automáticamente una gramática equivalente en Forma Normal de Chomsky.

Podemos entonces concretar que los objetivos principales de esta práctica son los siguientes:

- Consolidar los conocimientos adquiridos sobre Gramáticas.
- Estudiar y practicar el algoritmo de transformación de una gramática a su forma normal de Chomsky.
- Ampliar las funcionalidades definidas en la clase que se ha desarrollado para representar gramáticas.
- Profundizar en las capacidades de diseñar y desarrollar programas orientados a objetos en C++.

2. Introducción

En teoría de lenguajes formales se dice que una gramática independiente del contexto $G \equiv (V, \Sigma, S, P)$ está en forma normal de Chomsky (en honor a quien propuso este formato, CNF, *Chomsky Normal Form*) si:

- V sólo contiene símbolos útiles.
- Todas las producciones de G tienen una de las formas:
  $A \to BC$
  $A \to a$

Si $\epsilon \in L(G)$ se permite además una única producción $S \to \epsilon$ y en este caso no se permite que el símbolo de arranque figure en la parte derecha de ninguna regla de producción.

Al margen del interés teórico que tiene la forma normal de Chomsky (se utiliza por ejemplo en la demostración del lema del bombeo para lenguajes independientes del contexto), la conversión a CNF se utiliza como paso previo de otros algoritmos, como el algoritmo de análisis sintáctico (algoritmo de Cocke, Younger y Kasami, CYK [4]).

Cualquier gramática independiente del contexto puede ser transformada en una equivalente escrita en forma normal de Chomsky. Por otra parte, se puede demostrar que en una Gramática escrita en CNF, la derivación de una cadena de n símbolos tiene exactamente *2n - 1* pasos de derivación. Esta propiedad permite determinar si una cadena $w \in L(G)$ mediante una inspección exhaustiva de todas las derivaciones.

El Algoritmo 1 aplicado a una gramática independiente del contexto permite su transformación a forma normal de Chomsky.

Algorithm 1 Transformación de una CFG a su Forma Normal de Chomsky

1: **for all** $(A \to X_1X_2 \dots X_n$ (con $n \geq 2$, $X_i \in (\Sigma \cup V))$ **do**
2: **for all** (X_i) **do**
3: **if** $(X_i = a \in \Sigma)$ **then**
4: Add the production $C_a \to a$;
5: Replace X_i with C_a in $A \to X_1X_2 \dots X_n$;
6: **end if**
7: **end for**
8: **end for**
9: **for all** $(A \to B_1B_2 \dots B_m$ (con $m \geq 3$, $B_i \in V)$ **do**
10: Add $m - 2$ non-terminal symbols $D_1D_2 \dots D_{m-2}$;
11: Replace the production $A \to B_1B_2 \dots B_m$ with productions:
12: $A \to B_1D_1$
13: $D_1 \to B_2D_2$
14: $\dots$
15: $D_{m-2} \to B_{m-1}B_m$
16: **end for**

Es importante tener en cuenta que **el algoritmo toma como entrada una gramática que no contenga símbolos ni producciones inútiles, unitarias ni vacías**, y produce como salida una gramática equivalente a la de entrada, pero escrita en forma normal de Chomsky.

Considérese a modo de ejemplo la gramática definida por las siguientes producciones:

$S \to aXbX \mid abX \mid aXb \mid ab$

$X \to aY \mid bY \mid a \mid b$

$Y \to aY \mid bY \mid a \mid b \mid c$

El primer bucle del algoritmo 1 (líneas 1–8) introduce dos nuevos símbolos no terminales C_a y C_b y sus correspondientes producciones, de modo que la gramática se transforma en:

$S \to C_aXC_bX \mid C_aC_bX \mid C_aXC_b \mid C_aC_b$

$X \to C_aY \mid C_bY \mid a \mid b$

$Y \to C_aY \mid C_bY \mid a \mid b \mid c$

$C_a \to a$

$C_b \to b$

Finalmente el segundo bucle del algoritmo (líneas 9–16) transforma la gramática en:

$S \to C_aD_1 \mid C_aE_1 \mid C_aF_1 \mid C_aC_b$

$X \to C_aY \mid C_bY \mid a \mid b$

$Y \to C_aY \mid C_bY \mid a \mid b \mid c$

$C_a \to a$

$C_b \to b$

$D_1 \to XD_2$

$D_2 \to C_bX$

$E_1 \to C_bX$

$F_1 \to XC_b$

que ya está representada en forma normal de Chomsky. Estas referencias [5] [6] presentan ejemplos adicionales de conversión a CNF.

3. Ejercicio práctico

Desarrollar un programa `Grammar2CNF.cc` que lea un fichero `.gra` en el que figura la especificación de una gramática independiente del contexto y genere otro fichero `.gra` en el que se especifique una gramática equivalente ($L(G_{in}) = L(G_{out})$) a la de entrada, pero en este caso, escrita en forma normal de Chomsky. Antes de realizar la conversión se deberá comprobar al menos que la gramática de entrada G_{in} no contiene producciones unitarias ni vacías. Por simplicidad, se podrá omitar la comprobación de símbolos y producciones inútiles. La conversión de la gramática de entrada G_{in} a forma normal de Chomsky se realizará mediante la aplicación del Algoritmo 1.

El comportamiento del programa al ejecutarse en línea de comandos debiera ser:

```
$ ./Grammar2CNF
Modo de empleo: ./Grammar2CNF input.gra output.gra
Pruebe ‘Grammar2CNF --help’ para más información.
```

Donde `input.gra` y `output.gra` son los ficheros que especifican las gramáticas de entrada y salida respectivamente. La opción `--help` en línea de comandos ha de producir que se imprima en pantalla un breve texto explicativo del funcionamiento del programa.

Los ficheros de especificación de gramáticas son ficheros de texto plano con extensión `.gra` que contienen los elementos definitorios de la gramática $G \equiv (\Sigma, V, S, P)$ en este orden: símbolos terminales, símbolos no terminales, símbolo de arranque y producciones. El formato de cada uno de estos elementos en el fichero es el siguiente:

1. **Símbolos terminales (alfabeto)**: una línea que contiene N, el número de símbolos en el alfabeto seguida de N líneas, cada una de las cuales contiene un símbolo del alfabeto. Cada símbolo del alfabeto debe ser un único caracter imprimible.
2. **Símbolos no terminales**: una línea que contiene V, el número de símbolos no terminales, seguida de V líneas, cada una de las cuales contiene una cadena alfanumérica sin espacios.
3. **Símbolo de arranque**: una única línea que contiene el símbolo de arranque, S, de la gramática. Ha de ser uno de los símbolos no terminales relacionados anteriormente.
4. **Producciones**: una línea que contiene P, el número de producciones de la gramática, seguida por P líneas cada una de las cuales contiene una producción en el formato:

   `A ->` α

   siendo $\alpha \in (\Sigma \cup V)^*$, es decir una secuencia de símbolos terminales y no terminales. La cadena vacía, ϵ se representa mediante el carácter &.

4. Criterios de evaluación

Se señalan a continuación los aspectos más relevantes (la lista no es exhaustiva) que el profesorado tendrá en cuenta a la hora de evaluar el trabajo que el alumnado presentará en la sesión de evaluación de la práctica:

- Se valorará que el alumnado haya realizado, con anterioridad a la sesión de prácticas, y de forma efectiva, todas las tareas propuestas en este guión. Esto implicará que el programa compile y ejecute correctamente.
- También se valorará que, con anterioridad a la sesión de prácticas, el alumnado haya revisado los documentos que se enlazan desde este guión.
- El comportamiento del programa debe ajustarse a lo solicitado en el enunciado.
- El programa diseñado ha de ser fiel al paradigma de programación orientada a objetos. Se valorará que el alumnado haya identificado clases y objetos que permitan modelar adecuadamente el escenario de trabajo que se plantea.
- Se valorará que el programa se haya escrito de modo que las diferentes funcionalidades que se precisen hayan sido encapsuladas en métodos concretos cuya extensión textual se mantuviera acotada.
- El programa ha de ceñirse al formato de escritura de programas adoptado en las prácticas de la asignatura. Esto implica, entre otros, que el código desarrollado ha de estar escrito de acuerdo al estándar de la guía de estilo de Google para C++ [14].
- El programa desarrollado deberá compilarse utilizando la herramienta make [12, 13] y un fichero Makefile, separando los ficheros de declaración (`*.h`) y definición (`*.cc`) de clases.
- En la sesión de evaluación de este trabajo, todos los ficheros con códido fuente se han de alojar en un único directorio junto con el fichero Makefile de compilación.
- Se requiere que todos los atributos de las clases definidas en el proyecto tengan un comentario descriptivo de la finalidad del atributo en cuestión. Además, se requiere que los comentarios del código fuente sigan el formato especificado por Doxygen [9]. Utilice asimismo esta [10] referencia para mejorar la calidad de la documentación de su código fuente.
- Finalmente, se analizará la capacidad del programador(a) de introducir cambios en el programa desarrollado.

Si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas allí. Se espera que, a través de ese foro, el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas. También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

Referencias

[1] Transparencias del Tema 3 de la asignatura: Lenguajes y Gramáticas Independientes del Contexto, https://campusingenieriaytecnologia2223.ull.es/mod/resource/view.php?id=5932

[2] Context Free Grammar, https://en.wikipedia.org/wiki/Context-free_grammar

[3] Chomsky Normal Form https://en.wikipedia.org/wiki/Chomsky_normal_form

[4] CYK algorithm https://en.wikipedia.org/wiki/CYK_algorithm

[5] Conversion of CFG to Chomsky Normal Form https://tinyurl.com/cnf-example

[6] Chomsky Normal Form https://courses.cs.washington.edu/courses/cse322/08au/lec14.pdf

[7] Noam Chomsky https://en.wikipedia.org/wiki/Noam_Chomsky

[8] Chomsky hierarchy https://en.wikipedia.org/wiki/Chomsky_hierarchy

[9] Doxygen http://www.doxygen.nl/index.html

[10] Diez consejos para mejorar tus comentarios de código fuente https://www.genbeta.com/desarrollo/diez-consejos-para-mejorar-tus-comentarios-de-codigo-fuente

[11] Cómo identificar clases y objetos http://www.comscigate.com/uml/DeitelUML/Deitel01/Deitel02/ch03.htm

[12] Makefile Tutorial: http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor

[13] C++ Makefile Tutorial: https://makefiletutorial.com

[14] Google C++ Style Guide, https://google.github.io/styleguide/cppguide.html