Memoria Práctica Modelos

Ignacio Tirado, David Suárez grupo 101 de GISI

1. Introducción a la práctica

Esta práctica busca utilizar los conocimientos vistos en Programación I y II , y la teoría de gramática vista en clase de modelos. En la práctica hemos utilizado conceptos como reglas no generativas, símbolos inútiles, reglas de chomsky, reglas innecesarias, reglas unitarias y algoritmos de decisión.

2. Decisiones de diseño tomadas en la práctica y cómo se han implementado.

¿Cómo ha organizado los diferentes conceptos: elementos terminales, no terminales, axioma, producciones?

Los elementos terminales y los no terminales los hemos organizado como un set de caracteres, el axioma lo hemos organizado como un carácter, las producciones las hemos organizado como un mapa de caracteres y una lista de strings.

¿Qué estructuras de datos hemos utilizado?

Hemos utilizado mapas, conjuntos, listas y arrays.

¿Cómo es el acceso a las mismas?

Para acceder a la mayoría de las veces, utilizamos los bucles for para recorrer las estructuras.

3. Descripción de cada una de las pruebas de tests que el alumno ha definido

David ha ayudado en el razonamiento de los siguientes métodos de Ignacio: removeNonTerminal, removeProduction, getProductionstoString, getGrammar, deleteGrammar, removeUselessProductions. Además, como está comentado en el commit “removeLambdaProductions+CNF”, hubo un problema de flujo de repositorios en el test 3 debido a una falta de organización. Hicimos los 2 los mismos métodos del test 3.

TEST 1

addNonTerminal
Utilizamos un condicional para comprobar que la letra no esté repetida en la gramática, si esto sucede se lanza una excepción. Posteriormente, lanzamos otro condicional para comprobar que la letra es mayúscula y se añade a la gramática. Por último, utilizamos un else por si se hace que se lance una excepción si la letra no es mayúscula.

removeNonTerminal
Utilizamos dos bucles for para recorrer la lista de producciones y los no terminales de las producciones, añadimos una lista auxiliar ya que no podemos modificar la lista de producciones mientras la estamos recorriendo. Las producciones que estén en la lista auxiliar se borrarán. Si la producción contiene un no terminal y el no terminal pertenece a la gramática, se eliminará.

getNonTerminals
Retornamos el set de caracteres de los no terminales.

addTerminal
Utilizamos un condicional para comprobar que la letra no esté repetida en la gramática, si esto sucede se lanza una excepción. Sino se lanza un condicional que comprueba que la letra sea minúscula y se añade a la gramática. Por último, si ninguna de estas condiciones se cumple se lanza la excepción.

removeNonTerminal
Utilizamos un bucle que recorra los no terminales del mapa producciones, creamos una lista auxiliar, usamos un bucle que recorre las producciones del mapa, utilizamos un condicional que comprueba si la producción contiene al terminal y añade la producción a la lista auxiliar. Posteriormente, utilizamos un bucle que muestra el contenido de la lista auxiliar y elimina la producción. Después, utilizamos un condicional que comprueba si el terminal está en el conjunto y lo elimina de él. Por último, si no se cumple esta condición lanzamos una excepción si no está contenido en los terminales.

getTerminals
Retornamos el set de caracteres de los terminales.

setStartSymbol
Utilizamos un condicional que comprueba si el no terminal está comprendido en el conjunto de no terminales, si lo está fijamos ese no terminal como axioma, sino lanzamos una excepción ya que no pertenecerá al set de caracteres de no terminales.

getStartSymbol
Utilizamos un condicional que comprueba si el axioma está vacío, sino lo está lo retorna sino retorna una excepción debido a que el axioma no habrá sido establecido.

addProduction
Utilizamos un condicional que comprueba si está contenido o no el no terminal en el conjunto, si no lo está lanza una excepción. Posteriormente, utilizamos un bucle que recorre letra por letra la producción introducida. Dentro de un bucle for, utilizamos un condicional que comprueba que si la caracter está dentro de los terminales, no terminales, o sea lambda y si no se cumple se lanza una excepción. Después del bucle for, utilizamos un “putIfAbsent” que comprueba que no esté el no terminal no esté repetido y lo añade. Por último el inverse para obtener todas las producciones para añadir para luego finalmente añadirlas.

removeProduction
Utilizamos un bucle for que recorre la lista de producciones del no terminal introducido. Dentro del bucle for utilizamos un condicional que consigue la producción de la posición por la que va en el bucle que recorre la lista de producciones y comprueba si la producción por la que va en el bucle es igual a la producción que buscamos. Dentro del condicional, eliminamos la producción del no terminal introducido después como es un boolean devolvemos true. Si el “return true” no se ejecuta, entonces se ejecutará al salir del bucle for una excepción que muestre que no hay ninguna producción en la lista de producciones igual a la producción introducida.

getProductions
Retornamos la lista de producciones del no terminal introducido.

getProductionsToString
Señalamos la lista del map productions. Después, utilizamos un condicional que devuelve “ “ si es null, si la lista no es null ordenamos la lista. Posteriormente, ejecutamos un bucle for que recorrerá cada elemento de la lista y los separará con una barra. Al final si el bucle for se ha ejecutado, retornaremos nonterminal con el string symbol “::=”.

getGrammar
Primero, creamos una variable vacía, guardamos los terminales en la variable, guardamos los no terminales en la variable. Posteriormente, ejecutamos un bucle que recorre los no terminales del conjunto, por cada iteración del bucle guardamos las producciones en la variable. Después de salir del bucle for, utilizamos un condicional que comprueba que el axioma no sea nulo, si no es nulo guardamos el axioma en la variable. Por último retornamos el string grammar.

deleteGrammar
Utilizamos el .clear() de la siguiente manera, borramos el contenido el contenido de los no terminales, borramos el contenido de los terminales, borramos el contenido de las producciones. Por último borramos el valor de la axioma y da null.

isCFG
Retornamos el valor true ya que todas las gramáticas que introduzcamos van a ser independientes del contexto.

TEST 2

hasUselessProductions
Utilizamos dos bucles foreach, el primero devolverá las claves del mapa productions y las guardará en el nonterminal. El segundo devolverá los valores de productions y las guardará en production. Dentro del foreach, utilizaremos un condicional que nos dirá si se cumple que production es igual a nonterminal. Si se cumple retornará true y los 2 bucles foreach se romperán. Si no se llegará a cumplir esto último, al salir de los 2 bucles for each, se retornará false.

removeUselessProductions
Creamos una lista de eliminados y una lista que devolveremos. Utilizamos los 2 bucles foreach, que nos darán los no terminales y las producciones de productions. Dentro de los 2 bucles foreach, utilizamos un condiciona que nos dirá si se cumple que production es igual a nonterminal. Si se cumple lo añadiremos a lista de eliminados, ya que una lista no se puede modificar mientras se recorre un bucle de esa misma lista, posteriormente añadimos a la lista que devolveremos el no terminal seguido de un "::=" con la producción de ese no terminal. Al salir del segundo bucle for each eliminaremos la lista del mapa los que coincidan con la lista de eliminados. Por último, al salir del primer foreach retornaremos la lista que íbamos a devolver.

removeUselessSymbols
Basado en el pseudocódigo del pdf de la práctica, pasado de pseudocódigo a código.

hasLambdaProductions
Usaremos un "inverse" para obtener todas las producciones que generan lambda.

removeLambdaProductions
Declararemos un set de caracteres anulables y una lista de caracteres resultado. El primer paso es identificar los nos terminales que se puedan transformar en lambda. Utilizamos un do while en el cual mientras cambio sea falso se seguirá ejecutando. Dentro del do hay 2 bucles for each que recorren todo el mapa de producciones, si hay alguna producción que sea igual a lambda. Luego, utilizamos un condicional en el que si el no terminal no está ya en el set de caracteres anulable, se añade. Si se añade un nuevo no terminal a anulable cambio retornará true. El segundo paso es generar todas las combinaciones posibles. Creamos un mapa que contenga las nuevas producciones. Hacemos un bucle foreach,declaramos una lista de strings produccionAuxiliar, hacemos el otro bucle foreach para recorrer el mapa, si la producción es igual a lambda, la añadimos a la lista de produccionAuxiliar. Utilizamos un bucle for que itera sobre cada carácter. Si el carácter es un no terminal que puede derivar la cadena vacía / anulable, generamos una producción eliminando dicho carácter de la producción original. Si la nueva producción no está vacía y no está ya en produccionAuxiliar, se añade a la lista. Una vez que se han procesado todas las producciones del no terminal actual, se asocia el no terminal con la lista produccionAuxiliar en el nuevo mapa nuevasProducciones. En el tercer paso añadimos la transformación de lambda al axioma en el caso de que sea necesario. Por último actualizamos las producciones y devolvemos los no terminales que tenían producciones lambda.

hasUnitProductions
Utilizamos 2 bucles foreach para recorrer el mapa productions, después utilizamos un foreach que itere por cada carácter, si la letra es no terminal retorna true. Si el return true no se cumple al finalizar los 3 bucles foreach, retornará false.

TEST 3

checkCNFProduction
Utilizamos un condicional que comprueba el caso de producción igual a lambda, dentro de ese condicional usamos otro condicional que comprueba si el no terminal es el axioma, si no lo es lanza una excepción. Después utilizamos un else if que compruebe que la producción tenga longitud 1, usamos un condicional que compruebe si es minúscula la letra en la posición 0, si no lo es lanza una excepción. Posteriormente utilizamos otro else if que compruebe que la producción tiene longitud 2, usamos un condicional que compruebe si es minúscula la letra en la posición 0 o 1, si no lo es lanza una excepción. Por último si ninguna de las condiciones anteriores se cumple, se lanza una excepción.

isCNF
Utilizamos un doble bucle foreach que recorra todo el mapa de productions. Utilizamos un condicional que comprueba el caso de producción igual a lambda, dentro de ese condicional usamos otro condicional que comprueba si el no terminal es el axioma, si no lo es lanza una excepción. Después utilizamos un else if que compruebe que la producción tenga longitud 1, usamos un condicional que compruebe si es minúscula la letra en la posición 0, si no lo es lanza una excepción. Posteriormente utilizamos otro else if que compruebe que la producción tiene longitud 2, usamos un condicional que compruebe si es minúscula la letra en la posición 0 o 1, si no lo es lanza una excepción. Por último si ninguna de las condiciones anteriores se cumple, se lanza una excepción.

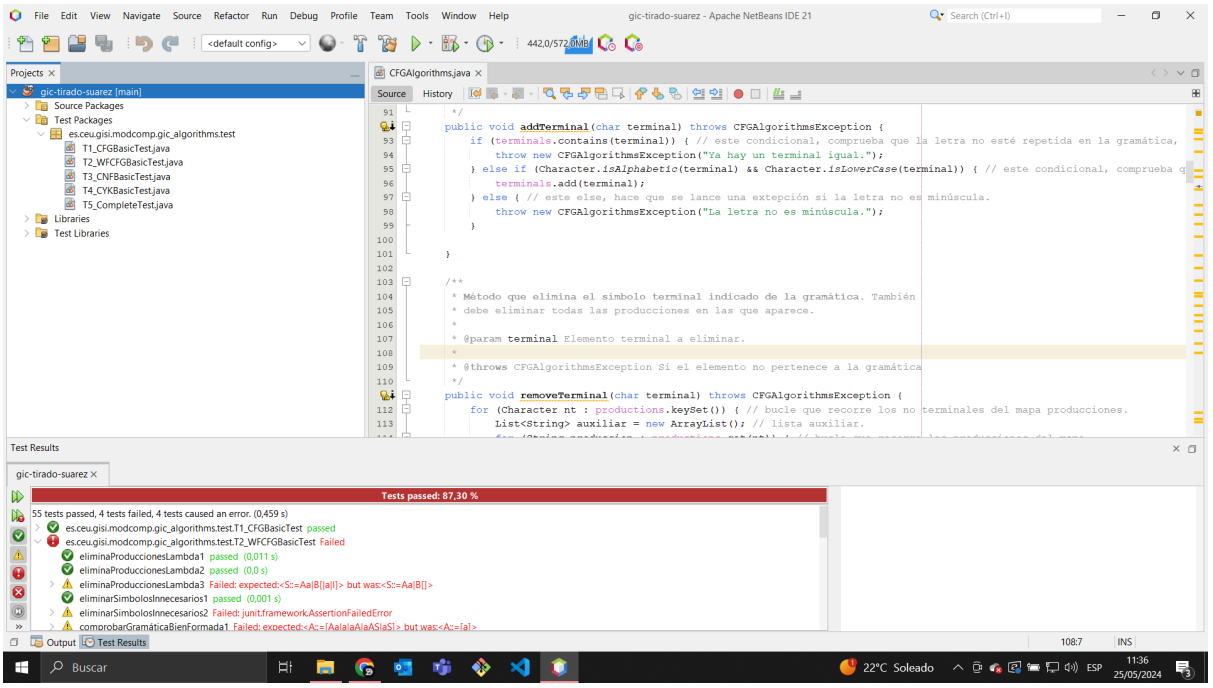
TEST 4

isDerivedUsingCYK
Utilizamos un condicional para comprobar si la gramática está en forma normal de chomsky, si no lo está lanzamos una excepción. Posteriormente, usamos otro condicional para comprobar si la palabra está vacía, si lo está retornamos false. A continuación usamos un bucle for que itere la longitud de la palabra introducida, dentro del bucle for utilizamos un condicional para comprobar si la letra de la palabra por la que va iterando es un terminal o no, si no lo es se lanza una excepción. Después, utilizamos un condicional que compruebe si el mapa productions está vacío o el axioma es null, si alguna de estas dos condiciones se cumple se lanza una excepción. Después, declaramos un int n que tiene el valor de la longitud de la palabra, un int r que tiene el tamaño de los no terminales y un set de caracteres table. Utilizamos un bucle for que itere n veces, por cada iteración cambiamos el valor del terminal al valor del carácter cuya posición es por la que va iterando el bucle en ese momento, luego declaramos otro bucle for que itera sobre cada entrada en el mapa productions. Cada entrada es un par clave-valor, donde la clave es un no terminal y el valor es una lista de producciones. Extraemos la clave de la entrada actual y lo almacenamos en una variable nonterminal. Hacemos un bucle foreach que itera sobre cada producción en la lista de producciones asociada al no terminal actual. Utilizamos un condicional que compruebe si la longitud de production es 1 o si el carácter 0 de la producción es terminal, si alguna de estas dos condiciones se cumple, añadimos el no terminal a una lista dentro de

una estructura de una tabla. Utilizamos un bucle for que itere comenzando desde l=2 hasta n (la longitud de la cadena). Utilizamos un bucle for que itere comenzando desde s=0 hasta n - l , Inicializamos una nueva entrada en la tabla CYK en la posición (s)(l-1) como un conjunto vacío. Utilizamos un que itere comenzando desde 1 hasta l, obtenemos los conjuntos de no terminales para las dos partes del subsegmento dividido. Utilizamos dos for que iteren sobre todos los no terminales en bset y cset. Para cada par de no terminales b y c iteraremos todas las producciones de la gramática. Por último, el return funcionará si table[0][n-1] contiene el axioma.

4. Completitud de los tests definidos en el proyecto y los definidos por el alumno

Porcentaje de todos los test :



5. Conclusiones

Ambos hemos utilizado 10 de nuestras horas trabajando en grupo y las otras 20 horas han sido en nuestras casas.