Práctica 6

IPN Escuela Superior de Cómputo
Ing. En Sis. Computacionales

Compiladores 3CV5
M.C Saucedo Delgado Rafael Norman

2014090186
Elizalde Díaz Roberto Carlos

Introducción

Esta práctica consiste en el diseño e implementación de algoritmos para la construcción de una tabla LL1 asociada a una gramática libre de contexto sin recursión a la izquierda.

La entrada del programa es una gramática con las características mencionadas y su salida será la tabla LL1 asociada.

Para la realización de esta práctica se utilizó el lenguaje Java en su versión libre (openjdk) 11.0.9.1 y Netbeans 11 como IDE, corriendo en el sistema operativo Ubuntu 20.04.1 LTS.

Desarrollo

Para el desarrollo de esta práctica no se utilizó metodología específica, sólo fue un análisis de la implementación y diseño de un algoritmo para la construcción de la tabla LL1.

En esta se encuentran implementadas las siguientes clases:

- LL1: Clase donde están implementados todos los pasos a realizar para la construcción de la tabla LL1.
- Production: Clase que mapea la definición básica de una producción.
- Grammar: Clase que mapea la definición de una gramática libre de contexto.
- AuxiliaryFunctions: Clase en donde se encuentra las funciones para calcular primero(first) y siguiente(follow)

Clase Production
Esta clase mapea lo que entendemos por una producción. Consta de dos miembros pasados como argumentos al constructor que son llamados:

1. char head
2. String body

Donde ***char head*** es una variable a la que define (parcialmente) una producción y ***String body*** es una cadena formada por cero o más símbolos terminales y variables. Esta variable es también denominada cuerpo.

Clase Grammar
Esta clase mapea lo que un gramatica en la que se incluye un lista de producciones(List<Productions>) y algunos métodos auxiliares para la manipulación de esta, como los son:

- ***public void addProduction(char head, String body)*** : Método que permite añadir producciones a la gramática.
- ***public List<Production> getProductions():*** Método para obtener todas las producciones de una gramática
- ***public int getNumOfProductions()***: Método para obtener el número total de producciones de una gramática
- public Production getProduction(int i): Método para obtener la i-ésima producción.

- ***public String getNoTerminals():*** Método para obtener los símbolos no terminales de la gramática. Se obtiene un String donde cada caracter representa un símbolo no terminal.
- ***public String getTerminals():*** Método para obtener los símbolos terminales de una gramática. Se obtiene un String donde cada caracter representa un símbolo terminal.
- ***private boolean hasSymbol(char c, StringBuilder s):*** Método para saber si un caracter 'c' se encuentra en una cadena 's'.

Para diferenciar los caracteres entre terminales, no terminales y épsilon, se utilizan las constantes: TERMINAL, NO_TERMINAL y EPSILON,

Por defecto, la primera producción es tomada como la producción inicial.

Clase AuxiliaryFunctions
Esta clase es la que se utiliza para el cálculo de las funciones primero(first) y siguiente(follow).

Al constructor de esta clase se le pasa como argumento un objeto Grammar para indicar que es esta la gramática con la que tiene que trabajar.

Los métodos principales para darle la funcionalidad a esta clase son los siguientes:

- public String first(String symbol) y public String first(String symbol, List<Character> alreadyProcessed): Estados dos métodos hacen ejecutan la función first(). El primer método manda a llamar a segundo pasándole como argumento el símbolo al que se le desea sacar first() y un ArrayList que se utilizará para guardar el rastro de los símbolos de los no terminales por los cuales ya se han aplicado first(), esto como medida implementada para evitar ciclos.
- public String follow(String symbol) y public String follow(String symbol , List<Character> alP) al igual que el método anterior el ArrayList para evitar ciclos es pasado como argumento. Este método realiza la función follow() a un símbolo(String), este último puede ser solo un carácter o una cadena donde se pueda encontrar terminales y no terminales.
- public boolean isCharFound(char c, StringBuilder b): Método para conocer si un caracter 'c' se encuentra en la cadena 'b'.
- private int getType(char symbol): Método para obtener el tipo de símbolo, Este puede ser épsilon, no terminal o terminal. Para esto se utilizan las constantes TERMINAL, NO_TERMINAL y EPSILON, declaradas dentro de la clase.

- private boolean isAlreadyProcessed(char c, List<Character> alreadyProcessed): Método que evalúa si un símbolo no terminal ha sido procesado(Evitar ciclos).
- public boolean hasEpsilon(String s): Método que detecta si épsilon se encuentra en una cadena, ya sea en el cuerpo de un producción o en el resultado de las operaciones first() y follow()
- private boolean isInitial(char c): Método para sabes si un caracter 'c' es la cabeza de la producción inicial.

Clase LL1

Esta clase contiene un algoritmo de llenado de la tabla LL1 en done su usarán las clases anteriores para este propósito. Los argumentos de esta clase serán solamente un objeto Grammar que contenga la gramática del que desea construir su tabla LL1.

Esta clase también cuenta con un clase interna llamada CellLL1 el cual es un objeto que representa cada celda de la tabla LL1, es decir, si la tabla resultante es de 3x8 celdas, entonces se tendrán que usar 3x8 objetos CellLL1. Estos objetos tienen como miembros:

- char terminal: Almacena el caracter del terminal de esa columna.
- char no terminal: Almacena el caracter del no terminal de esa fila.
- List<Productions>: Lista de producciones que pertenecen a cada celda de la tabla LL1.

Este clase interna cuenta con un solo método para poder agregar producciones.

Los métodos de la clase LL1 más importantes son:

- makeLL1Table(): Este método comienza inicializando nuestra tabla, la cual es una lista de listas, con los caracteres asociados a los símbolos no terminales y los terminales de la siguiente forma.

  Lista1

  (NoTerminal1).....(NoTerminal1) ……..(NoTerminal1)

  (Terminal1) (Terminal2) (Terminaln)

  (NoTerminal2).....(NoTerminal2) .

  (Terminal1) (Terminal2) .

  (NoTerminal3).....(NoTerminal3).....(NoTerminalN)

  (Terminal1) (Terminal2) (Terminaln)

Posteriormente se emplean un objeto AuxiliarFunctions para calcular first() y follow() según sea el caso para llenar cada celda.

- private void fillCell(Production production, char noTermi, char termi): Método para agregar una producción, el cual se basará en los caracteres noTermi y termi para buscar el objeto CellLL1 en nuestra tabla LL1 y en ese objeto agregar la producción 'production'.
- public void showLL1Table(): Método para imprimir en pantalla la tabla LL1 resultante. Este método recorre cada objeto CellLL1 e imprime la siguiente estrucura por cada CellLL1 que encuentre:
  | NoTerminal , Terminal, Producciones (separadas por espacio) |
  | NoTerminal , Terminal, ~(Representa celda con 0 producciones)|

Pruebas

Para la pruebas realizadas se utilizó el siguiente ejemplo visto en clase:

Gramática

S -> abC (1)
C -> Sab (2)
C -> bSA (3)
A -> aSC (4)
A -> a
S -> c

Tabla

	a	b	c	$
S	1		6	
A	⅘			
C	2	3	2	

Para empezar hacer funcionar el programa creamos un objeto Grammar y empezamos a agregar las transiciones de la gramática de la siguiente forma:

```
Grammar grammar = new Grammar();
grammar.addProduction('S', "abC");
grammar.addProduction('C', "Sab");
grammar.addProduction('C', "bSA");
grammar.addProduction('A', "aSC");
grammar.addProduction('A', "a");
grammar.addProduction('S', "c");
```

Y posteriormente pasamos el objeto grammar como argumento del constructor del la clase LL1 e imprimimos para ver el resultado

```
|S, a,  S->abC|  |S, b, ~|  |S, c,  S->c|  |S, $, ~|

|C, a,  C->Sab|  |C, b,  C->bSA|  |C, c,  C->Sab|  |C, $, ~|

|A, a,  A->aSC A->a|  |A, b, ~|  |A, c, ~|  |A, $, ~|
```

	a	b	c	$
S	S->abC		S->c	
C	C->Sab	C->bSA	C->Sab	
A	A->aSC A->a			

Conclusiones

Bajo la cantidad de requerimientos que se tenían que desarrollar en esta práctica como las implementaciones de las funciones first() y follow() se nos anticipó que era la práctica más complicada y de más larga duración de todas, sin embargo en mi caso puedo decir con seguridad que fue la segunda pues más allá de la cantidad de cosas a implementar, no encontré demasiada dificultad de implementarlas, pues muchas de ellas ya estaban planteadas en forma de reglas como lo son de las funciones auxiliares y con los ejemplos propuestos y resueltos en clase puede retener algunas ideas para desarrollar el algoritmo de llenada de manera más sencilla. Junto que las anteriores, me siento satisfecho de poder saber las entrañas de un compilador y no solo manejar herramientas que internamente hagan lo que esta práctica, si no también que tuve la capacidad de implementar aunque sea de forma básica, un módulo de estas herramientas.

Referencias

- Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman, Compiladores: principios, técnicas y herramientas, Segunda Edición.
- Documentación ArrayList Java 8 https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- Videos de clase