Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en Sistemas Computacionales

Práctica 6: Implementación del algoritmo LL1.

Compiladores

Profesor: M.C Rafael Norman Saucedo Delgado
Alumno: Iris Yonitzi Marínez González Boleta: 2015090419
irismartinezgonzalez@gmail.com
26 de enero de 2021

Índice

1. Introducción

Las **gramáticas** son aquellas que definen y manejan el lenguaje, están compuestas por 4 elementos:

$$G = T, N, P, S$$

Donde :

- T = símbolos finales o terminales.
- N = símbolos no terminales.
- P = producciones.
- S = símbolo inicial que sale de los no terminales.

Tenemos otras gramáticas que se llaman o se definen como **gramáticas ambiguas**, las cuales son aquellas que tiene más de un árbol sintáctico/derivación para una cadena. Por su parte las gramáticas libres de contexto son aquellas que sus producciones P, están formadas por la siguiente forma:

$$P = N- > (N|T)+$$

Donde :

- N = símbolos finales o terminales.
- T = símbolos no terminales.

En pocas palabras,una gramática es una notación formal que sirve para expresar definiciones recursivas de los lenguajes, tienen 4 componentes, es por eso que es una 4ta tupla.

Dicho esto repasemos un tema.

Analizadores sintácticos

Los analizadores sintácticos usan mucho las gramáticas para llevar su función, la cual es verificar el orden de los tokens que resive como entrada, existen dos tipos de nalizadores, para esta práctica nos enfocaremos en los descendentes.

Analizadores sintácticos Descendentes: son aquellos que su orden es de la raíz a las hojas, tienen problemas con las gramáticas recursivas por la izquierda, estas gramáticas causan que los analizadores descendentes mueran, es decir se ciclan .

Los analizadores LL(1) solo funcionan con gramáticas de tipo LL(1) y ocupan una tabla LL(1) para validar si una cadena pertenece a la gramática dada. Es una automatización de los analizadores sintácticos por descenso recursivo.

- L Left to right: entramos de izquierda a derecha
- L Leftmost derivation: busca derivación por la izquierda
- 1 Número de símbolos de derivación

Funciones auxiliares: Primero() y Siguiente():

- $Primero()$: las cuales se siguen de 3 reglas básicas:
  - $Primero(T) = T$ si es un no terminal.
  - $Primero(N)$ = para toda produccion $N- > \alpha$ agregar épsilon sí $N- > ésilon$.
  - $Primero(\alpha)$ = agregar $Primero(\alpha)$ sí $Primero(\alpha_n)$ produce épsilon y si todas la producen entonces agregar épsilon.
- $Siguiente()$:las cuales se siguen de 4 reglas básicas:
  - $Siguiente(S) = S$ si es el inicial entonces agregar \$.
  - $Siguiente(X)$ =buscar en todas las producciones la forma $A- > \alpha X$ y para cada una agrergar el $Siguiente(A)$.
  - $Siguiente(X)$ = buscar en todas las producciones la forma $A- > \alpha X\beta$ y para cada una agrergar el $Primero(\beta)$.
  - $Siguiente(X)$ = si X es un no terminal, buscar en todas las producciones la forma $A- > \alpha X\beta$ y si épsilón esta en $Primero(\beta)$ agregar $Siguiente(A)$.

El objetivo de esta práctica es implementar el algoritmo $LL1$ con ambas funciones, siguiendo el paradigma Orientado a Objetos para general la tabla $LL1$ de dicho algoritmo.

2. Desarrollo

Para esta practica primero se analizó que necesita el algoritmo para funcionar, es este caso el algoritmo necesita de una gramática con sus producciones correspondientes, esto indica que necesitamos dos clases una que se llame Gramática y la otra Producción, una gramática esta compuesta de muchas o al menos una producción. Debemos tener en cuenta que debe de a ver una clase que lea la gramática que vamos a trabajar.

Ahora tenemos una clase que se llama tabla, vi que era necesaria ya que es el propósito del algoritmo, entonces la pregunta es ¿Quién crea esa tabla?, respuesta, el algoritmo $LL1$ con las funciones de *Primero* y *Siguiente* según sea necesario, y ahí esta la última clase que se necesita. El diagrama de clases después del análisis es el siguiente:

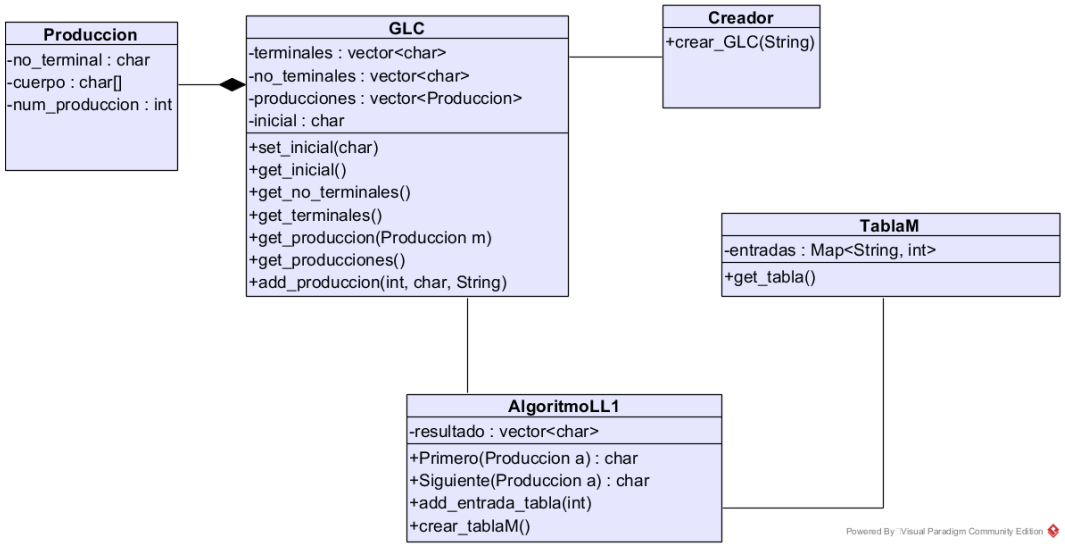

Figura 1: Diagrama de Clases

Crear el maquetado de las clases es sencillo, el problema es cuando ya metemos el algoritmo a funcionar, al principio quise separa el propio algoritmo en tres clases, es decir, tener una que se llama $AlgoritmoLL1$ y que esta llamara a las clases *Primero* y *Siguiente*, cada clase con su correspondiente método un por cada regla, pero el problema con esto es que se hacia algo tedioso y más al momento de detectar siclos se volvió complicado, es por eso que preferí dejar todo dentro de la clase Algoritmo y los métodos volverlos recursivos.

Este ultimo no se si cumpla con el paradigma orientado a objetos, ya que las funciones recursivas si las he tratado pero los métodos recursivos no.

Ahora procedamos a ver la lógica del método recursivo para la función de *Primero*():

```
//resivimos el no terminal de la produccion y su alpha
   public String calculaPrimero(char noterminal , String alpha){
        String resultado ="";
        int i=0,j=0;
        while (i<alpha.length()) {
          if(97<=alpha.charAt(i) && alpha.charAt(i)<=122 || alpha.charAt(i)==69){
            //si es la regla 1 de los primeros Terminal
                resultado = resultado + alpha.charAt(i);
                break;
            } else if (65 <= alpha.charAt(i) && alpha.charAt(i) <= 90) {
            //si es la regla 2 de los primeros  No Terminal
                //leeemos otra vez toda la gramatica
                while (j < numproducciones) {
                    Produccion a = new Produccion();
                    a = gramat.obtenerProduccion(j);
                    String aux=diccionario.get(String.valueOf(a.obtenerNoterminal()
                    if (a.obtenerNoterminal() == alpha.charAt(i) && aux==null) {
                    //Verificamos que no lo hayamos calculado
                    resultado = resultado + calculaPrimero(a.obtenerNoterminal(),
                    a.obtenerAlpha());
                    }else if (aux!=null){
                        resultado = resultado + aux;
                    }
                    j++;
                }
            }
            i++;
        }
        return resultado;
    }
```

Como podemos ver, se siguen los pasas de cada regla segun el algoritmo.

3. Conclusiones

Con esta práctica aprendí cosas curiosas en Java, me estoy acostumbrando al lenguaje y a los beneficios de programar en orientado a objetos, aunque me he dado cuenta que la pienso mucho el empezar a programar, es decir, no es que no me guste, me gusta hacerlo pero cuando pienso todo lo que hay que hacer en ocaciones me da flojera y alargó las cosas, pero ya cuando empiezo se me va el tiempo volando como.

Independiente de que tenía que repasar la teoría del algoritmo para aplicarla al, creo que el retro para mi en estas practicas es aprender todas las facilidades que ofrece Java.

4. Bibliografía y Referencias

- AHO, ALFRED V. COMPILADORES. PRINCIPIOS, TÉCNICAS Y HERRAMIENTAS. Segunda edición PEARS ON EDUCACIÓN, México, 2008