Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en Sistemas Computacionales

Compiladores

Práctica 5: Diseño e implementación de la clase para obtener la tabla LL(1) de una gramática libre de contexto sin recursión izquierda

Profesor: Rafael Norman Saucedo Delgado

Alumno: Ayala Segoviano Donaldo Horacio

Boleta: 2019630415

Introducción

Analizadores LL(1)

Un analizador LL (1) es un algoritmo de descenso recursivo que utiliza un solo token de búsqueda anticipada para construir una derivación gramatical para una secuencia de entrada.

Los analizadores sintácticos LL(1) se componen de dos partes principales: la tabla LL(1) y el análisis sintáctico LL(1), para poder realizar el análisis sintáctico LL(1) se necesita de la tabla LL(1).

Tabla LL(1)

Una tabla de análisis de LL (1) es una estructura de datos que codifica la información anticipada de una gramática. Un analizador LL (1) utiliza una tabla de análisis como un oráculo; consulta la mesa para elegir qué producciones aplicar mientras construye una derivación para una secuencia de tokens o símbolos.

Las filas de una tabla de análisis están etiquetadas con no terminales y sus columnas están etiquetadas con símbolos de anticipación. Sus celdas contienen lados derechos de las producciones. A continuación, se muestra el algoritmo de construcción de una tabla LL(1).

Paso 1: Numerar las producciones.
Paso 2: Recorrer todas las producciones: A -> alfa (#)
Paso 2.1: Por cada símbolo ‘a’ en Primero(alfa)
Paso 2.1.1: Agregar ‘#’ en la tabla para la entrada (A,a)
Paso 2.2: Si épsilon aparece en Primero(alfa)
Paso 2.2.1: Para cada símbolo ‘a’ en Siguiente(a)
Paso 2.2.1.1 Agregar ‘#’ en la tabla para la entrada (A,a)

Análisis Sintáctico LL(1)

El proceso de análisis sintáctico nos permite decidir si una cadena pertenece al lenguaje generado por una gramática. Para hacer esto, se necesita un algoritmo, que no serpa visto en esta práctica, se necesita también la cadena a analizar, y la tabla LL(1).

Características de la implementación de la práctica

En la presente práctica se implementará la clase que generará la tabla LL(1) para una gramática libre de contexto sin recursión izquierda. La clase leerá la gramática desde un archivo de texto, el cual tiene que ser especificado al llamar al método correspondiente, y escribirá la gramática, así como las producciones numeradas, igualmente en un archivo de salida.

Desarrollo

1. Definición del formato del archivo de texto de entrada y salida

El formato del archivo de texto de entrada donde se definirá la gramática a la cual se le generará la tabla LL(1) consiste de:

Primera línea: Consistirá de un solo carácter que denotará el símbolo inicial de la gramática.

Siguientes líneas: cadenas de la forma *S->alfa*, es decir, *S* es un carácter de las mayúsculas denotándola cabeza de la producción, seguido de la cadena "->", y seguido de esta cadena viene *alfa* que es una secuencia de caracteres mayúsculas y minúsculas que determina el cuerpo de la producción. Se pueden escribir tantas producciones como se desee, siempre y cuando la gramática descrita sea una gramática válida.

Es importante destacar que se denotará un solo cuerpo por producción y se especificará una sola producción por línea, es decir, la especificación de varios cuerpos por producción, por ejemplo: *S->a|b* no está permitida.

El alfabeto de entrada consistirá solamente de:

Mayúsculas del alfabeto inglés: denotan los no terminales.

Minúsculas del alfabeto inglés: denotan los terminales.

Carácter '3': Denota a épsilon.

El formato de archivo de salida, donde se escribirá el resultado del programa, es decir, la tabla LL(1) consistirá de lo siguiente:

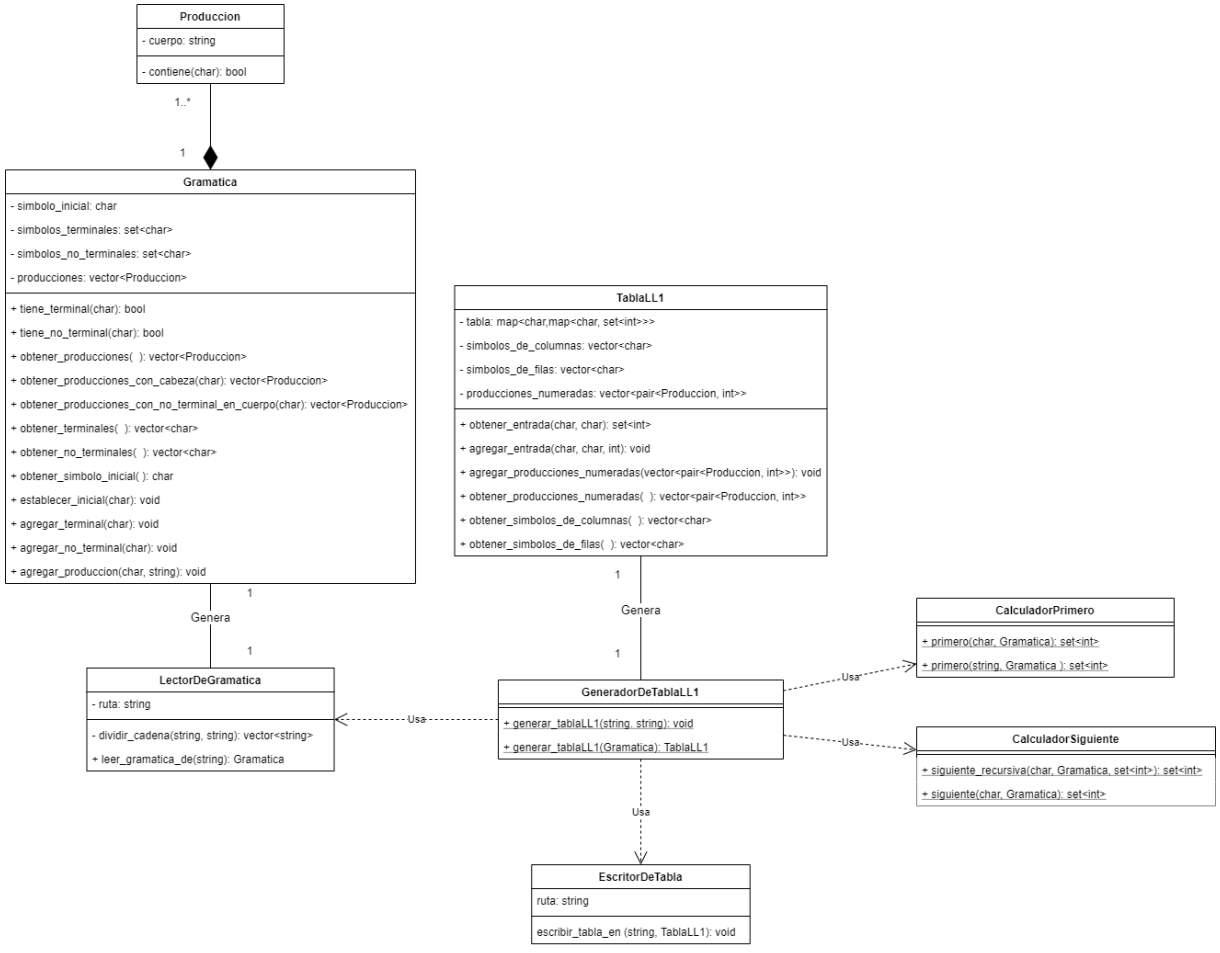
Primeras n líneas: Serán las n producciones seguidas de su correspondiente numeración.

Siguiente línea: Corresponderá a el encabezado de la tabla que son los terminales y el símbolo '$'.

Siguientes x líneas: Serán las filas de la tabla, el primer carácter será el terminal que representa esa fila y estará seguido de las entradas de la tabla en su posición correspondiente.

2. Diseño del diagrama de clases

A continuación se muestra el diagrama de clases correspondiente al generador de la tabla LL(1) acompañada de las clases necesarias. El diagrama va a ser utilizado para la implementación de la práctica.



3. Implementación de las clases

Ahora se procederá a implementar en el lenguaje de programación C++ las clases especificadas en el diagrama de clases. El código de dichas clases será adjuntado a la práctica para su consulta.

4. Pruebas.

Una vez terminada la codificación de las clases, se procederá a hacer pruebas para comprobar que el generador funciona correctamente. Para esto, se usarán algunos ejemplos revisados en clase y así poder asegurarnos que la tabla generada es correcta.

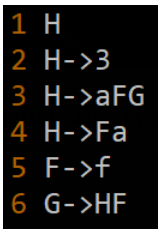


Figura 1. Contenido del archivo de entrada que especifica la gramática.

En la Figura 1 se muestra el contenido del archivo que se entregará a el programa para que genere la tabla LL(1), y ahora, se ejecutará el programa, y se observará la salida.

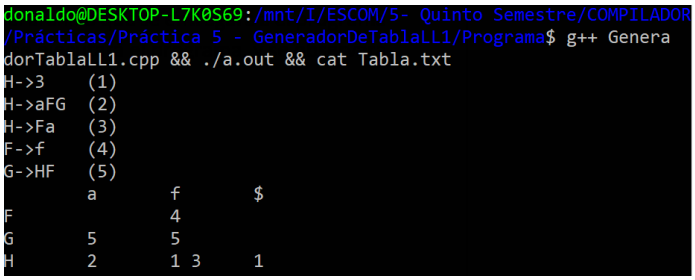


Figura 2. Salida para la gramática especificada en la figura 1.

En la figura 2, se muestra la salida de la ejecución del programa, como se puede observar, la salida es correcta, y a continuación se muestra el proceso de obtención para verificar que el resultado es correcto

.

```
Ejemplo: construir la tabla LL(1)
H->epsilón      (1)   Siguiente(H) = {$, f} ..... Primero(F) = {f}
H->aFG          (2)   Primero(aFG)    = {a}
H->Fa           (3)   Primero (Fa)  = { f }
F->f            (4)   Primero(f) = {f}
G->HF           (5)   Primero(HF) = { a, f }

   a    f    $
H  2   1/3   1          <- entrada doble
F       4
G  5    5
```

Figura 3. Obtención de la tabla LL(1) para la gramática de la figura 1.

Ahora se realizará una segunda prueba, para verificar distintos casos de uso. A continuación, se muestra el archivo de la gramática de entrada.

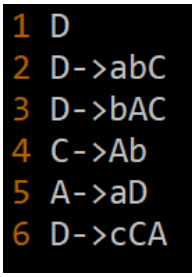


Figura 4. Contenido del archivo de entrada que especifica la gramática.

En la figura 4, se muestra el contenido del archivo de entrada que especifica la segunda gramática a ejecutar. Ahora se ejecutará el programa y se mostrarán los resultados obtenidos.

```
donaldo@DESKTOP-L7K0S69:/mnt/I/ESCOM/5- Quinto Semestre/COMPILADOR
/Prácticas/Práctica 5 - GeneradorDeTablaLL1/Programa$ g++ Genera
dorTablaLL1.cpp && ./a.out && cat Tabla.txt
D->abC  (1)
D->bAC  (2)
C->Ab   (3)
A->aD   (4)
D->cCA  (5)
        a       b       c       $
A       4
C       3
D       1       2       5
```

Figura 5. Salida para la gramática especificada en la figura 4.

Como se puede observar en la figura 5, se muestra el contenido del archivo donde se guarda la información de la salida de la ejecución del programa generador de la tabla LL(1) para la gramática especificada en la figura 4. Ahora, se procederá a comprobar el resultado.

```
D->abC  (1)     alfa = abC  Primero(abC) = { a }
D->bAC  (2)     Primero(bAC) = { b }
C->Ab   (3)     Primero(Ab) = { a } .....Primero(A)  =  { a }
A->aD   (4)     Primero(aD) = { a }
D->cCA  (5)     Primero(cCA) = { c }

  a b c $
A 4
C 3
D 1 2 5
```

Figura 6. Comprobación de la tabla LL(1) para la gramática de la figura 4.

Como se puede observar en la figura 6, el resultado obtenido coincide con el resultado de la ejecución del programa, así, podemos concluir que nuestro programa funciona correctamente.

Conclusiones

Durante la realización de la práctica, se tuvieron dificultades del momento de diseñar el diagrama de clases, ya que algunos conceptos aún no estaban claros. Sin embargo, se pudieron reforzar los conocimientos del algoritmo de generación de la tabla LL(1).

Se pudo concluir, que la realización de esta práctica va a ser útil para la realización de practicas siguientes, ya que se reutilizan los algoritmos de Primero y Siguiente.

Referencias

- Aho A., Lam M. (2008) *Compiladores, principios técnicas y herramientas. Segunda edición*. México: Pearson Education.
- Lasser S., Casinghino C., Fisher K., Roux C. (2019) *A Verified LL(1) Parser Generator.* Tufts University, Medford, MA, USA.