

CASTRO CRUCES JORGE E.	INSTITUTO POLITÉCNICO NACIONAL	ESCUELA SUPERIOR DE CÓMPUTO
EXTRAORDINARIO	PRÁCTICA 6: GRAMATICAS LL(1)	29/06/2021
3CV18	COMPILADORES	Hernández Olvera Luis E

SECCIÓN TEÓRICA O INTRODUCCIÓN

ANÁLISIS SINTÁCTICO DESCENDENTE

El análisis sintáctico descendente puede verse como el problema de construir un árbol de análisis sintáctico para la cadena de entrada, partiendo desde la raíz y creando los nodos del árbol de análisis sintáctico en preorden (profundidad primero, como vimos en la sección 2.3.4). De manera equivalente, podemos considerar el análisis sintáctico descendente como la búsqueda de una derivación por la izquierda para una cadena de entrada.

En cada paso de un análisis sintáctico descendente, el problema clave es el de determinar la producción que debe aplicarse para un no terminal, por decir A. Una vez que se elige una producción A, el resto del proceso de análisis sintáctico consiste en “relacionar” los símbolos terminales en el cuerpo de la producción con la cadena de entrada. Esta sección empieza con una forma general del análisis sintáctico descendente, conocida como análisis sintáctico de descenso recursivo, la cual puede requerir de un rastreo hacia atrás para encontrar la producción A correcta que debe aplicarse. La sección 2.4.2 introdujo el análisis sintáctico predictivo, un caso especial de análisis sintáctico de descenso recursivo, en donde no se requiere un rastreo hacia atrás. El análisis sintáctico predictivo elige la producción A correcta mediante un análisis por adelantado de la entrada, en donde se ve un número fijo de símbolos adelantados; por lo general, sólo necesitamos ver un símbolo por adelantado (es decir, el siguiente símbolo de entrada).[1]

ANÁLISIS SINTÁCTICO DE DESCENSO RECURSIVO

Un programa de análisis sintáctico de descenso recursivo consiste en un conjunto de procedimientos, uno para cada no terminal. La ejecución empieza con el procedimiento para el símbolo inicial, que se detiene y anuncia que tuvo éxito si el cuerpo de su procedimiento explora toda la cadena completa de entrada. En la figura 4.13 aparece el pseudocódigo para un no terminal común. Observe que este pseudocódigo es no determinista, ya que empieza eligiendo la producción A que debe aplicar de una forma no especificada. El descenso recursivo general puede requerir de un rastreo hacia atrás; es decir, tal vez requiera exploraciones repetidas sobre la entrada. Sin embargo, raras veces se necesita el rastreo hacia atrás para analizar las construcciones de un lenguaje de programación, por lo que los analizadores sintácticos con éste no se ven con frecuencia. Incluso para situaciones como el análisis sintáctico de un lenguaje natural, el rastreo hacia atrás no es muy eficiente, por lo cual se prefieren métodos tabulares como el algoritmo de programación dinámica del ejercicio 4.4.9, o el método de Earley (vea las notas bibliográficas). Para permitir el rastreo hacia atrás, hay que modificar el código de la figura 4.13. En primer lugar, no podemos elegir una producción A única en la línea (1), por lo que debemos probar cada una de las diversas producciones en cierto orden. Después, el fallo en la línea (7) no es definitivo, sino que sólo sugiere que necesitamos regresar a la línea (1) y probar otra producción A. Sólo si no hay más producciones A para probar es cuando declaramos que se ha encontrado un error en la entrada. Para poder probar otra producción A, debemos restablecer el apuntador de entrada a la posición en la que se encontraba cuando llegamos por primera vez a la línea (1). Es decir, se requiere una variable local para almacenar este apuntador de entrada, para un uso futuro.[2]

PRIMERO Y SIGUIENTE

La construcción de los analizadores sintácticos descendentes y ascendentes es auxiliada por dos funciones, PRIMERO y SIGUIENTE, asociadas con la gramática G . Durante el análisis sintáctico descendente, PRIMERO y SIGUIENTE nos permiten elegir la producción que vamos a aplicar, con base en el siguiente símbolo de entrada. Durante la recuperación de errores en modo de pánico, los conjuntos de tokens que produce SIGUIENTE pueden usarse como tokens de sincronización. Definimos a $\text{PRIMERO}(\alpha)$, en donde α es cualquier cadena de símbolos gramaticales, como el conjunto de terminales que empiezan las cadenas derivadas a partir de α . Si $\alpha \Rightarrow^* \text{lm}$, entonces también se encuentra en $\text{PRIMERO}(\alpha)$. Por ejemplo, en la figura 4.15, $A \Rightarrow^* \text{lm } c\gamma$, por lo que c está en $\text{PRIMERO}(A)$.

Para una vista previa de cómo usar PRIMERO durante el análisis sintáctico predictivo, considere dos producciones A , $A \rightarrow \alpha \mid \beta$, en donde $\text{PRIMERO}(\alpha)$ y $\text{PRIMERO}(\beta)$ son conjuntos separados. Entonces, podemos elegir una de estas producciones A si analizamos el siguiente símbolo de entrada a , ya que a puede estar a lo más en $\text{PRIMERO}(\alpha)$ o en $\text{PRIMERO}(\beta)$, pero no en ambos. Por ejemplo, si a está en $\text{PRIMERO}(\beta)$, elegimos la producción $A \rightarrow \beta$. Exploraremos esta idea en la sección 4.4.3, cuando definamos las gramáticas LL(1). Definimos a $\text{SIGUIENTE}(A)$, para el no terminal A , como el conjunto de terminales a que pueden aparecer de inmediato a la derecha de A en cierta forma de frase; es decir, el conjunto de terminales A de tal forma que exista una derivación de la forma $S \Rightarrow^* \alpha A a \beta$, para algunas α y β , como en la figura 4.15. Observe que pudieron haber aparecido símbolos entre A y a , en algún momento durante la derivación, pero si es así, derivaron a y desaparecieron. Además, si A puede ser el símbolo por la derecha en cierta forma de frase, entonces $\$$ está en $\text{SIGUIENTE}(A)$; recuerde que $\$$ es un símbolo “delimitador” especial, el cual se supone que no es un símbolo de ninguna gramática. Para calcular $\text{PRIMERO}(X)$ para todos los símbolos gramaticales X , aplicamos las siguientes reglas hasta que no pueden agregarse más terminales o a ningún conjunto PRIMERO.

1. Si X es un terminal, entonces $\text{PRIMERO}(X) = \{X\}$.
2. Si X es un no terminal y $X \rightarrow Y_1 Y_2 \dots Y_k$ es una producción para cierta $k \geq 1$, entonces se coloca a en $\text{PRIMERO}(X)$ si para cierta i , a está en $\text{PRIMERO}(Y_i)$, y está en todas las funciones $\text{PRIMERO}(Y_1)$, ..., $\text{PRIMERO}(Y_{i-1})$; es decir, $Y_1 \dots Y_{i-1} \Rightarrow^*$. Si está en $\text{PRIMERO}(Y_j)$ para todas las $j = 1, 2, \dots, k$, entonces se agrega a $\text{PRIMERO}(X)$. Por ejemplo, todo lo que hay en $\text{PRIMERO}(Y_1)$ se encuentra sin duda en $\text{PRIMERO}(X)$. Si Y_1 no deriva a , entonces no agregamos nada más a $\text{PRIMERO}(X)$, pero si $Y_1 \Rightarrow^*$, entonces agregamos $\text{PRIMERO}(Y_2)$, y así sucesivamente.
3. Si $X \rightarrow$ es una producción, entonces se agrega a $\text{PRIMERO}(X)$. Ahora, podemos calcular PRIMERO para cualquier cadena $X_1 X_2 \dots X_n$ de la siguiente manera. Se agregan a $\text{PRIMERO}(X_1 X_2 \dots X_n)$ todos los símbolos que no sean de $\text{PRIMERO}(X_1)$. También se agregan los símbolos que no sean de $\text{PRIMERO}(X_2)$, si está en $\text{PRIMERO}(X_1)$; los símbolos que no sean de $\text{PRIMERO}(X_3)$, si está en $\text{PRIMERO}(X_1)$ y $\text{PRIMERO}(X_2)$; y así sucesivamente. Por último, se agrega a $\text{PRIMERO}(X_1 X_2 \dots X_n)$ si, para todas las i , se encuentra en $\text{PRIMERO}(X_i)$.

Para calcular $\text{SIGUIENTE}(A)$ para todas las no terminales A , se aplican las siguientes reglas hasta que no pueda agregarse nada a cualquier conjunto SIGUIENTE.

1. Colocar $\$$ en $\text{SIGUIENTE}(S)$, en donde S es el símbolo inicial y $\$$ es el delimitador derecho de la entrada.
2. Si hay una producción $A \rightarrow \alpha B \beta$, entonces todo lo que hay en $\text{PRIMERO}(\beta)$ excepto está en $\text{SIGUIENTE}(B)$.
3. Si hay una producción $A \rightarrow \alpha B$, o una producción $A \rightarrow \alpha B \beta$, en donde $\text{PRIMERO}(\beta)$ contiene a , entonces todo lo que hay en $\text{SIGUIENTE}(A)$ está en $\text{SIGUIENTE}(B)$. [3]

GRAMÁTICAS LL(1)

Los analizadores sintácticos predictivos, es decir, los analizadores sintácticos de descenso recursivo que no necesitan rastreo hacia atrás pueden construirse para una clase de gramáticas llamadas LL(1). La primera “L” en LL(1) es para explorar la entrada de izquierda a derecha (por left en inglés), la segunda “L” para producir una derivación por la izquierda, y el “1” para usar un símbolo de entrada de anticipación en cada paso, para tomar las decisiones de acción del análisis sintáctico. [4]

DIAGRAMAS DE TRANSICIÓN PARA ANALIZADORES SINTÁCTICOS PREDICTIVOS

Los diagramas de transición son útiles para visualizar los analizadores sintácticos predictivos. Por ejemplo, los diagramas de transición para las no terminales E y E de la gramática (4.28) aparecen en la figura 4.16(a). Para construir el diagrama de transición a partir de una gramática, primero hay que eliminar la recursividad por la izquierda y después factorizar la gramática por la izquierda. Entonces, para cada no terminal A,

1. Se crea un estado inicial y un estado final (retorno).
2. Para cada producción $A \rightarrow X_1X_2 \dots X_k$, se crea una ruta desde el estado inicial hasta el estado final, con los flancos etiquetados como X_1, X_2, \dots, X_k . Si $A \rightarrow \epsilon$, la ruta es una línea que se etiqueta como ϵ .

Los diagramas de transición para los analizadores sintácticos predictivos difieren de los diagramas para los analizadores léxicos. Los analizadores sintácticos tienen un diagrama para cada no terminal. Las etiquetas de las líneas pueden ser tokens o no terminales. Una transición sobre un token (terminal) significa que tomamos esa transición si ese token es el siguiente símbolo de entrada. Una transición sobre un no terminal A es una llamada al procedimiento para A. Con una gramática LL(1), la ambigüedad acerca de si se debe tomar o no una línea puede resolverse si hacemos que las transiciones sean la opción predeterminada. Los diagramas de transición pueden simplificarse, siempre y cuando se preserve la secuencia de símbolos gramaticales a lo largo de las rutas. También podemos sustituir el diagrama para un no terminal A, en vez de una línea etiquetada como A. Los diagramas en las figuras 4.16(a) y (b) son equivalentes: si trazamos rutas de E a un estado de aceptación y lo sustituimos por E, entonces, en ambos conjuntos de diagramas, los símbolos gramaticales a lo largo de las rutas forman cadenas del tipo $T + T + \dots + T$. El diagrama en (b) puede obtenerse a partir de (a) mediante transformaciones semejantes a las de la sección 2.5.4, en donde utilizamos la eliminación de recursividad de la parte final y la sustitución de los cuerpos de los procedimientos para optimizar el procedimiento en un no terminal.[5]

PROBLEMAS RELACIONADOS AL TEMA A LOS CUALES TE ENFRENTASTE AL PROGRAMAR LA PRÁCTICA

No se por donde empezar, ya que esta práctica ha sido la más complicada que he tenido que llevar a cabo en mi vida.

Primeramente, los problemas empezaron desde el momento en que tuve que estudiar y comprender el tema de Gramáticas LL(1), hasta la parte de la implementación en un lenguaje bastante robusto para este tipo de programas, como lo es Java.

Otro gran reto al que me enfrenté fue recordar mis clases de Estructuras de Datos y de Programación Orientada a Objetos, ya que me fueron de gran utilidad para construir las estructuras de manejo de la información, en este caso:

- Pilas
- Colas
- Arreglos
- Tablas Hash

Para empezar, las pilas me fueron de gran ayuda a la hora de ingresar y sacar datos de la secuencia de movimientos equivalentes a la derivación por la derecha, porque con el simple hecho de invocar al método pop() o push(), es que pude manejar a placer la pila.

Como si todo lo anterior no fuese suficiente, los algoritmos PRIMERO y SIGUIENTE, me causaron un estrés tremendo, al no poder comprender a priori el funcionamiento de estos, pero una vez que logré construirlos, fue muy sencillo construir la Tabla de Análisis Sintáctico Predictivo.

Considero que otro gran reto al que me enfrenté fue saber que mi programa estaba funcionando correctamente, porque a pesar de que me apoyé en usar las gramáticas que el profesor usó de ejemplo en los videos explicativos de la clase, tuve que realizar el cálculo, tanto de los PRIMEROS como de los SIGUIENTES, para corroborar que mis resultados eran correctos.

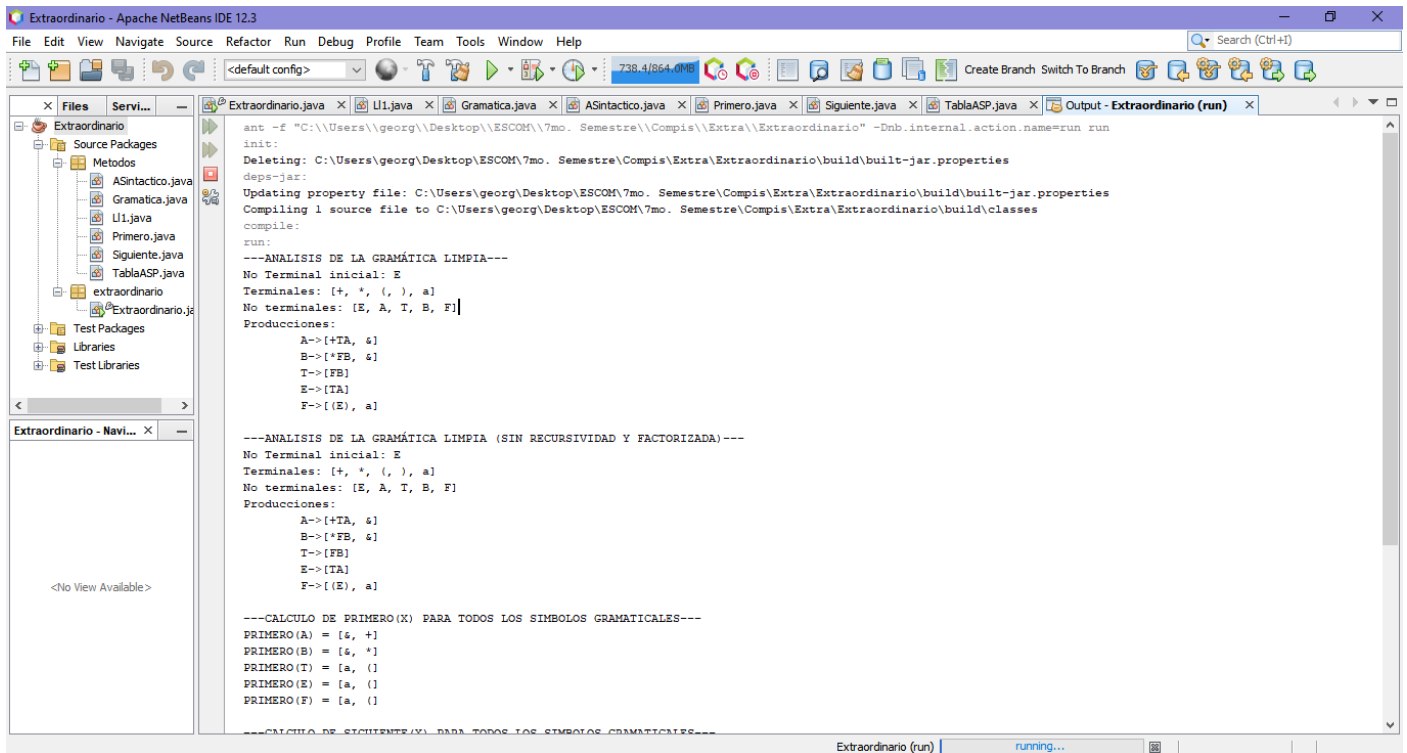
Considerando el tiempo entre que se dejó la práctica y la entrega de la misma, fue muy corto el tiempo para elaborar un programa con gran detalle, por lo cual es que envió mi examen Extraordinario, con los requerimientos solicitados por el profesor, que son:

- Análisis de la Gramática Limpia
- Análisis de la Gramática Limpia (Quitando la Recursividad y Factorizando las producciones)
- Cálculo de los PRIMEROS de cada no Terminal
- Cálculo de los SIGUIENTES de cada no Terminal
- Cálculo de la Tabla de Análisis Sintáctico Predictivo
- Comprobación de la pertenencia de una cadena al lenguaje formado por la Gramática
- Impresión de la secuencia de movimientos (Equivalente a la derivación por la izquierda)

CAPTURAS DE PANTALLA

A continuación se muestran las capturas de pantalla de la ejecución del programa, para la Gramática vista en clase:

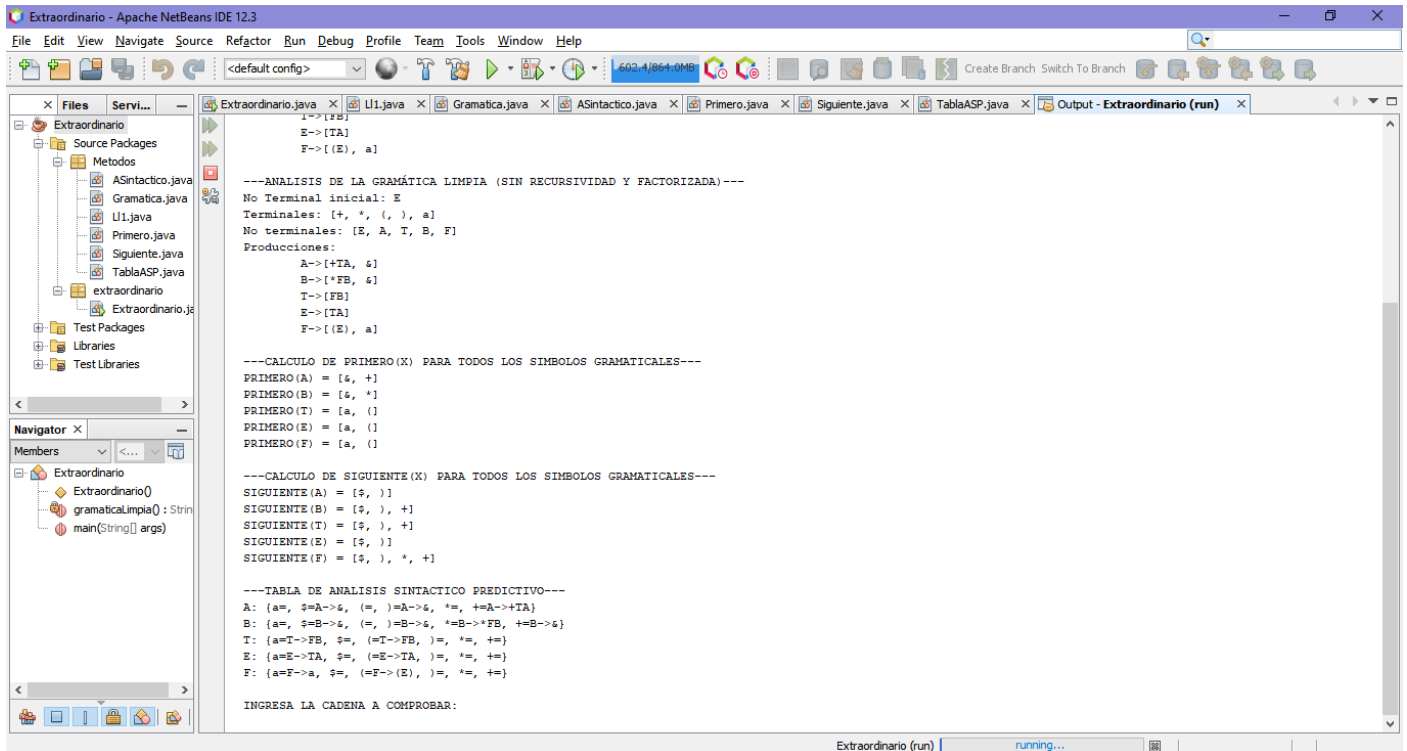
$$\begin{aligned} E &\rightarrow T E' \\ E' &\rightarrow + T E' \mid \epsilon \\ T &\rightarrow F T' \\ T' &\rightarrow * F T' \mid \epsilon \\ F &\rightarrow (E) \mid id \end{aligned}$$



```
ant -f "C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario" -Dnb.internal.action.name=run run
init:
Deleting: C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario\build\build-jar.properties
deps-jar:
Updating property file: C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario\build\build-jar.properties
Compiling 1 source file to C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario\build\classes
compile:
run:
---ANALISIS DE LA GRAMÁTICA LIMPIA---
No Terminal inicial: E
Terminales: [+ , ( , ) , a]
No terminales: [E, A, T, B, F]
Producciones:
A->[+TA, &]
B->[*FB, &]
T->[FB]
E->[TA]
F->[(E), a]

---ANALISIS DE LA GRAMÁTICA LIMPIA (SIN RECURSIVIDAD Y FACTORIZADA)---
No Terminal inicial: E
Terminales: [+ , ( , ) , a]
No terminales: [E, A, T, B, F]
Producciones:
A->[+TA, &]
B->[*FB, &]
T->[FB]
E->[TA]
F->[(E), a]

---CALCULO DE PRIMERO(X) PARA TODOS LOS SIMBOLOS GRAMATICALES---
PRIMERO(A) = {+, &}
PRIMERO(B) = {+, &}
PRIMERO(T) = {+, &}
PRIMERO(E) = {+, &}
PRIMERO(F) = {+, &}
```



```
1->[FB]
E->[TA]
F->[(E), a]

---ANALISIS DE LA GRAMÁTICA LIMPIA (SIN RECURSIVIDAD Y FACTORIZADA)---
No Terminal inicial: E
Terminales: [+ , ( , ) , a]
No terminales: [E, A, T, B, F]
Producciones:
A->[+TA, &]
B->[*FB, &]
T->[FB]
E->[TA]
F->[(E), a]

---CALCULO DE PRIMERO(X) PARA TODOS LOS SIMBOLOS GRAMATICALES---
PRIMERO(A) = {+, &}
PRIMERO(B) = {+, &}
PRIMERO(T) = {+, &}
PRIMERO(E) = {+, &}
PRIMERO(F) = {+, &}

---CALCULO DE SIGUIENTE(X) PARA TODOS LOS SIMBOLOS GRAMATICALES---
SIGUIENTE(A) = {+, &}
SIGUIENTE(B) = {+, &}
SIGUIENTE(T) = {+, &}
SIGUIENTE(E) = {+, &}
SIGUIENTE(F) = {+, &}

---TABLA DE ANALISIS SINTACTICO PREDICTIVO---
A: {a=, &=A->&, (=, )=A->&, *=, +=A->+TA}
B: {a=, &=B->&, (=, )=B->&, *=B->*FB, +=B->&}
T: {a=T->FB, &=, (=T->FB, )=, *=, +=}
E: {a=E->TA, &=, (=E->TA, )=, *=, +=}
F: {a=F->a, &=, (=F->(E), )=, *=, +=}

INGRESA LA CADENA A COMPROBAR:
```

Ahora, se va a comprobar la pertenencia de dos cadenas a la Gramática ya mencionada:

The screenshot shows the Apache NetBeans IDE with the 'Extraordinario' project. The 'Output - Extraordinario (run)' window displays the following text:

```
---TABLA DE ANALISIS SINACTICO PREDICTIVO---
A: {a=, $=A->$, (, )=A->$, *=, +=A->+TA}
B: {a=, $=B->$, (, )=B->$, *=B->*FB, +=B->+}
T: {a=T->FB, $=, (=T->FB, )=, *=, +=}
E: {a=E->TA, $=, (=E->TA, )=, *=, +=}
F: {a=F->a, $=, (=F->(E), )=, *=, +=}

INGRESA LA CADENA A COMPROBAR:
(a+a*a)

COINCIDENCIA      PILA      ENTRADA      ACCION
(a+a*a)           $E      (a+a*a)$      EMITIR E->TA
(a+a*a)           $AT      (a+a*a)$      EMITIR T->FB
(a+a*a)           $ABF      (a+a*a)$      EMITIR F->(E)
(a+a*a)           $AB)E(      (a+a*a)$      RELACIONAR (
(a+a*a)           $AB)E      a+a*a)$      EMITIR E->TA
(a+a*a)           $AB)AT      a+a*a)$      EMITIR T->FB
(a+a*a)           $AB)ABF      a+a*a)$      EMITIR F->a
(a+a*a)           $AB)ABa      a+a*a)$      RELACIONAR a
(a+a*a)           $AB)AB      +a*a)$      EMITIR B->+
(a+a*a)           $AB)A      +a*a)$      EMITIR A->+TA
(a+a)             $AB)AT+      +a*a)$      RELACIONAR +
(a+)              $AB)AT      a*a)$      EMITIR T->FB
(a+)              $AB)ABF      a*a)$      EMITIR F->a
(a+a)             $AB)ABa      a*a)$      RELACIONAR a
(a+a)             $AB)AB      *a)$      EMITIR B->*FB
(a+a)             $AB)ABF*      *a)$      RELACIONAR *
(a+a)             $AB)ABF      a)$      EMITIR F->a
(a+a*a)           $AB)ABa      a)$      RELACIONAR a
(a+a*a)           $AB)AB      )$      EMITIR B->+
(a+a*a)           $AB)A      )$      EMITIR A->+
(a+a*a)           $AB)      )$      RELACIONAR )
(a+a*a)           $AB      $      EMITIR B->+
(a+a*a)           $A      $      EMITIR A->+
(a+a*a)$          $      $      RELACIONAR $

CADENA VERIFICADA Y ACEPTADA PARA LA GRAMATICA...

INGRESA LA CADENA A COMPROBAR:
```

Efectivamente, la cadena: **(a+a*a)** si pertenece a la gramática.

Extraordinario - Apache NetBeans IDE 12.3

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config> 518.3/864.0MB Create Branch Switch To Branch

Extraordinario.java U1.java Gramatica.java ASintactico.java Primero.java Siguiente.java TablaASP.java Output - Extraordinario (run)

Extraordinario

Source Packages

Metodos

ASintactico.java

Gramatica.java

U1.java

Primero.java

Siguiente.java

TablaASP.java

extraordinario

Extraordinario.java

Test Packages

Libraries

Test Libraries

Navigator

Members

Extraordinario

Extraordinario()

gramaticaLimpia() : String

main(String[] args)

COINCIDENCIA	PILA	ENTRADA	ACCION
(\$E	(a+a*)\$	EMITIR E->TA
\$AT	(a+a*)\$	(a+a*)\$	EMITIR T->FB
\$ABF	(a+a*)\$	(a+a*)\$	EMITIR F->(E)
\$ABE	(a+a*)\$	(a+a*)\$	RELACIONAR (
\$ABE	(a+a*)\$	(a+a*)\$	EMITIR E->TA
\$ABAT	(a+a*)\$	(a+a*)\$	EMITIR T->FB
\$ABABF	(a+a*)\$	(a+a*)\$	EMITIR F->a
(a	\$AB)ABa	(a+a*)\$	RELACIONAR a
(a	\$AB)AB	(a+a*)\$	EMITIR B->\$
(a	\$AB)A	(a+a*)\$	EMITIR A->+TA
(a+	\$AB)A+	(a+a*)\$	RELACIONAR +
(a+	\$AB)AT	(a+a*)\$	EMITIR T->FB
(a+	\$AB)ABF	(a+a*)\$	EMITIR F->a
(a+	\$AB)ABa	(a+a*)\$	RELACIONAR a
(a+	\$AB)AB	(a+a*)\$	EMITIR B->*FB
(a+a*	\$AB)ABF*	(a+a*)\$	RELACIONAR *
(a+a*	\$AB)ABF	(a+a*)\$	EMITIR F->a
(a+a*	\$AB)ABa	(a+a*)\$	RELACIONAR a
(a+a*	\$AB)AB	(a+a*)\$	EMITIR B->\$
(a+a*	\$AB)A	(a+a*)\$	EMITIR A->\$
(a+a*)	\$AB)	(a+a*)\$	RELACIONAR)
(a+a*)	\$AB	(a+a*)\$	EMITIR B->\$
(a+a*)	\$A	(a+a*)\$	EMITIR A->\$
(a+a*)\$	\$	(a+a*)\$	RELACIONAR \$

CADENA VERIFICADA Y ACEPTADA PARA LA GRAMATICA...

INGRESA LA CADENA A COMPROBAR:

aaa+a*aaa

COINCIDENCIA PILA ENTRADA ACCION

\$E aaa+a*aaa\$ EMITIR E->TA

\$AT aaa+a*aaa\$ EMITIR T->FB

\$ABF aaa+a*aaa\$ EMITIR F->a

\$ABa aaa+a*aaa\$ RELACIONAR a

\$AB aaa+a*aaa\$ EMITIR

\$AB aaa+a*aaa\$ ERROR

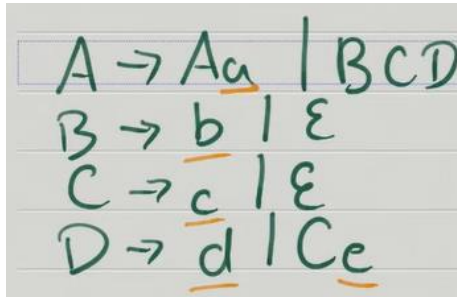
INGRESA LA CADENA A COMPROBAR:

Extraordinario (run) running...

17°C Chubascos 06:58 p. m. 28/06/2021

Efectivamente, la cadena: **aaa+a*aaa** no pertenece a la gramática, por lo que manda error.

A continuación se muestran las capturas de pantalla de la ejecución del programa, para la segunda Gramática vista en clase:



```

ant -f "C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario\build\build-jar.properties"
init:
Deleting: C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario\build\build-jar.properties
deps-jar:
Updating property file: C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario\build\build-jar.properties
Compiling 1 source file to C:\Users\georg\Desktop\ESCOM\7mo. Semestre\Compis\Extra\Extraordinario\build\classes
compile:
run:
---ANALISIS DE LA GRAMÁTICA LIMPIA---
No Terminal inicial: A
Terminales: [a, b, c, d, e]
No terminales: [A, E, B, C, D]
Producciones:
A->[Aa, BCD]
B->[b, ε]
C->[c, ε]
D->[d, Ce]

---ANALISIS DE LA GRAMÁTICA LIMPIA (SIN RECURSIVIDAD Y FACTORIZADA)---
No Terminal inicial: A
Terminales: [a, b, c, d, e]
No terminales: [A, E, B, C, D]
Producciones:
A->[BCDE]
B->[b, ε]
C->[c, ε]
D->[d, Ce]
E->[aE, ε]

---CALCULO DE PRIMERO(X) PARA TODOS LOS SIMBOLOS GRAMATICALES---
PRIMERO(A) = [a, b, c, d, e, ε]
PRIMERO(B) = [b, ε]
PRIMERO(C) = [c, ε]
PRIMERO(D) = [c, d, e]
PRIMERO(E) = [a, ε]

---CALCULO DE SIGUIENTE(X) PARA TODOS LOS SIMBOLOS GRAMATICALES---
SIGUIENTE(A) = [ε]

```

```

B->[b, ε]
C->[c, ε]
D->[d, Ce]

---ANALISIS DE LA GRAMÁTICA LIMPIA (SIN RECURSIVIDAD Y FACTORIZADA)---
No Terminal inicial: A
Terminales: [a, b, c, d, e]
No terminales: [A, E, B, C, D]
Producciones:
A->[BCDE]
B->[b, ε]
C->[c, ε]
D->[d, Ce]
E->[aE, ε]

---CALCULO DE PRIMERO(X) PARA TODOS LOS SIMBOLOS GRAMATICALES---
PRIMERO(A) = [a, b, c, d, e, ε]
PRIMERO(B) = [b, ε]
PRIMERO(C) = [c, ε]
PRIMERO(D) = [c, d, e]
PRIMERO(E) = [a, ε]

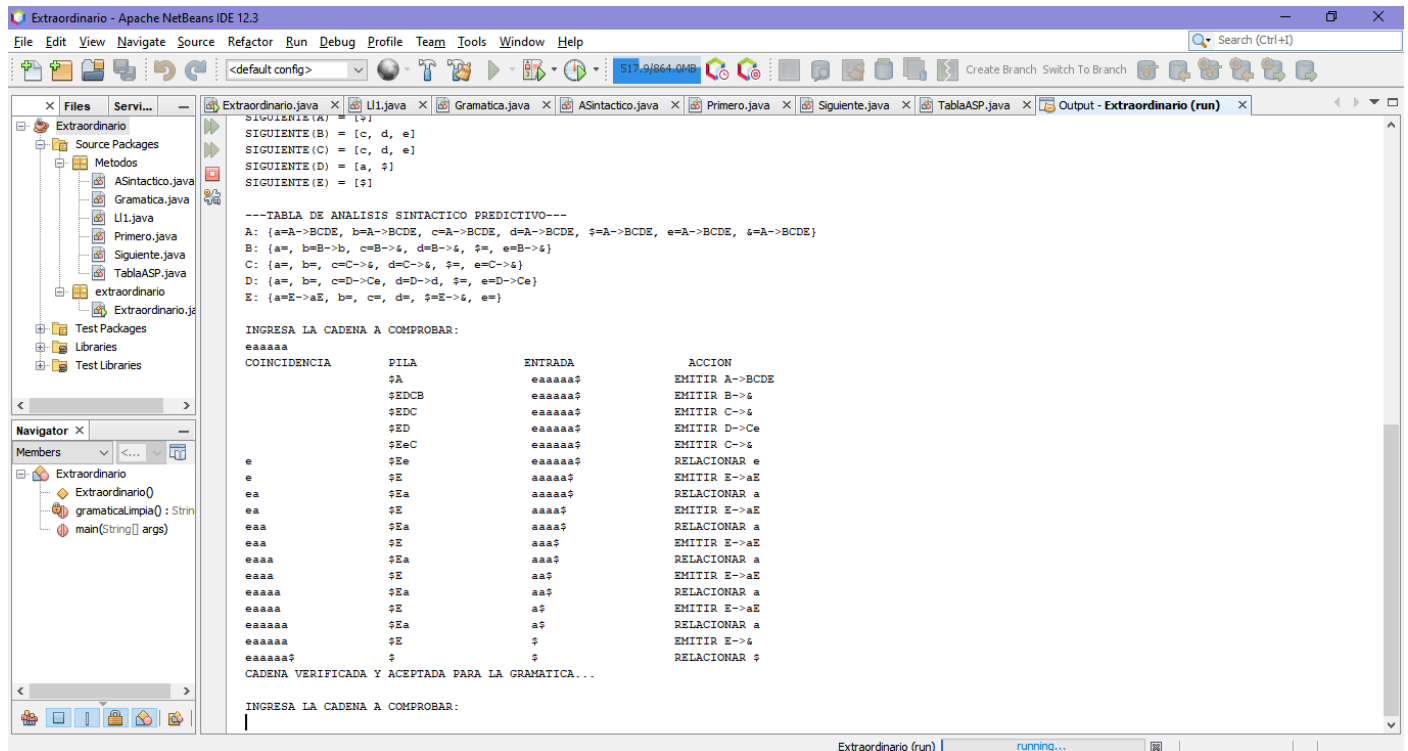
---CALCULO DE SIGUIENTE(X) PARA TODOS LOS SIMBOLOS GRAMATICALES---
SIGUIENTE(A) = [ε]
SIGUIENTE(B) = [c, d, e]
SIGUIENTE(C) = [c, d, e]
SIGUIENTE(D) = [a, ε]
SIGUIENTE(E) = [ε]

---TABLA DE ANALISIS SINTACTICO PREDICTIVO---
A: {a=A->BCDE, b=A->BCDE, c=A->BCDE, d=A->BCDE, $=A->BCDE, e=A->BCDE, ε=A->BCDE}
B: {a=, b=B->b, c=B->ε, d=B->ε, $=, e=B->ε}
C: {a=, b=, c=C->c, d=C->ε, $=, e=C->ε}
D: {a=, b=, c=D->Ce, d=D->d, $=, e=D->Ce}
E: {a=E->aE, b=, c=, d=, $=E->ε, e=}

INGRESA LA CADENA A COMPROBAR:

```


Ahora, se va a comprobar la pertenencia de dos cadenas a la Gramática ya mencionada:



```
Extraordinario.java | U1.java | Gramatica.java | ASintactico.java | Primero.java | Siguiente.java | TablaASP.java | Output - Extraordinario (run) |
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config> 517.9/864.0MB Create Branch Switch To Branch
Extraordinario
  Source Packages
  Metodos
  ASintactico.java
  Gramatica.java
  U1.java
  Primero.java
  Siguiente.java
  TablaASP.java
  extraordinario.java
Test Packages
Libraries
Test Libraries
Members
Extraordinario
  Extraordinario()
  gramaticaLimpia() : String
  main(String[] args)
Extraordinario.java
SIGUIENTE(A) = [a]
SIGUIENTE(B) = [c, d, e]
SIGUIENTE(C) = [c, d, e]
SIGUIENTE(D) = [a, $]
SIGUIENTE(E) = [$]

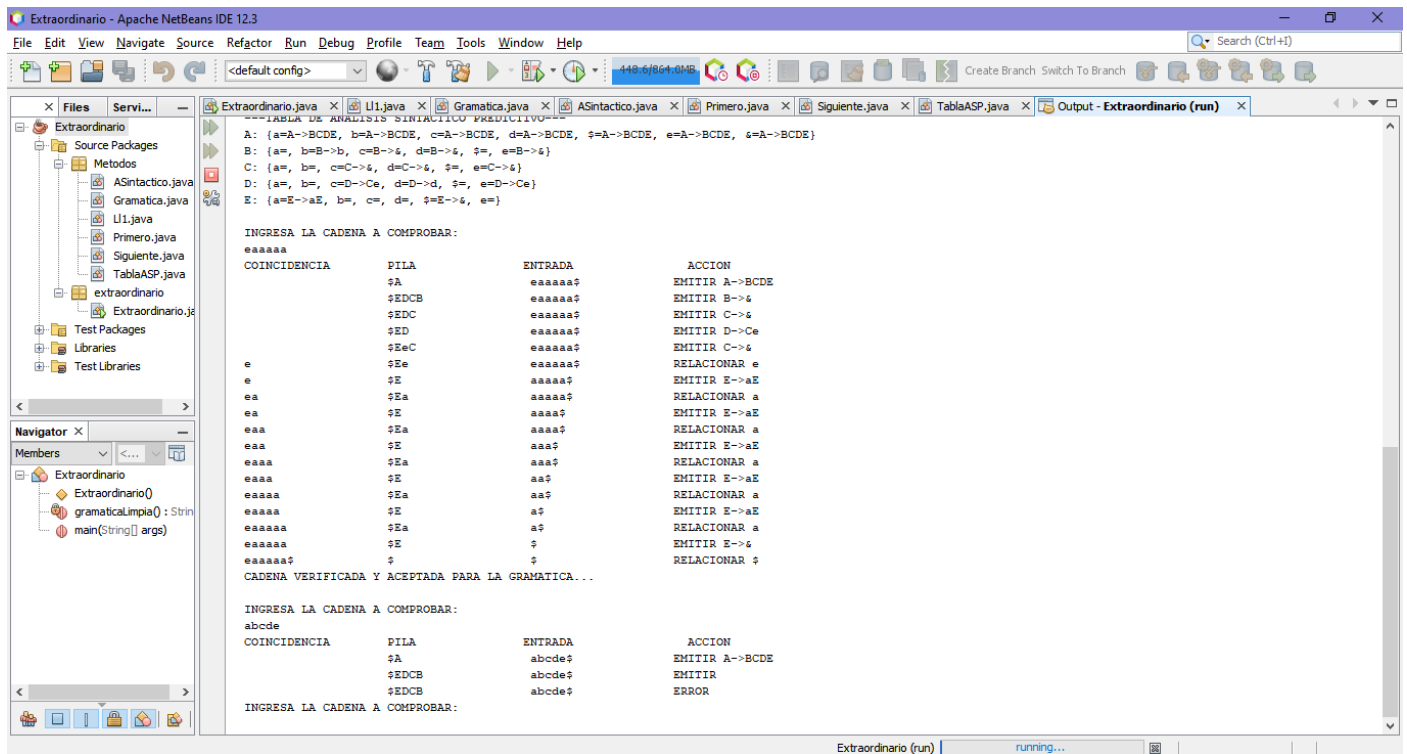
---TABLA DE ANALISIS SINTACTICO PREDICTIVO---
A: {a=A->BCDE, b=A->BCDE, c=A->BCDE, d=A->BCDE, $=A->BCDE, e=A->BCDE, $=A->BCDE}
B: {a=, b=B->b, c=B->$, d=B->$, $=, e=B->$}
C: {a=, b=, c=C->$, d=C->$, $=, e=C->$}
D: {a=, b=, c=D->Ce, d=D->d, $=, e=D->Ce}
E: {a=E->aE, b=, c=, d=, $=E->$, e=}

INGRESA LA CADENA A COMPROBAR:
eaaaaa
COINCIDENCIA      PILA      ENTRADA      ACCION
eaaaaa             $A             eaaaaa$      EMITIR A->BCDE
                  $EDCB             eaaaaa$      EMITIR B->$
                  $EDC             eaaaaa$      EMITIR C->$
                  $ED             eaaaaa$      EMITIR D->Ce
                  $EeC             eaaaaa$      EMITIR C->$
                  $Ee             eaaaaa$      RELACIONAR e
e                  $E             aaaaaa$      EMITIR E->aE
ea                 $Ea             aaaaaa$      RELACIONAR a
ea                 $E             aaaaaa$      EMITIR E->aE
aaa                $Ea             aaaaaa$      RELACIONAR a
aaa                $E             aaaaaa$      EMITIR E->aE
aaaa               $Ea             aaaaaa$      RELACIONAR a
aaaa               $E             aaaaaa$      EMITIR E->aE
aaaaa              $Ea             aaaaaa$      RELACIONAR a
aaaaa              $E             aaaaaa$      EMITIR E->aE
aaaaaa             $Ea             aaaaaa$      RELACIONAR a
aaaaaa             $E             aaaaaa$      EMITIR E->aE
aaaaaa             $Ea             aaaaaa$      RELACIONAR a
aaaaaa             $E             aaaaaa$      EMITIR E->$
aaaaaa$            $             aaaaaa$      RELACIONAR $

CADENA VERIFICADA Y ACEPTADA PARA LA GRAMATICA...

INGRESA LA CADENA A COMPROBAR:
|
```

Efectivamente, la cadena **eaaaaa** si pertenece al lenguaje de la Gramática.



Claramente, podemos ver que la cadena **abcde** no pertenece a la Gramática, por lo que arroja un error.

CONCLUSIÓN

Como comentario extra, quiero decir que los lenguajes que forman cada una de las gramáticas, son los siguientes:

1. Primera gramática: Es capaz de reconocer sumas y multiplicaciones que se encuentran encerradas por paréntesis y a pesar de que estén anidados.
2. Segunda gramática: Es capaz de reconocer únicamente cadenas que empiecen ya sea con **e** o con **d** y con una cantidad infinita de **a**'s a su derecha.

¿Qué aprendí de esta práctica?

Claramente, aprendí muchas cosas, entre ellas está:

- Aprender a programar de forma orientada a objetos.
- Comprender de mejor forma las gramáticas.
- Aprender la diferencia entre un terminal y un no terminal.
- Recordar Estructuras de Datos que vi en materias pasadas.
- Aprendí el funcionamiento de los Analizadores Sintácticos Predictivos.
- Comprendí el funcionamiento de las gramáticas LL(1), tanto sus ventajas, como las desventajas que conlleva.

¿En qué me ayudó esta práctica?

Principalmente, me ayudó a comprender de mejor forma el funcionamiento de un Analizador Sintáctico y por supuesto me ayudó a recordar algunos temas que aprendí en materias pasadas.

Pero como punto más importante, espero que comprender estos temas me ayude a acreditar la materia, ya que espero poder terminar mi carrera en tiempo y forma, para o tener que arrastrar materias.

REFERENCIAS

[1, 2, 3, 4, 5] Aho, A., 2011. *Compiladores*. Pearson Educación de México, SA de CV.