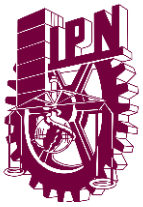


Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en Sistemas Computacionales

Compiladores

Práctica 6: Implementación de la clase que para obtener las tablas LR0, LR1, LALR.

Profesor: Rafael Norman Saucedo Delgado

Alumno: Ayala Segoviano Donaldo Horacio

Boleta: 2019630415

Introducción

Analizadores LR

El tipo más frecuente de analizador sintáctico ascendente en la actualidad se basa en un concepto conocido como análisis sintáctico LR(k); la "L" indica la exploración de izquierda a derecha de la entrada, la "R" indica la construcción de una derivación por la derecha a la inversa, y la k para el número de símbolos de entrada de preanálisis que se utilizan al hacer decisiones del análisis sintáctico.

Los analizadores sin tácticos LR son controlados por tablas, en forma muy parecida a los analizadores sintácticos LL no recursivos.

Un analizador sintáctico LR realiza las decisiones de desplazamiento-reducción mediante el mantenimiento de estados, para llevar el registro de la ubicación que tenemos en un análisis sintáctico. Los estados representan conjuntos de "elementos".

Algoritmo de análisis sintáctico LR

En la figura 3 se puede observar el diagrama de un analizador sintáctico LR. Consiste en una entrada, una salida, una pila, un programa controlador y una tabla de análisis sintáctico que tiene dos partes (ACCION y el ir _ A). El programa controlador es igual para todos los analizadores sintácticos LR; sólo la tabla de análisis sintáctico cambia de un analizador sintáctico a otro. El programa de análisis sintáctico lee caracteres de un búfer de entrada, uno a la vez. En donde un analizador sintáctico de desplazamiento-reducción desplazaría a un símbolo, un analizador sintáctico LR desplaza a un estado.

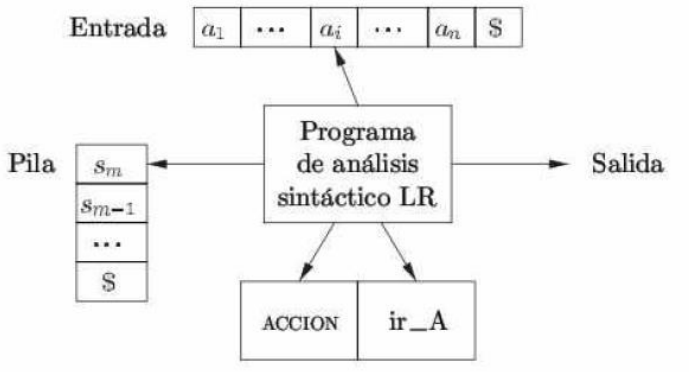


Figura 1. Modelo de un analizador sintáctico LR.

Algoritmo de análisis sintáctico LR

A continuación, se muestra el algoritmo de análisis sintáctico LR. Este algoritmo será el mismo para los diferentes analizadores sintácticos, sin importar con que tabla se esté trabajando.

ENTRADA: Una cadena de entrada w y una tabla de análisis sintáctico LR con las funciones ACCION e ir_A, para una gramática G.

SALIDA: Si w está en $L(G)$, los pasos de reducción de un análisis sintáctico ascendentes para w; en cualquier otro caso, una indicación de error.

MÉTODO: Al principio, el analizador sintáctico tiene s_0 en su pila, en donde s_0 es el estado inicial y $w\$$ está en el búfer de entrada. Entonces, el analizador ejecuta el programa en la figura 4.36. □

```
hacer que a sea el primer símbolo de w$;
while(1) { /* repetir indefinidamente */
    hacer que s sea el estado en la parte superior de la pila;
    if ( ACCION[s, a] = desplazar t ) {
        meter t en la pila;
        hacer que a sea el siguiente símbolo de entrada;
    } else if ( ACCION[s, a] = reducir A → β ) {
        sacar |β| símbolos de la pila;
        hacer que el estado t ahora esté en la parte superior de la pila;
        meter ir_A[t, A] en la pila;
        enviar de salida la producción A → β;
    } else if ( ACCION[s, a] = aceptar ) break; /* terminó el análisis sintáctico */
    else llamar a la rutina de recuperación de errores;
}
```

Figura 2. Algoritmo de análisis sintáctico LR.

Tabla LR

El algoritmo de análisis sintáctico LR es controlado por una tabla, conocida como tabla LR (esta puede ser LR(0), LR(1) o LALR). La tabla de análisis sintáctico consiste en dos partes: una función de análisis sintáctico llamada *acción*, y una función ir_A.

1. La función ACCION recibe como argumentos un estado *i* y un terminal *a* (o $, el marcador de fin de entrada). El valor de ACCION [i, a] puede tener una de cuatro formas:

(a) Desplazar j, en donde j es un estado. La acción realizada por el analizador sintáctico desplaza en forma efectiva la entrada *a* hacia la pila, pero usa el estado j para representar la *a*.

(b) Reducir A->β. La acción del analizador reduce en forma efectiva a β en la parte superior de la pila, al encabezado A.

(c) Aceptar. El analizador sintáctico acepta la entrada y termina el análisis sintáctico.

(d) Error. El analizador sintáctico descubre un error en su entrada y realiza cierta acción correctiva.

2. Extendemos la función ir _ A , definida en los conjuntos de elementos, a los estados: s i ir_A $[I_i, A] = I_j$, entonces ir_A también asigna un estado *i* y un no terminal A al estado j.

Durante esta práctica se implementarán las tablas para los analizadores sintácticos LR(0), LR(1) y LALR.

Algoritmo de subconjuntos LR0

Un elemento LR(0) de una gramática G es una producción de G con un punto en cierta posición del cuerpo.

Una colección de conjuntos de elementos LR(0), conocida como la colección LR(0) canónica, proporciona la base para construir un autómata finito determinista, el cual se utiliza para realizar decisiones en el análisis sintáctico.

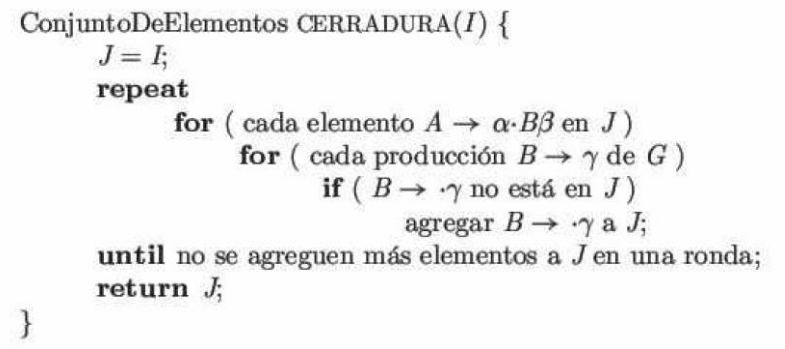
Para construir la colección LR(0) canónica de una gramática, definimos una gramática aumentada y dos funciones, CERRADURA e ir_A . Si G e s una gramática con el símbolo inicial S, entonces G' la **gramática aumentada** para G, es G con un nuevo símbolo inicial S' y la producción S' -> S.

Cerradura

Si *I* es un conjunto de elementos para una gramática G, entonces CERRADURA(*I*) es el conjunto de elementos que se construyen a partir de *I* mediante las siguientes dos reglas:

1. A l principio, agregar cada elemento en *I* a CERRADURA(I).

2. Si A->α.Bβ está en CERRADURA(*I*) y B -> γ es una producción, entonces agregar el elemento B -> γ a CERRADURA(J), si no se encuentra ya ahí. Aplicar esta regla hasta que no puedan agregarse más elementos nuevos a CERRADURA (*I*).

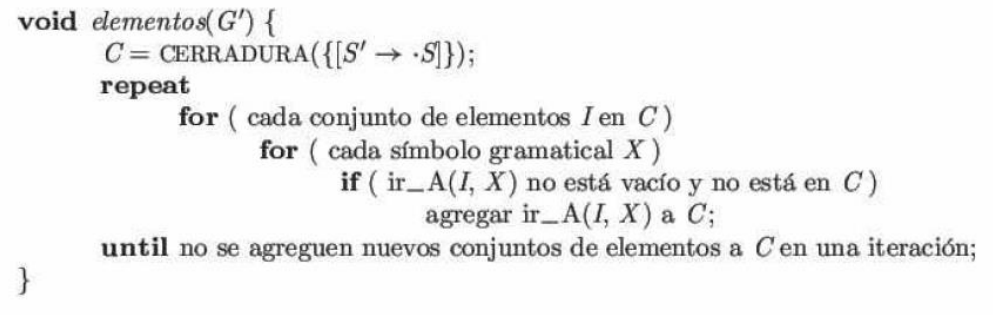


```
ConjuntoDeElementos CERRADURA(I) {
        J = I;
        repeat
                for ( cada elemento A → α·Bβ en J )
                        for ( cada producción B → γ de G )
                                if ( B → ·γ no está en J )
                                        agregar B → ·γ a J;
        until no se agreguen más elementos a J en una ronda;
        return J;
}
```

Figura 3. Pseudocódigo para el conjunto Cerradura.

Ir_A

La segunda función útil es ir_A (*I*, X), en donde *I* es un conjunto de elementos y X es un símbolo gramatical. ir _ A (*I*, X) se define como la cerradura del conjunto de todos los elementos [A -> αX**.**β], de tal forma que [A - > α**.**Xβ] se encuentre en *I*. La función ir _ A se utiliza para definir las transiciones en el autómata LR(0) para una gramática. Los estados del autómata corresponden a los conjuntos de elementos, y ir_A (J, X) especifica la transición que proviene del estado para *I*, con la entrada X.



```
void elementos(G') {
        C = CERRADURA({[S' → ·S]});
        repeat
                for ( cada conjunto de elementos I en C )
                        for ( cada símbolo gramatical X )
                                if ( ir_A(I, X) no está vacío y no está en C )
                                        agregar ir_A(I, X) a C;
        until no se agreguen nuevos conjuntos de elementos a C en una iteración;
}
```

Figura 4. Cálculo de la colección canónica de conjuntos de elementos LR(0).

En la figura 4 se muestra el algoritmo que ayuda a calcular la colección canónica de conjuntos de elementos LR(0) para una gramática aumentada G'.

Algoritmo de construcción de una tabla de análisis sintáctico SLR

A continuación se muestra el algoritmo de construcción de una tabla de análisis sintáctico SLR o LR(0).

ENTRADA: Una gramática aumentada G'.

SALIDA: Las funciones ACCION e ir_A para G' de la tabla de análisis sintáctico SLR.

MÉTODO:

1. Construir $C = \{ I_0, I_1, \ldots, I_n \}$, la colección de conjuntos de elementos LR(0) para G'.
2. El estado i se construye a partir de I_i. Las acciones de análisis sintáctico para el estado i se determinan de la siguiente forma:
   (a) Si $[A \rightarrow \alpha \cdot a \beta]$ está en I_i e ir_A$(I_i, a) = I_j$, entonces establecer ACCION$[i, a]$ a "desplazar j". Aquí, a debe ser una terminal.
   (b) Si $[A \rightarrow \alpha \cdot]$ está en I_i, entonces establecer ACCION$[i, a]$ a "reducir $A \rightarrow \alpha$" para toda a en SIGUIENTE(A); aquí, A tal vez no sea S'.
   (c) Si $[S' \rightarrow S \cdot]$ está en I_i, entonces establecer ACCION$[i, \$]$ a "aceptar".

   Si resulta cualquier acción conflictiva debido a las reglas anteriores, decimos que la gramática no es SLR(1). El algoritmo no produce un analizador sintáctico en este caso.
3. Las transiciones de ir_A para el estado i se construyen para todos los no terminales A, usando la regla: Si ir_A$(I_i, A) = I_j$, entonces ir_A$[i, A] = j$.
4. Todas las entradas que no estén definidas por las reglas (2) y (3) se dejan como "error".
5. El estado inicial del analizador sintáctico es el que se construyó a partir del conjunto de elementos que contienen $[S' \rightarrow \cdot S]$.

Figura 5. Algoritmo de construcción de una tabla de análisis sintáctico SLR o LR(0).

Algoritmo de subconjuntos LR1

Es posible transportar más información en el estado, que nos permita descartar algunas reducciones inválidas mediante A->α. Al dividir estados según sea necesario, podemos hacer que cada estado de un analizador sintáctico LR indique con exactitud qué símbolos de entrada pueden ir después de un mango a para el cual haya una posible reducción a A.

La información adicional se incorpora al estado mediante la redefinición de elementos, para que incluyan un símbolo terminal como un segundo componente. La forma general de un elemento se convierte en [A -> α .β, *a*], en donde A -> αβ es una producción y *a* es un terminal o el delimitador $ derecho. A un objeto de este tipo le llamamos **elemento LR(1)**.

El método para construir la colección de conjuntos de elementos LR(1) válidos es en esencia el mismo que para construir la colección canónica de conjuntos de elementos LR(0). Sólo necesitamos modificar los dos procedimientos CERRADURA e ir _ A .

```
ConjuntoDeElementos CERRADURA(I) {
        repeat
                for ( cada elemento [A → α·Bβ, a] en I )
                        for ( cada producción B → γ en G' )
                                for ( cada terminal b en PRIMERO(βa) )
                                        agregar [B → ·γ, b] al conjunto I;
        until no se agreguen más elementos a I;
        return I;
}

ConjuntoDeElementos ir_A(I, X) {
        inicializar J para que sea el conjunto vacío;
        for ( cada elemento [A → α·Xβ, a] en I )
                agregar el elemento [A → αX·β, a] al conjunto J;
        return CERRADURA(J);
}

void elementos(G') {
        inicializar C a CERRADURA({[S' → ·S, $]});
        repeat
                for ( cada conjunto de elementos I en C )
                        for ( cada símbolo gramatical X )
                                if ( ir_A(I, X) no está vacío y no está en C )
                                        agregar ir_A(I, X) a C;
        until no se agreguen nuevos conjuntos de elementos a C;
}
```

Figura 6. Funciones usadas para la construcción de elementos LR(1) para la gramática G'.

Algoritmo de construcción de una tabla de análisis sintáctico LR(1)

A continuación se muestra el algoritmo para la construcción de una tabla de análisis sintáctico LR(1) o LR canónico.

ENTRADA: Una gramática aumentada G'.

SALIDA: Las funciones ACCION e ir_A de la tabla de análisis sintáctico LR canónico para G'.

MÉTODO:

1. Construir $C' = \{ I_0, I_1, \cdots, I_n \}$, la colección de conjuntos de elementos LR(1) para G'.
2. El estado i del analizador sintáctico se construye a partir de I_i. La acción de análisis sintáctico para el estado i se determina de la siguiente manera.
   (a) Si $[A \to \alpha \cdot a\beta, b]$ está en I_i, e ir_A$(I_i, a) = I_j$, entonces hay que establecer ACCION$[i, a]$ a "desplazar j". Aquí, a debe ser una terminal.
   (b) Si $[A \to \alpha \cdot, a]$ está en I_i, $A \neq S'$, entonces hay que establecer ACCION$[i, a]$ a "reducir $A \to \alpha$".
   (c) Si $[S' \to S \cdot, \$]$ está en I_i, entonces hay que establecer ACCION$[i, \$]$ a "aceptar".

   Si resulta cualquier acción conflictiva debido a las reglas anteriores, decimos que la gramática no es LR(1). El algoritmo no produce un analizador sintáctico en este caso.
3. Las transiciones ir_A para el estado i se construyen para todos los no terminales A usando la regla: Si ir_A$(I_i, A) = I_j$, entonces ir_A$[i, A] = j$.
4. Todas las entradas no definidas por las reglas (2) y (3) se vuelven "error".
5. El estado inicial del analizador sintáctico es el que se construye a partir del conjunto de elementos que contienen $[S' \to \cdot S, \$]$.

Figura 7. Algoritmo de construcción de la tabla de análisis sintáctico LR(1).

Algoritmo de construcción de la tabla de análisis sintáctico LALR

Por último, se mostrará a continuación el algoritmo para generar tablas de análisis sintáctico LALR, se puede observar en el algoritmo que usa el algoritmo de subconjuntos para la construcción de la tabla LR(1).

ENTRADA: Una gramática aumentada G'.

SALIDA: Las funciones de la tabla de análisis sintáctico LALR ACCION e ir_A para G'.

MÉTODO:

1. Construir $C = \{I_0, I_1, \ldots, I_n\}$, la colección de conjuntos de elementos LR(1).
2. Para cada corazón presente entre el conjunto de elementos LR(1), buscar todos los conjuntos que tengan ese corazón y sustituir estos conjuntos por su unión.
3. Dejar que $C' = \{J_0, J_1, \ldots, J_m\}$ sean los conjuntos resultantes de elementos LR(1). Las acciones de análisis sintáctico para el estado i se construyen a partir de J_i, de la misma forma que en el Algoritmo 4.56. Si hay un conflicto de acciones en el análisis sintáctico, el algoritmo no produce un analizador sintáctico y decimos que la gramática no es LALR(1).
4. La tabla ir_A se construye de la siguiente manera. Si J es la unión de uno o más conjuntos de elementos LR(1), es decir, $J = I_1 \cap I_2 \cap \cdots \cap I_k$, entonces los corazones de ir_A(I_1, X), ir_A(I_2, X), ..., ir_A(I_k, X) son iguales, ya que $I_1, I_2, \ldots, I_k$ tienen el mismo corazón. Dejar que K sea la unión de todos los conjuntos de elementos que tienen el mismo corazón que ir_A(I_1, X). Entonces, ir_A$(J, X) = K$.

Figura 8. Algoritmo para la construcción de la tabla de análisis sintáctico LALR.

Características de la implementación de la práctica

En la presente práctica se implementará la clase que generarán la tabla LR(0) para una gramática libre de contexto. 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.

El lenguaje de programación utilizado es C++ con la versión 7.5.0.

Las características del archivo de entrada y salida serán específicadas durante el desarrollo.

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 LR es de la siguiente forma:

Primera línea: Consistirá de un solo carácter que denotará el símbolo inicial de la gramática, debe pertenecer al conjunto de los símbolos no terminales, que son las mayúsculas del alfabeto inglés a excepción de la *Z*, ya que será usada para aumentar la gramática.

Siguientes líneas: cadenas de la forma *S->alfa*, donde, *S* es un carácter de las mayúsculas del alfabeto inglés denotando la cabeza de la producción a excepción de *Z* (será usado para extender la gramática), seguido de la cadena "*->*", y seguido de esta cadena viene *alfa* que es una secuencia de caracteres mayúsculas y minúsculas, incluyendo a *'3'* que denota a épsilon, 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 debe denotar un solo cuerpo por producción y se especificará una sola producción por línea, es decir, la especificación de varios cuerpos por producción en una sola línea, (por ejemplo: *S->a*|*b*) no está permitida.

El alfabeto de entrada consistirá solamente de:

Mayúsculas del alfabeto inglés a excepción de *Z*: 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 LR consistirá de lo siguiente:

Primera línea: se compondrá de la cadena "Símbolo inicial: " seguida del carácter que denota al símbolo inicial de la gramática de entrada.

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

Siguiente línea: Corresponderá a el encabezado de la tabla que son los terminales, el símbolo '$' y los no terminales dependiendo de la tabla generada.

Siguientes x líneas: Serán las filas de la tabla, que consisten en los números correspondientes a el número asignado a los conjuntos obtenidos en el algoritmo de los subconjuntos seguidos de las entradas de la tabla. Las entradas de la tabla, serán de 4 tipos:

- OK: Es la entrada que indica la aceptación de una cadena.

- d#: El carácter 'd' seguido de un número decimal, indica el desplazamiento al conjunto #.

- r#: El carácter 'r' seguido de un número decimal #, indica la reducción con la producción numerada con el número #.

- #: Un número decimal # solo, indica la acción de MOVER.

En caso de que no exista una entrada, se considera un error al hacer el análisis sintáctico.

2. Diseño del diagrama de clases

Antes de comenzar la implementación en el lenguaje de programación, se necesita hacer un bosquejo del programa, para esto se realizará un diagrama de clases que permitirá programar de una forma más organizada y ayudará a eficientizar el trabajo. A continuación, se muestra el diagrama de clases diseñado para la implementación del programa.

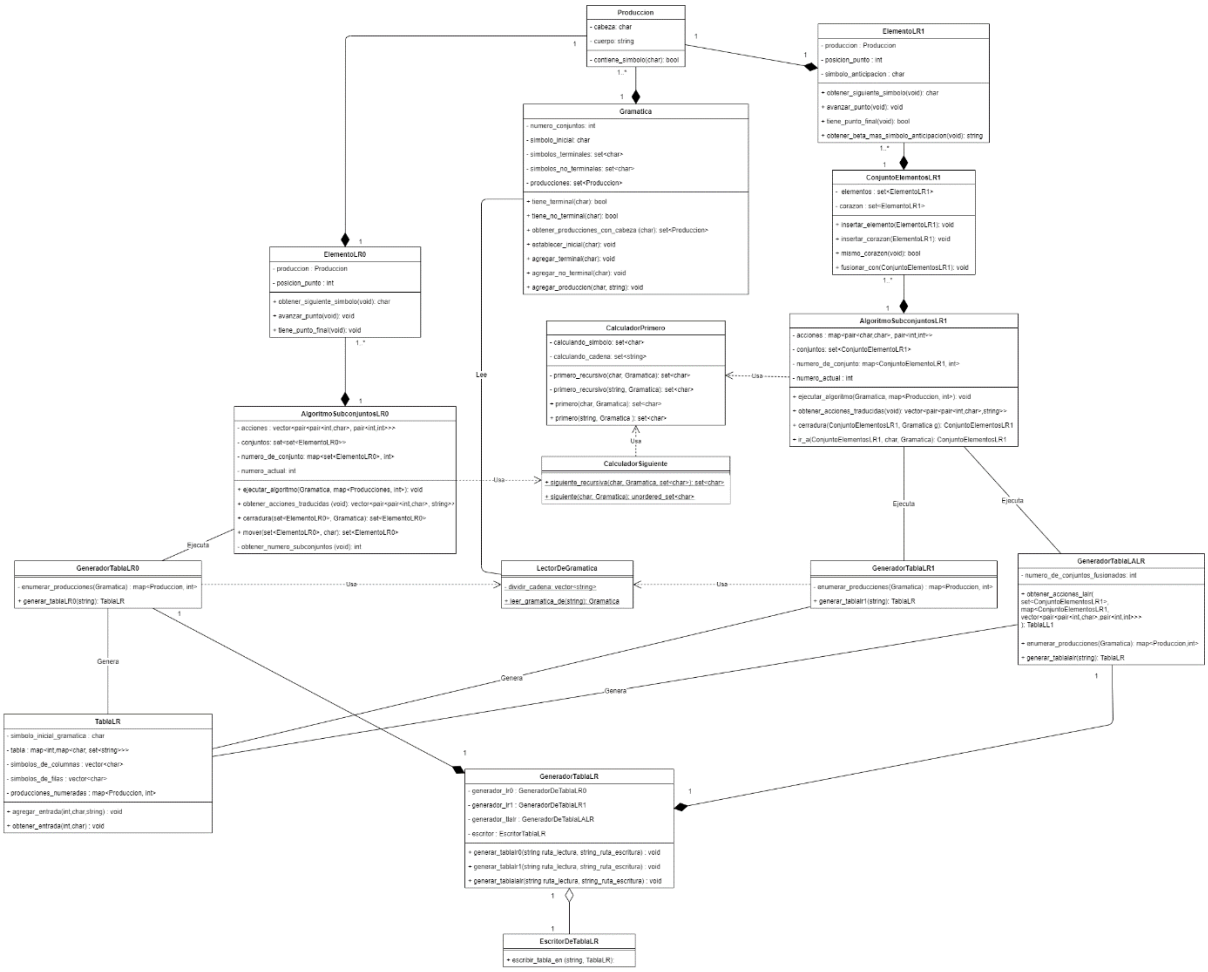


Figura 9. Diagrama de clases para la obtención de las tablas de análisis sintáctico LR.

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 usará una gramática con la que ya se ha trabajado manualmente el algoritmo para verificar el funcionamiento del programa.

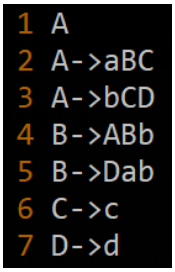


Figura 10. Contenido del archivo de entrada que especifica la gramática.

En la Figura 10 se muestra el contenido del archivo “Gramatica.txt” que contiene la gramática y que se entregará a el programa para que generen las tablas LR. Ahora, se procederá a ejecutar el programa y se observará la salida. Se comenzará por obtener la tabla de análisis sintáctico LR(0).

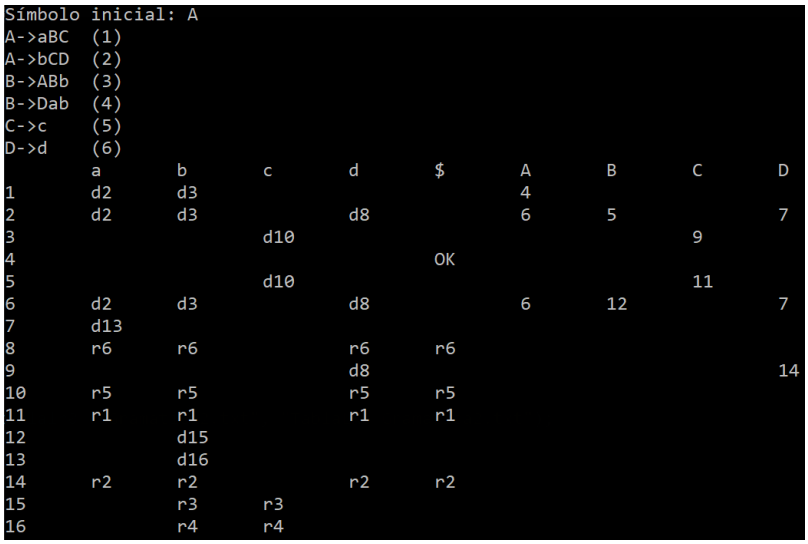


Figura 11. Tabla de análisis sintáctico LR(0) para la gramática especificada en la figura7.

Para verificar que la tabla es correcta, se analizará una cadena obtenida mediante derivaciones de la gramática, con el algoritmo de análisis sintáctico LR.

```
Gramatica:
A->aBC
A->bCD
B->ABb
B->Dab
C->c
D->d

Derivación de la cadena:
A->aBC->aDabC->adabC->adabc

Cadena de entrada: adabc$

Accion                  Pila
1,a->d2                 1 $
2,d->d8                 2 1 $
8,a->r6                 8 2 1 $ r6=(D->d) 2,D->7
7,a->d13                7 2 1 $
13,b->d16               13 7 2 1 $
16,c->r4                16 13 7 2 1 $ r4=(B->Dab) 2,B->5
5,c->d10                5 2 1 $
10,$->r5                10 5 2 1 $ r5=(C->c) 5,C->11
11,$->r1                11 5 2 1 $ r5=(A->aBC) 1,A->4
4,$->OK                 4 1 $
Por lo tanto, la cadena adabc pertenece al lenguaje
generado por la gramática.
```

Figura 12. Ejecución del algoritmo de análisis sintáctico LR con la tabla LR(0) obtenida.

En la figura 12, se puede observar la ejecución del algoritmo de análisis sintáctico LR para una cadena que fue derivada de la gramática a la cual se le generó la tabla LR(0), y dado que el algoritmo de análisis sintáctico que usa la tabla LR(0) generada por el programa hecho en esta práctica aceptó la cadena, entonces podemos concluir que la tabla LR(0) generada es correcta y el programa realizado funciona correctamente.

Ahora se generará la tabla de análisis sintáctico LR(1) para la misma gramática especificada en la figura 10 mediante el programa, y se muestra el resultado a continuación.

```
Símbolo inicial: A
A->aBC  (1)
A->bCD  (2)
B->ABb  (3)
B->Dab  (4)
C->c    (5)
D->d    (6)
        a       b       c       d       $       A       B       C       D
1       d2      d3                              4
2       d5      d7              d10             8       6               9
3                       d12                             11
4                                       OK
5       d5      d7              d10             8       13              9
6                       d15                             14
7                       d12                             16
8       d5      d7              d10             17      18              19
9       d20
10      r6
11                              d22                             21
12                              r5
13                      d24                             23
14                              r1
15                              r5
16                              d26                             25
17      d5      d7              d10             17      27              19
18              d28
19      d29
20              d30
21                              r2
22                              r6
23      r1      r1              r1
24      r5      r5              r5
25      r2      r2              r2
26      r6      r6              r6
27              d31
28                      r3
29              d32
30                      r4
31              r3
32              r4
```

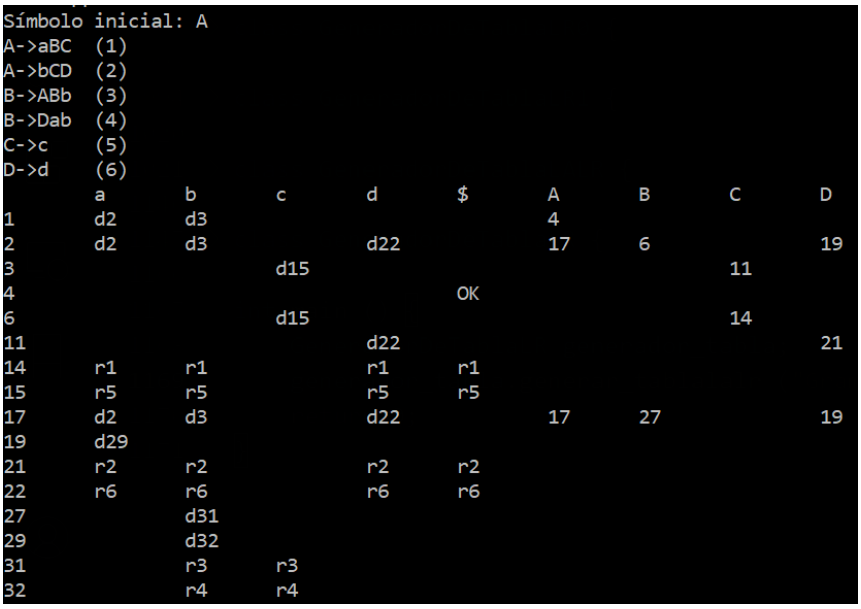
Figura 13. Tabla de análisis sintáctico LR(1) para la gramática especificada en la figura 7.

Se realizará el análisis con la misma cadena que en la tabla LR(0) para verificar si es correcta.

```
 1 Cadena de entrada: adabc$
 2
 3 Acción                  Pila
 4 1,a->d2                 1 $
 5 2,d->d10                2 1 $
 6 10,a->r6                10 2 1 $ (D->d)
 7 9,a->d20                9 2 1 $
 8 20,b->d30               20 9 2 1 $
 9 30,c->r4                30 20 9 2 1 $ (B->Dab)
10 6,c->d15                6 2 1 $
11 15,$->r1                15 6 2 1 $ (A->aBC)
12 4,$->OK                 4 1 $
13 Por lo tanto, la cadena "adabc" pertenece al
14 lenguaje generado por la gramática.
```

Figura 14. Ejecución del algoritmo de análisis sintáctico LR con la tabla LR(1) obtenida, especificada en la figura 13.

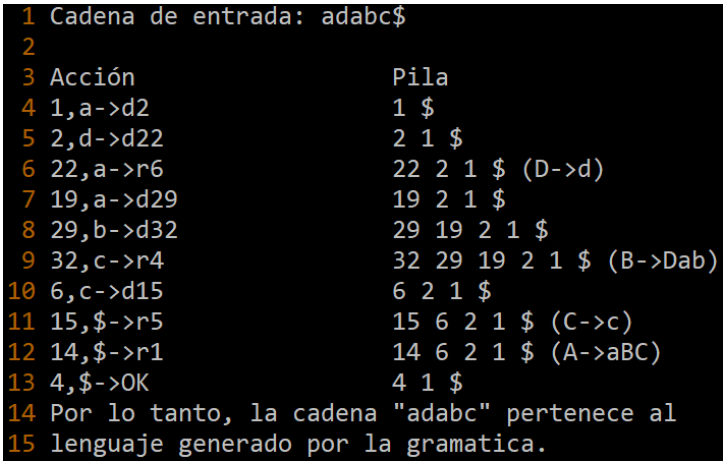
Por último, se procederá a ejecutar el programa para obtener la tabla de análisis sintáctico LALR para la gramática especificada en la figura 7. A continuación se muestra el resultado de la ejecución.



```
Símbolo inicial: A
A->aBC  (1)
A->bCD  (2)
B->ABb  (3)
B->Dab  (4)
C->c    (5)
D->d    (6)
        a       b       c       d       $       A       B       C       D
1       d2      d3                              4
2       d2      d3              d22             17      6               19
3                       d15                             11
4                                       OK
6                       d15                             14
11                              d22                             21
14      r1      r1              r1      r1
15      r5      r5              r5      r5
17      d2      d3              d22             17      27              19
19      d29
21      r2      r2              r2      r2
22      r6      r6              r6      r6
27              d31
29              d32
31              r3      r3
32              r4      r4
```

Figura 15. Tabla de análisis sintáctico LALR para la gramática especificada en la figura 10.

Ahora, al igual que con las tablas anteriores se ejecutará el algoritmo de análisis sintáctico para la cadena derivada de la gramática original para verificar que la tabla obtenida es correcta.



```
 1 Cadena de entrada: adabc$
 2
 3 Acción                  Pila
 4 1,a->d2                 1 $
 5 2,d->d22                2 1 $
 6 22,a->r6                22 2 1 $ (D->d)
 7 19,a->d29               19 2 1 $
 8 29,b->d32               29 19 2 1 $
 9 32,c->r4                32 29 19 2 1 $ (B->Dab)
10 6,c->d15                6 2 1 $
11 15,$->r5                15 6 2 1 $ (C->c)
12 14,$->r1                14 6 2 1 $ (A->aBC)
13 4,$->OK                 4 1 $
14 Por lo tanto, la cadena "adabc" pertenece al
15 lenguaje generado por la gramatica.
```

Figura 16. Ejecución del algoritmo de análisis sintáctico LR con la tabla LALR obtenida, especificada en la figura 15.

En la figura 14 también se obtuvo que la cadena pertenece al lenguaje generado por la gramática de entrada, por lo que se comprueba que la tabla LALR obtenida por el programa también es correcta.

Así, se puede concluir, dado que las tres tablas de análisis sintáctico son correctas, que el programa genera tablas LR correctas.

Conclusiones

Durante la realización de la práctica se tuvieron algunas dificultades debido que el algoritmo para generar la tablas LR es algo difícil de modelar con código. Se pudo llegar a la conclusión de que es mejor programar un generador de tablas LR que hacer el procedimiento a mano, ya que puede ser un procedimiento bastante largo.

Durante la realización de la práctica también se pudo obtener una mejor comprensión de los algoritmos para generar la tabla LR0. También se pudieron obtener diversos conocimientos de programación, específicamente del lenguaje de programación C++, ya que se enfrentaron diferentes retos para modelar mediante objetos el algoritmo.

Referencias

- Aho A., Lam M. (2008) *Compiladores, principios técnicas y herramientas. Segunda edición*. México: Pearson Education.
- Hopcroft, J.E., Ullman, J.D. (1979) *Introduction to Automata Theory, Languages and Computation*. Pearson Addison-Wesley.