TRABAJO DE LA SEMANA 4 A LA SEMANA 8 POR EQUIPO

INSTITUTO TECNOLOGICO DE IZTAPALAPA

ALUMNOS:

MARTINEZ HERNANDEZ
RUIZ MORALES CORINA

MATERIA:
LENGUAJES Y AUTOMATAS 2

PROFESOR:
PARRA HERNANDEZ ABIEL TOMAS

"SEMANA 4"

- Mod-01 Lec-01 An Overview of a Compiler

Un compilador es un programa informático que traduce un programa escrito aun lenguaje de programación a otro lenguaje de programación, es decir, convierte un lenguaje de alto nivel a un lenguaje de bajo nivel o de máquina de tal forma que sea más entendible.

Fases de un compilador:

Análisis léxico: constituye la primera fase y se lee el código fuente de izquierda a derecha y se agrupa en componentes léxicos que son los toquens que son secuencias de caracteres que tiene un significado y todos los espacios en blanco se eliminan del programa fuente también se comprueban que los símbolos y palabras clave sean escritos correctamente, necesita los métodos de especificación, reconocimiento de patrones se usan principalmente los autómatas infinitos que acepten expresiones regulares, también un analizador léxico es la parte de traductor que maneja la entrada del código fuente por eso debe funcionar lo mejor posible.

Analizador sintáctico: en esta fase los caracteres o componentes léxicos se agrupan jerárquicamente en grupos gramaticales que el compilador utiliza para sintetizar la salida se comprueba si es correcto, obedece la gramática, esto se hace mediante un árbol de análisis sintáctico.

Análisis semántico: es el que revisa el programa fuente para ver si hay errores semánticos y reúnen información para la parte posterío, en ella se utiliza la estructura jerárquica determinada por la fase anterior para identificar los operadores y operando las preposiciones.

Generación de código intermedio: es un programa para una maquina abstracta esta representación debe tener dos propiedades fáciles de producir y fácil de traducir al programa objeto, la representación puede tener diferentes formas código de tres direcciones.

Optimización de código: es la fase donde se mejora el código intermedio de modo que se genere un código maquina más rápido de ejecutar.

- Mod-02 Lec-02 Lexical Analysis - Part 1

Constituye la primera fase del proceso de compilado, es donde se leen el programa fuente y se agrupan los tokens, esto es una cadena de caracteres que tienen un significado y un lexema, esto son los temporales. Esto es a la par ya que un token puede tener mil lexemas dentro.

Vamos explicar la fase de análisis léxico

Programa o lenguaje fuente, pasa por fase de análisis léxico esto genera una cantidad de errores y se trasforman en tokens o lexemas estos serán el anclaje o punto de entrada para el análisis sintáctico.
Las funciones del analizador léxico:

*Eliminar los espacios en blanco o información innecesaria
*Se comprueba que las palabras reservadas, operadores entre otros sean escritos correctamente.
*Registro de líneas de la línea que está analizando en el momento.
*Genera los mensajes de error léxico y los recupera.
*Convierte el programa fuente a cadena de tokens.

La manera en que trata un error el analizador léxico:

Tiene tres reglas básicas
1.- Modo pánico: obviar los siguientes caracteres de entrada hasta encontrar un token bien armado.
2.- Tratar de arreglar la entrada: Borra el carácter extraño, remplaza caracteres erróneos.
3.- Palabras reservadas: no se puede tomar una palabra reservada para que se utilice como identificador.

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLOGICO DE IZTAPALAPA

ALUMNOS:

MARTINEZ HERNANDEZ
RUIZ MORALES CORINA

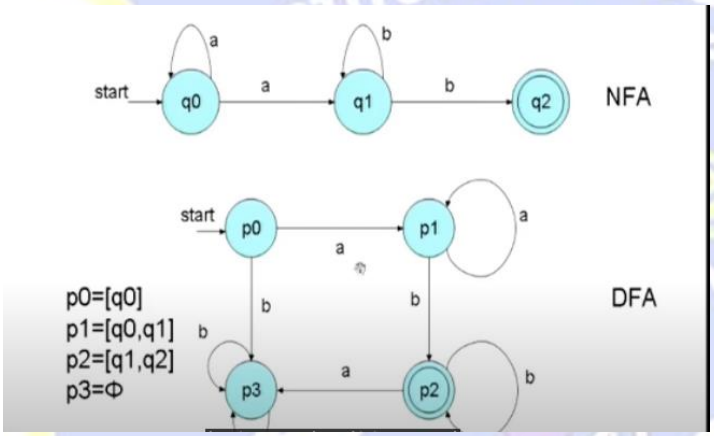
MATERIA:
LENGUAJES Y AUTOMATAS 2

PROFESOR:
PARRA HERNANDEZ ABIEL TOMAS

“SEMANA 5”

Es la segunda parte del análisis léxico, en este video analizaremos el por qué debe separarse el análisis léxico del análisis sintáctico, veremos tokens, patrones, lexemas y las dificultades que hay en el análisis léxico.

En el video nos muestran un ejemplo de NFA que debemos construir un DFA, que axceptua exactamente.

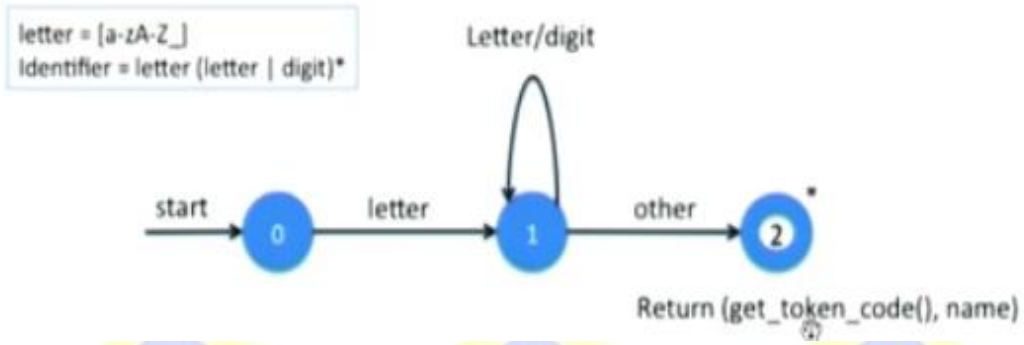


El NFA muestra dos transiciones en el símbolo a, en el símbolo b también muestra dos transiciones, se muestra que en q0 y q1 no hay transición en a, entonces la forma en la que procede construir el DFA es bastante sencillo.

El estado inicial se sabe que es q0, se es posible hacer una transición a un conjunto de estados desde cada estado de NFA, por lo que cada estado del DFA corresponderá a un subconjunto de del total de estados. Entonces p0 tiene un solo estado (q0) del NFA mientras que p1 tiene dos estados (q0, q1) del NFA, p2 tiene (q1, q2) del NFA y p3 no tiene nada es to es que es un estado phi que en realidad es un estado error.

Teniendo lo anterior claro, la forma de construir el DFA se inicia en q0 hacia p1 en p1 se tiene un retorno (estado), continua p1 hacia p2 con un estado b y un retorno, de p2 a p3 con estados en a marcando dos retornos en los estados a y b y para finalizar el estado p3 hacia el p0 en el estado b.

En el video número dos nos muestra un diagrama de transición para identificadores y palabras reservadas.



Vamos a considerar este diagrama que representa palabras reservadas, este diagrama comienza en el estado 0, seguimos consumiendo letras y dígitos en el estado 1 y cuando es diferente a la letra o digito llegamos al estado 2, donde volvemos la palabra simbólica (que es donde se va a traducir el programa).

```
TOKEN gettoken() {
   TOKEN mytoken; char c;
   while(1) { switch (state) {
     /* recognize reserved words and identifiers */
       case 0: c = nextchar(); if (letter(c))
               state = 1; else state = failure();
               break;
       case 1: c = nextchar();
               if (letter(c) || digit(c))
               state = 1; else state = 2; break;
       case 2: retract(1);
               mytoken.token = search_token();
               if (mytoken.token == IDENTIFIER)
               mytoken.value = get_id_string();
               return(mytoken);
```

Nuestro analizador léxico se llama get token y devuelve un tipo llamado token, dentro del analizador léxico tenemos dos variables locales que son mi token y C. entonces hay un ciclo while, que se ejecuta hasta el final del archivo, por lo que, para empezar estaremos en el 0. Y es una letra que cambiamos para indicar uno, de lo contario informamos que es una falla. Y en el estado uno verificamos si es una letra o digito después de leer en el mismo estado, si es de lo contrario cambiamos el estado dos. En el estado en realidad nos estamos preparando para anunciar un token, por lo que el símbolo que nos llevó al estado dos no se consume, se devuelve a la entrada que se obtiene el token buscando en la tabla de tokens. Se sabe que la palabra reservada es devuelto por el token de búsqueda y si es un identificador, simplemente obtenemos la cadena correspondiente al identificador, la colocamos en nuestro token y devolvemos.

TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLOGICO DE IZTAPALAPA

ALUMNOS:
MARTINEZ HERNANDEZ
RUIZ MORALES CORINA

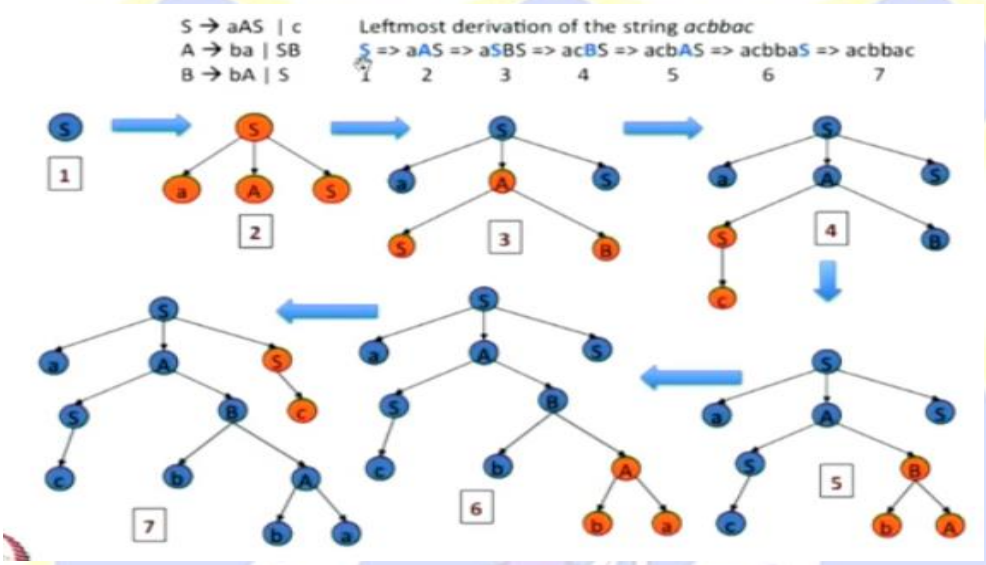
MATERIA:
LENGUAJES Y AUTOMATAS 2

PROFESOR:
PARRA HERNANDEZ ABIEL TOMAS

“SEMANA 6”

En este video nos van hablar sobre la gramática libre de contexto, la creación de autómatas hacia abajo y el movimiento de análisis de arriba hacia abajo y a la viceversa.

Ejemplo en el análisis LL de arriba hacia abajo

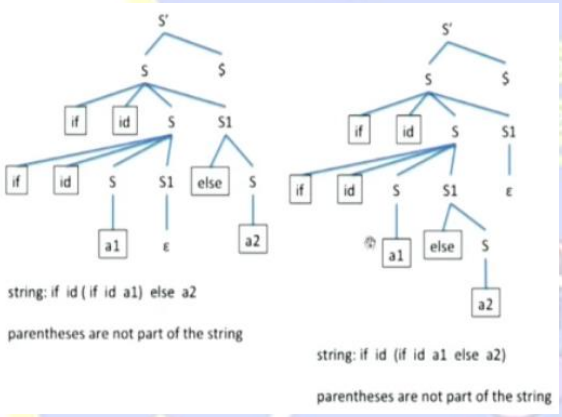


Vamos a considerar la cadena a, c, b, b, a, c y veamos la derivación más a la izquierda de esta cadena. En azul se ve el símbolo que se va a expandir a continuación, S es el símbolo de inicio para comenzar aplicamos la producción a A, S luego aplicamos la producción que va a ser SB. Obtenemos esta forma enunciativa a S, BS ahora la izquierda más no termina en S, aplicamos la producción S así obteniendo una C y permanecen los dos símbolos B y S. B resulta ser el mas a la izquierda, así que expande B por b A para obtener acs, A, S esto implica que expande una ganancia, es decir, conoces a A . Finalmente S se expande a c. así obtenemos nuestra cadena a, c, b, b, ac que esta es la derivación de la cadena. Cuando hacemos el análisis sintáctico LL o de arriba hacia abajo usando la estrategia LL, la construcción del árbol de análisis ocurre de esta manera, por lo que simplemente S. Ahora hay dos producciones para S yendo a A, s y S yendo a C, la razón se llama análisis sintáctico predictivo es que necesitamos adivinar qué producción es aplicable en este particular punto o necesitamos predecir la producción que es aplicable en este punto, hasta aquí sabemos que S va con s. en este momento ocurre la expansión y el árbol de análisis sintáctico, ahora que S se ha expandido queda

enunciado el formulario que tenemos es un AS que es visible en el esquema se expande la producción A que va a SB y luego el siguiente no terminal es el que queda, de modo que se expande de S a C, luego tenemos dos B y s no terminales. S va a ac se ha completado en este paso, entonces B que va a b es la siguiente expansión que ocurre.

Tenemos A y S, entonces A es el más a la izquierda de modo que se expande A yendo ab y finalmente, el no terminal sobrante se expande por S yendo a c. Como se puede ver la construcción del árbol de análisis y la derivación más ala izquierda están sincronizadas, sabemos cuál con S no terminal particular expandido en el árbol de análisis sintáctico.

LL tabla de construcción



La cadena if i d, if i d a 1 else, a 2 sabemos que producir estos dos árboles de análisis, hasta aquí los dos árboles son idénticos, producimos un épsilon o producimos else S a 2 else y para S la expansión es si i, ds, s1 por lo que si i d, s1 y para este s producimos un épsilon o lo expandimos a otros este la ambigüedad de siempre. Por esto los dos árboles de análisis son diferentes y debido al gramar es ambiguo y falla la prueba LL.

TECNOLÓGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE IZTAPALAPA

INSTITUTO TECNOLOGICO DE IZTAPALAPA

ALUMNOS:

MARTINEZ HERNANDEZ
RUIZ MORALES CORINA

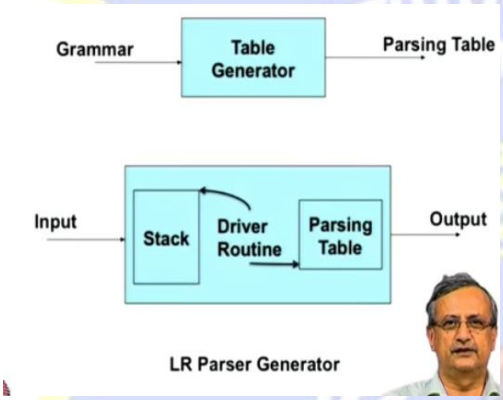
MATERIA:
LENGUAJES Y AUTOMATAS 2

PROFESOR:
PARRA HERNANDEZ ABIEL TOMAS

“SEMANA 7”

Gramática, análisis sintáctico, autómatas pushdown y análisis sintáctico sin contexto.

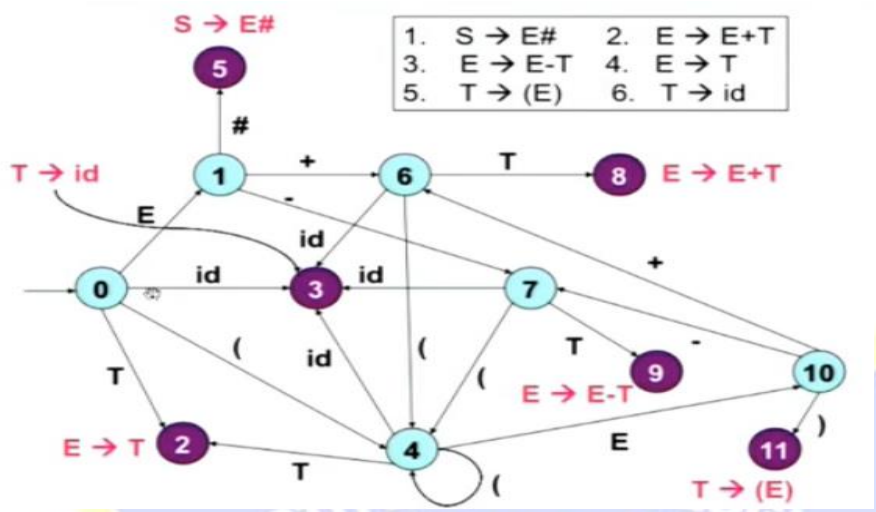
Generación de analizador.



El generador de analizador sintáctico es un dispositivo muy simple que toma gramáticas como entrada y genera una tabla de análisis sintáctico LR. La tabla del analizador encaja en otra caja, que contiene una pila y una rutina de unidad, todo este conjunto es el analizador.

En esta imagen se puede apreciar la rutina de unidad de pila y la tabla de análisis juntos, esto forma el analizador, toma el programa como entrada y entrega como salida posiblemente una sintaxis libre.

DFA para prefijos viables-lR(0)



Con respecto a esto DFA consideramos la identificación de cadena simple que se deriva de esta gramática. Comenzamos con S yendo a E, aplicamos una producción y luego cuando aplicamos la producción E yendo a T a id obtenemos el id de cadena asi que se nota como pasa usando autómata LR0.

Comenzamos con el estado 0 que es el estado inicial y luego en el id de entrada vamos al estado 3, debemos recordar el estado 0 en la pila, ahora la identificación se envía a la pila cuando vemos y los lleva al estado 3 que nuevamente va a empujar hacia debajo de la pila por lo que en lugar de 3 se dice que es un estado reducido y la reducción es por la producción t yendo a id, por lo que la implicación de esto sería hacer estallar el símbolo superior de la pila, dado que el estado también está incluido sacamos símbolos de la pila y eso en realidad expone el estado 0 en la pila. Apilar ahora el no terminal en el lado izquierdo es t, por lo que aplicamos la función en el estado 0 y esto nos lleva al estado número 2. Ahora hemos presionado la t no terminal en la pila y el estado numero 2 también está en la pila ahora dice que se reduce de E a T así que sacamos el estado 2 y la t no terminal de la pila nuevamente obtenemos el estado 0 como la parte superior de la pila y ahora dado que no termina en el lado izquierdo es E, aplicamos la función ir a en el estado 0 y el no terminal E que nos lleva al estado número 1. El siguiente símbolo de entrada tiene un estado de cambio, no es un estado de reducción y luego pasa al estado 5 donde indica una reducción de s yendo a E por lo que nuevamente sacamos por lo que nuevamente sacamos elementos para la pila. El estado S no terminal a la pila y eso es una aceptación de excepción así es como pasa la identificación de la cadena del terminal.

Considere una terminal un poco más complicada que sabe string id más id por lo que id más id se deriva de esta gramática usando S yendo a E tiene luego E yendo E más T, este id ira a E hasta a T y luego ira a id de modo que así es como se obtendría id más id se derivaría nuevamente, comenzamos desde 0 llegamos al estado 3 lo que indica una reducción, regresamos a T y vamos al estado 2 que nuevamente indica una reducción, así que regresemos al estado 0 símbolos pop luego vaya al estado 1 en E, por lo que ahora el signo más nos lleva al estado 6 y nos lleva a recordar que 3 y 6 están en la pila por lo que sacamos estos identificadores y la pila para exponer el no terminal 6 y 6 en el no terminal T toma . En el estado 8 tenemos la pila E uno más 6 T y 8 lo que indica una reducción de E a E más T así que volvemos hasta el estado 0 y en E volvemos al estado 1 sucesivamente lo ha hecho pasar al estado 5 y aceptamos la cadena así es como ocurre un análisis usando este autómata.

INSTITUTO TECNOLOGICO DE IZTAPALAPA

ALUMNOS:
MARTINEZ HERNANDEZ
RUIZ MORALES CORINA

MATERIA:
LENGUAJES Y AUTOMATAS 2

PROFESOR:
PARRA HERNANDEZ ABIEL TOMAS

“SEMANA 8”

Escritura de una gramática
Las gramáticas son capaces de describir casi la mayoría de la sintaxis de los lenguajes de programación. Por ejemplo, el requerimiento de que los identificadores deben declararse antes de usarse, no puede describirse mediante una gramática libre de contexto. Por lo tanto, las secuencias de los tokens que acepta un analizador sintáctico forman un super conjunto del lenguaje de programación; las fases siguientes del compilador deben analizar la salida del analizador sintáctico, para asegurar que cumpla con las reglas que no verifica el analizador sintáctico.

Comparación entre análisis léxico y análisis sintáctico
Sería razonable preguntar: "¿Por qué usar expresiones regulares para definir la sintaxis léxica de un lenguaje?" Existen varias razones.
1. Al separar la estructura sintáctica de un lenguaje en partes léxicas y no léxicas, se proporciona una manera conveniente de colocar en módulos la interfaz de usuario de un compilador en dos componentes de un tamaño manejable.

2. Las reglas léxicas de un lenguaje son con frecuencia bastante simples, y para describirlas no necesitamos una notación tan poderosa como las gramáticas.

3. Por lo general, las expresiones regulares proporcionan una notación más concisa y fácil de entender para los tokens, en comparación con las gramáticas.

4. Pueden construirse analizadores léxicos más eficientes en forma automática a partir de expresiones regulares, en comparación con las gramáticas arbitrarias.

Eliminación de la ambigüedad

Algunas veces, una gramática ambigua puede rescribirse para eliminar la ambigüedad. Como ejemplo, vamos a eliminar la ambigüedad de la siguiente gramática del "else colgante":
Eliminación de la recursividad por la izquierda

Una gramática es recursiva por la izquierda si tiene una terminal A tal que haya una derivación A ⇒ Aα para cierta cadena α. Los métodos de análisis sintáctico descendente no pueden manejar las gramáticas recursivas por la izquierda, por lo que se necesita una transformación para eliminar la recursividad por la izquierda.

La recursividad inmediata por la izquierda puede eliminarse mediante la siguiente técnica, que funciona para cualquier número de producciones A.

tica de expresiones (4.1). El par recursivo por la izquierda de las producciones $E \to E + T \mid T$ se sustituye mediante $E \to T\,E'$ y $E' \to +\,T\,E' \mid \epsilon$. Las nuevas producciones para T y T' se obtienen de manera similar, eliminando la recursividad inmediata por la izquierda. □

La recursividad inmediata por la izquierda puede eliminarse mediante la siguiente técnica, que funciona para cualquier número de producciones A. En primer lugar, se agrupan las producciones de la siguiente manera:

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

en donde ninguna β_i termina con una A. Después, se sustituyen las producciones A mediante lo siguiente: