INSTITUTO TECNOLOGICO DE IZTAPALAPA

INGENERIA EN SISTEMAS COMPUTACIONALES

Lenguajes y autómatas 2

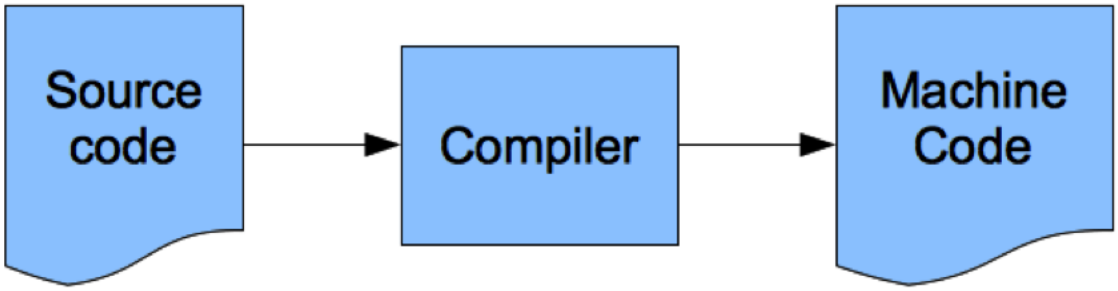
Ramírez Peña Carlos Iván

24/01/21

Primera semana

Introducción a Compiladores

Es un Software que traduce un programa escrito en un lenguaje de programación de alto nivel (C / C ++, COBOL, etc.) en lenguaje de máquina. Un compilador generalmente genera lenguaje ensamblador primero y luego traduce el lenguaje ensamblador al lenguaje máquina. Una utilidad conocida como «enlazador» combina todos los módulos de lenguaje de máquina necesarios en un programa ejecutable que se puede ejecutar en la computadora.



El proceso de traducción se compone internamente de varias etapas o fases, que realizan distintas operaciones lógicas. Es útil pensar en estas fases como en piezas separadas dentro del traductor, y pueden en realidad escribirse como operaciones codificadas separadamente aunque en la práctica a menudo se integren juntas.

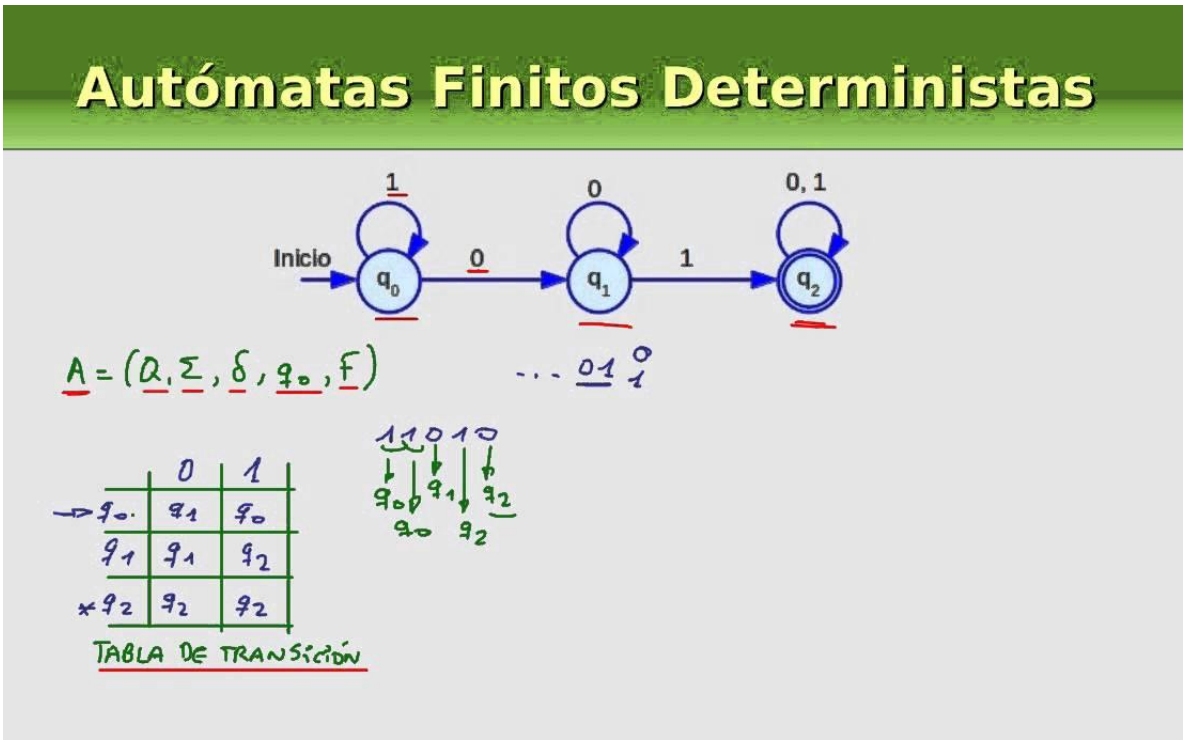
El analizador léxico o lexicográfico (Scanner en inglés) es la primera etapa del proceso de compilación, el cual se encarga de dividir el programa en Tokens, los cuales, según una tabla de símbolos definida por el mismo lenguaje.
De esta forma cada token del programa es clasificado según su significado para ser procesados en la segunda etapa del proceso de compilación

Segunda semana de videos

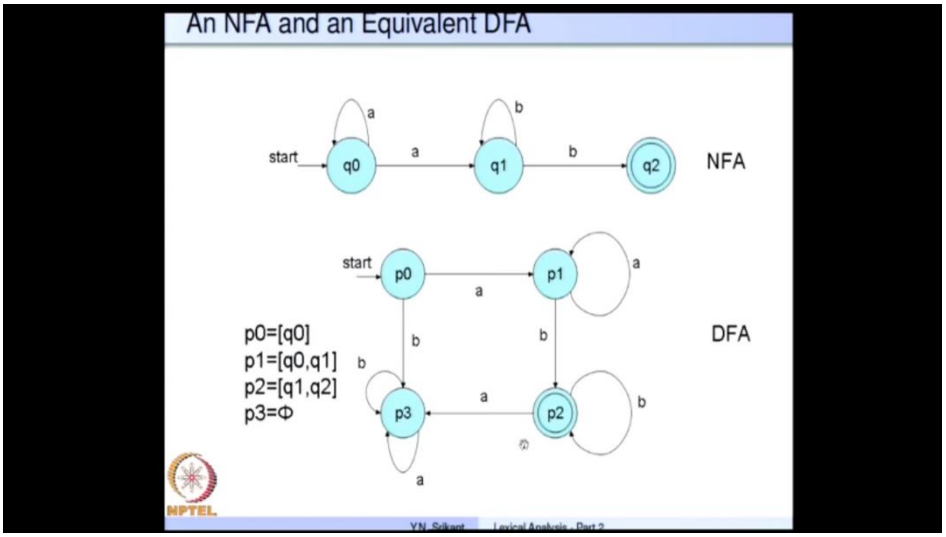
Mod-02 Lec-03 Lexical Analysis - Part 2

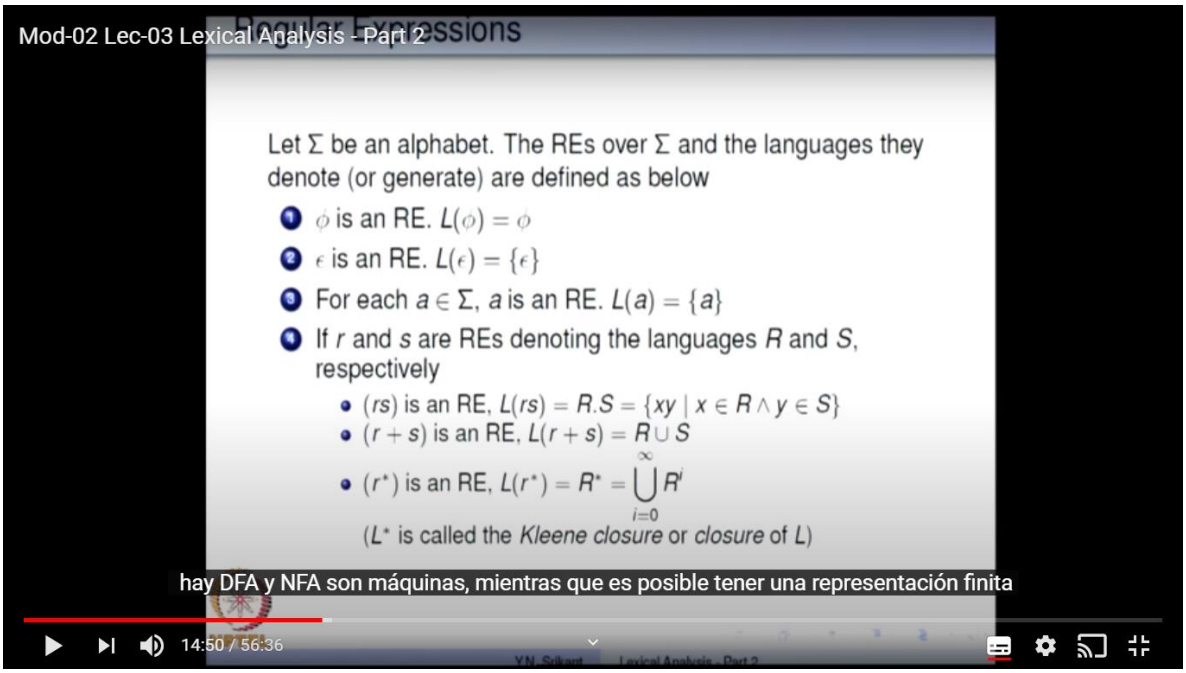
autómata determinista

un autómata determinista tiene una transición en cada símbolo alfabético la tabla de transiciones el cual describe los autómatas.



En esta imagen tenemos un ejemplo de un autómata determinista y no determinista.
en la primer autómata tenemos el no determinista a donde tiene dos transiciones con el mismo valor en el segundo solo tiene uno


An NFA and an Equivalent DFA
start
q0
q1
q2
NFA
start
p0
p1
p2
p3
p0=[q0]
p1=[q0,q1]
p2=[q1,q2]
p3=Φ
DFA
NPTEL


Mod-02 Lec-03 Lexical Analysis - Part 2
Regular Expressions
Let Σ be an alphabet. The REs over Σ and the languages they denote (or generate) are defined as below
1. ϕ is an RE. $L(\phi) = \phi$
2. ϵ is an RE. $L(\epsilon) = \{\epsilon\}$
3. For each $a \in \Sigma$, a is an RE. $L(a) = \{a\}$
4. If r and s are REs denoting the languages R and S, respectively
   - (rs) is an RE, $L(rs) = R.S = \{xy \mid x \in R \wedge y \in S\}$
   - $(r + s)$ is an RE, $L(r + s) = R \cup S$
   - (r^*) is an RE, $L(r^*) = R^* = \bigcup_{i=0}^{\infty} R^i$

(L^* is called the *Kleene closure* or *closure* of L)
hay DFA y NFA son máquinas, mientras que es posible tener una representación finita
14:50 / 56:36

lenguaje aceptado para o reconocido para DFA para tener una representación finita

En este texto vamos a ver uno de los métodos que se usan para transformar autómatas finitos deterministas en expresiones regulares, el método de eliminación de estados.

Cuando tenemos un autómata finito, determinista o no determinista, podemos considerar que los símbolos que componen a sus transiciones son expresiones regulares. Cuando eliminamos un estado, tenemos que reemplazar todos los caminos que pasaban a través de él como transiciones
directas que ahora se realizan con el ingreso de expresiones regulares, en vez de con símbolos.

Crear una FSA es muy parecido a escribir un programa. Es decir, es un proceso creativo sin una “receta” simple que se pueda seguir para llevarte siempre a un diseño correcto. Dicho esto, muchas de las habilidades que aplique a la programación también funcionarán al crear FSA.

Mod-02 Lec-04 Lexical Analysis - Part 3

Un NFA puede tener cero, uno o más de un movimiento de un estado dado en un símbolo de entrada dado. Un NFA también puede tener movimientos NULL (movimientos sin símbolo de entrada). Por otro lado, DFA tiene un solo movimiento desde un estado dado en un símbolo de entrada dado.

Conversión de NFA a DFA

Suponga que hay un NFA N <Q, ∑, q0, δ, F> que reconoce un lenguaje L.Entonces el DFA D <Q ', ∑, q0, δ', F '> se puede construir para idioma L como:

Paso 1: Inicialmente Q '= ϕ.

Paso 2: agregue q0 a Q '.

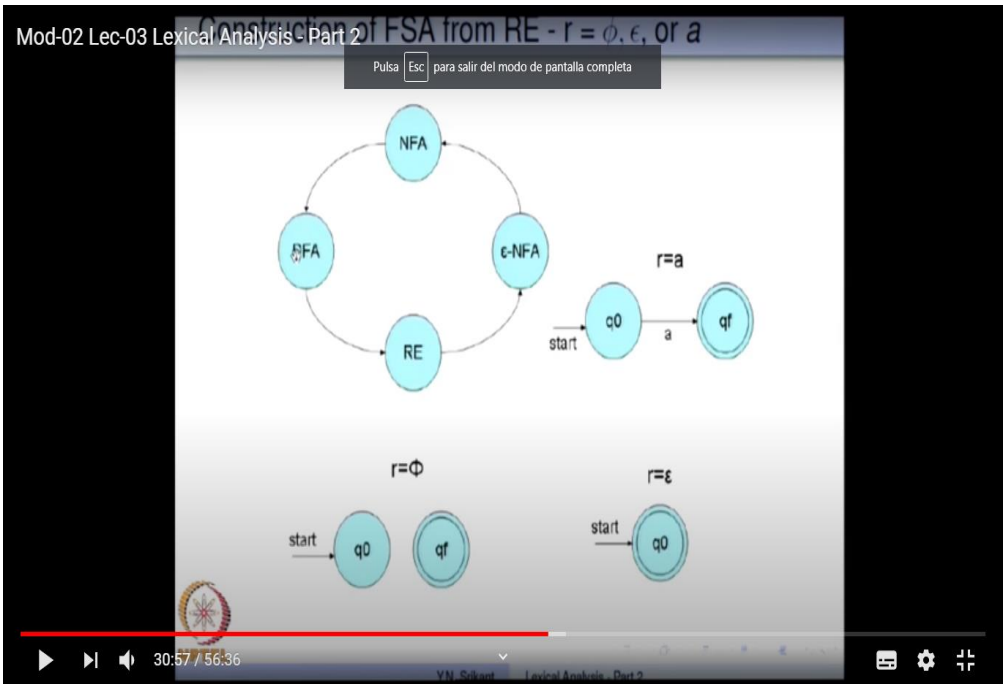
Paso 3: Para cada estado en Q ', encuentre el posible conjunto de estados para cada símbolo de entrada usando la función de transición de NFA. Si este conjunto de estados no está en Q ', agréguese a Q'.

Paso 4: El estado final de DFA serán todos los estados que contengan F (estados finales de NFA)

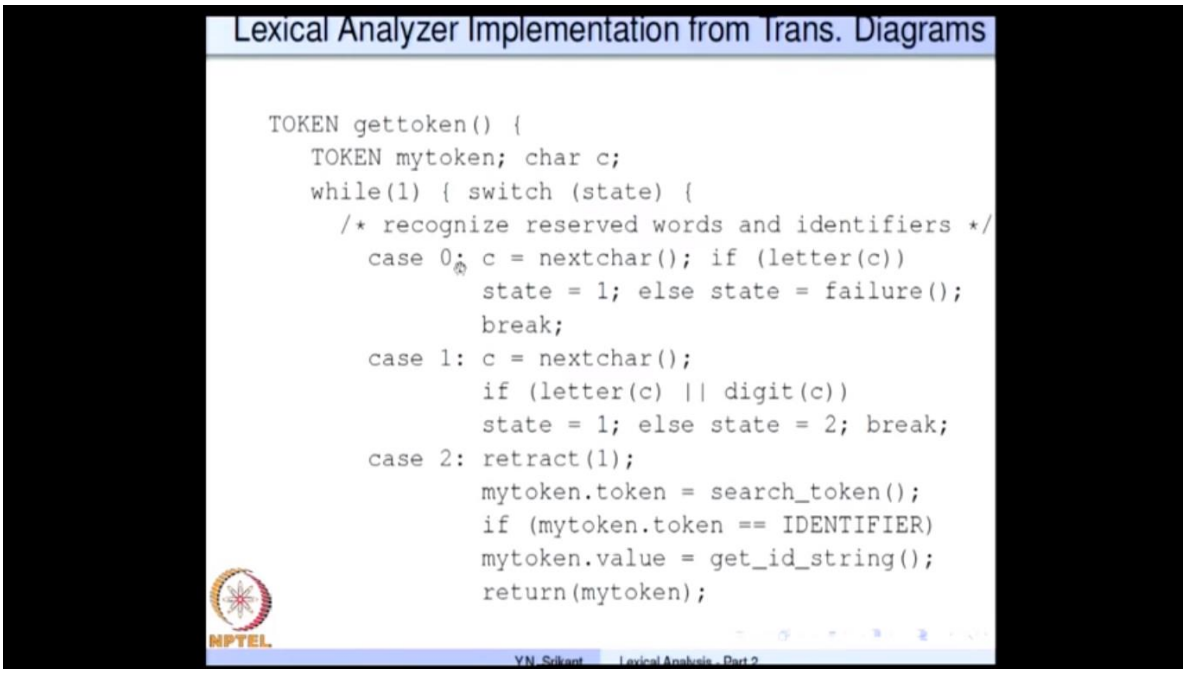
Equivalencia de FSA

El algoritmo de llenado de tabla nos proporciona una forma fácil de comprobar si dos lenguajes regulares son el mismo. Supongamos que tenemos los lenguajes L y M, cada uno de ellos representado de una manera, por ejemplo, uno mediante una expresión regular y el otro mediante un AFN. Convertimos cada una de las representaciones a un AFD. Ahora, imaginemos un AFD cuyos estados sean la unión de los estados de los AFD correspondientes a L y M. Técnicamente, este AFD tendrá dos estados iniciales, pero realmente el estado inicial es irrelevante para la cuestión de comprobar la equivalencia de estados, por lo que consideraremos uno de ellos como único estado inicial

Ahora se comprueba si los estados iniciales de los dos AFD originales son equivalentes, utilizando el algoritmo de llenado de tabla. Si son equivalentes, entonces L=M, y si no lo son, entonces L≠M.



Representación del analizador léxico basándose en los anteriores autómatas.

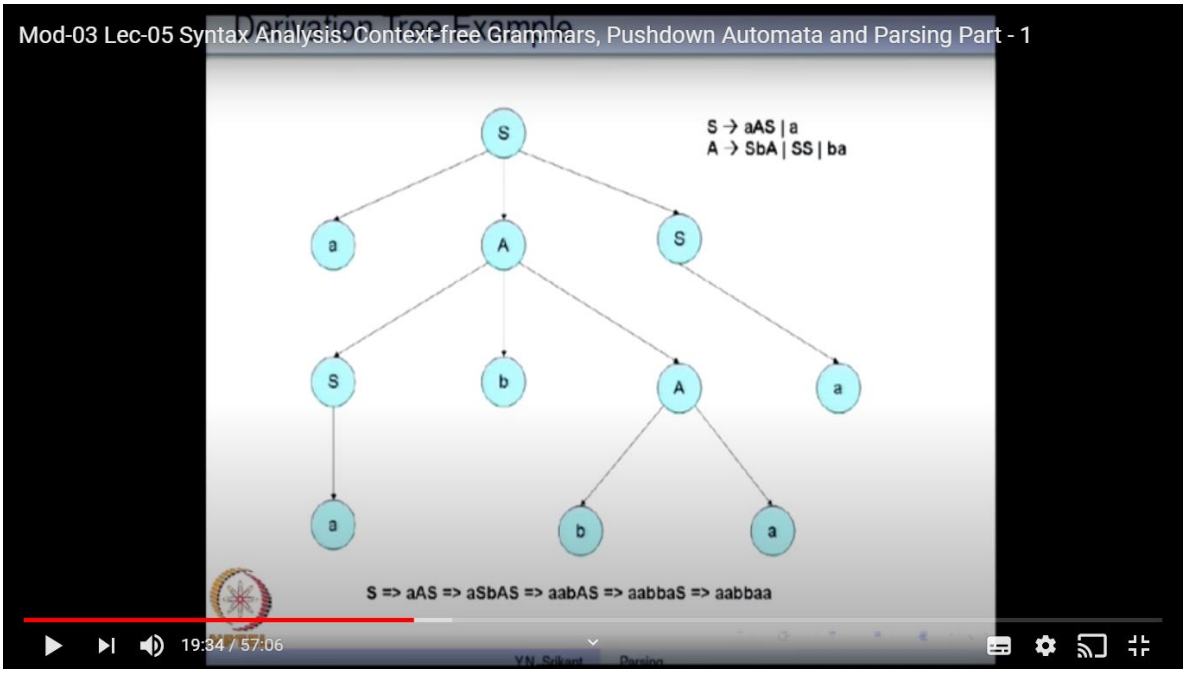


Mod-03 Lec-05 Syntax Analysis

gramáticas libres de contexto que son la base para la especificación de la programación idiomas que se analizan sin contexto, se describen con precisión y se utiliza una gramática para describir la sintaxis de los lenguajes programación se describen la estructura sintáctica para su ejecución

Las gramáticas se usan generalmente para la especificación de sintaxis de los lenguajes de programación. Durante el analisis lexico idiomas regulares, lenguajes sensibles al contexto en lenguajes de tipo 0. Utilizados para especificar lenguajes libres de contexto y estos son los más utilizados para un análisis sintáctico.

Gramática de contexto



• Las Gramáticas Libres de Contexto (´ Context-Free Languages) o CFL's jugaron un papel central en lenguaje natural desde los 50's y en los compiladores desde los 60's

• Las Gramáticas Libres de Contexto forman la base de ´ la sintaxis BNF

´ • Son actualmente importantes para XML y sus DTD's (document type definition)

Vamos a ver los CFG's, los lenguajes que generan, los árboles de parseo, el ´ pushdown automata y las propiedades de cerradura de los CFL's.

• Ejemplo: Considere Lpal = {w ∈ Σ ∗ : w = w R}. Por ejemplo, oso ∈ Lpal, anitalavalatina ∈ Lpal,

• Sea Σ = {0, 1} y supongamos que Lpal es regular.

• Sea dada por el pumping lemma. Entonces 0 n 10 ∈ Lpal. Al leer 0n el FA debe de entrar a un ciclo.

Ambigüedad. Árboles de derivación. Gramáticas ambiguas

Def (árbol de derivación) ***Dada una GI*** (Σ, V, I, P) ***, un árbol de derivación tiene las siguientes características:***

1. **Cada nodo interno está etiquetado con una variable** $\in V$ **.**
2. **Cada hoja está etiquetada con una variable** $\in V$ **, con un terminal** $\in \Sigma$ **ó con** λ **.**
3. **Si un nodo está etiquetado por** A **, y sus hijos por** $x_1, x_2, \ldots, x_n$ **,** **entonces debe existir una regla en P donde** $A \rightarrow x_1, x_2, \ldots x_n$ **.**

Ejemplo

Árbol de derivación para $a * 0 + b$

$$\Sigma = \{a, b, 0, 1, +, *, -, /, (,)\}$$

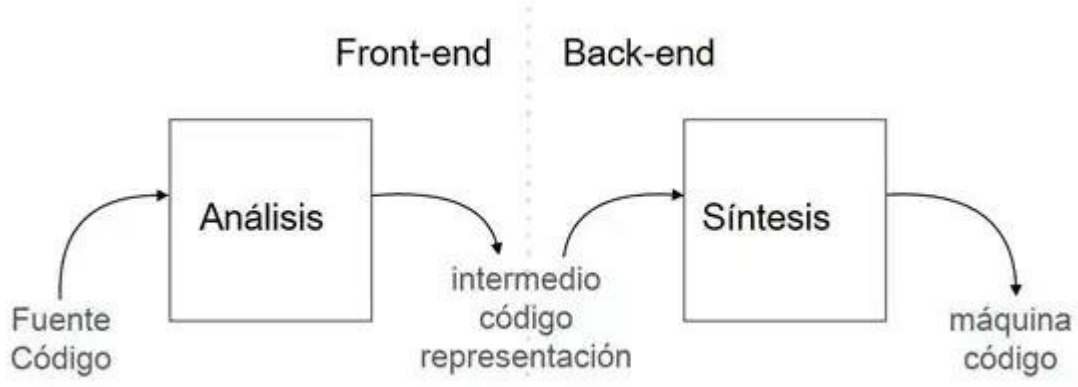
$$E \rightarrow E + E \mid E * E \mid E - E \mid E/E \mid (E) \mid I \mid N$$

$$N \rightarrow 0 \mid 1 \mid N0 \mid N1$$

$$I \rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1$$

Secciones del Front-End (Análisis Léxico, Sintáctico y Semántico)

Los compiladores básicamente se dividen en dos partes, **siendo la primera de ellas el "Front End",** y que es la parte del compilador encargada de analizar y comprobar la validez del código fuente y en base a ella crear los valores de la tabla de símbolos. Esta parte generalmente es independiente del sistema operativo para el cual se está compilando un programa.

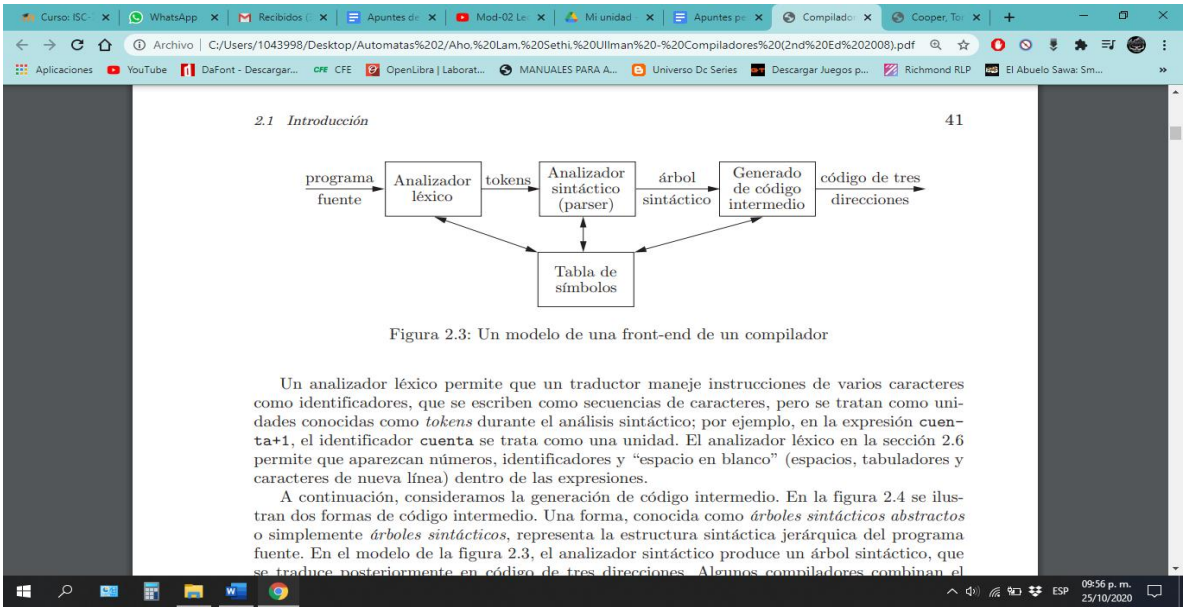


La segunda parte del compilador es la llamada "Back End", parte en la cual es generado el código máquina, el cual es creado, de acuerdo a lo analizado en el "Front End", para una plataforma específica, que puede ser Windows, Mac, Linux o cualquier otra.
Cabe destacar que los resultados obtenidos por el "Back End" no pueden ser ejecutados en forma directa, para ello es necesario la utilización de un proceso de enlazado, llamado "Linker", el cual básicamente es un software que recoge los objetos generados en la primera instancia de compilación, incluyendo la información de las bibliotecas, los depura y luego enlaza el código objeto con sus respectivas biblioteca, para finalmente crear un archivo ejecutable.

Libro

La fase de análisis de un compilador descompone un programa fuente en piezas componentes y produce una representación interna, a la cual se le conoce como código intermedio. La fase de síntesis traduce el código intermedio en el programa destino.

Por cuestión de simplicidad, consideramos la traducción orientada a la sintaxis de las expresiones infijas al formato postfijo, una notación en la cual los operadores aparecen después de sus operaciones. Por ejemplo, el formato postfijo de la expresión 9 − 5 + 2 es 95 − 2 +. La traducción al formato postfijo es lo bastante completa como para ilustrar el análisis sintáctico, y a la vez lo bastante simple como para que se pueda mostrar el traductor por completo en la sección 2.5. El traductor simple maneja expresiones como 9 − 5 + 2, que consisten en dígitos separados por signos positivos y negativos. Una razón para empezar con dichas expresiones simples es que el analizador sintáctico puede trabajar directamente con los caracteres individuales para los operadores y los operandos



Un analizador léxico permite que un traductor maneje instrucciones de varios caracteres como identificadores, que se escriben como secuencias de caracteres, pero se tratan como unidades conocidas como tokens durante el análisis sintáctico; por ejemplo, en la expresión cuen− ta+1, el identificador cuenta se trata como una unidad. El analizador léxico en la sección 2.6 permite que aparezcan números, identificadores y "espacio en blanco" (espacios, tabuladores y caracteres de nueva línea) dentro de las expresiones

Definición de sintaxis

una instrucción if-else es la concatenación de la palabra clave if, un paréntesis abierto, una expresión, un paréntesis cerrado, una instrucción, la palabra clave else y otra instrucción. Mediante el uso de la variable expr para denotar una expresión y la variable instr para denotar una instrucción, esta regla de estructuración puede expresarse de la siguiente manera: instr → if (expr) instr else instr en donde la flecha se lee como "puede tener la forma". A dicha regla se le llama producción. En una producción, los elementos léxicos como la palabra clave if y los paréntesis se llaman terminales. Las variables como expr e instr representan secuencias de terminales, y se llaman no terminales

Comparación entre tokens y terminales En un compilador, el analizador léxico lee los caracteres del programa fuente, los agrupa en unidades con significado léxico llamadas lexemas, y produce como salida tokens que representan estos lexemas. Un token consiste en dos componentes, el nombre del token y un valor de atributo

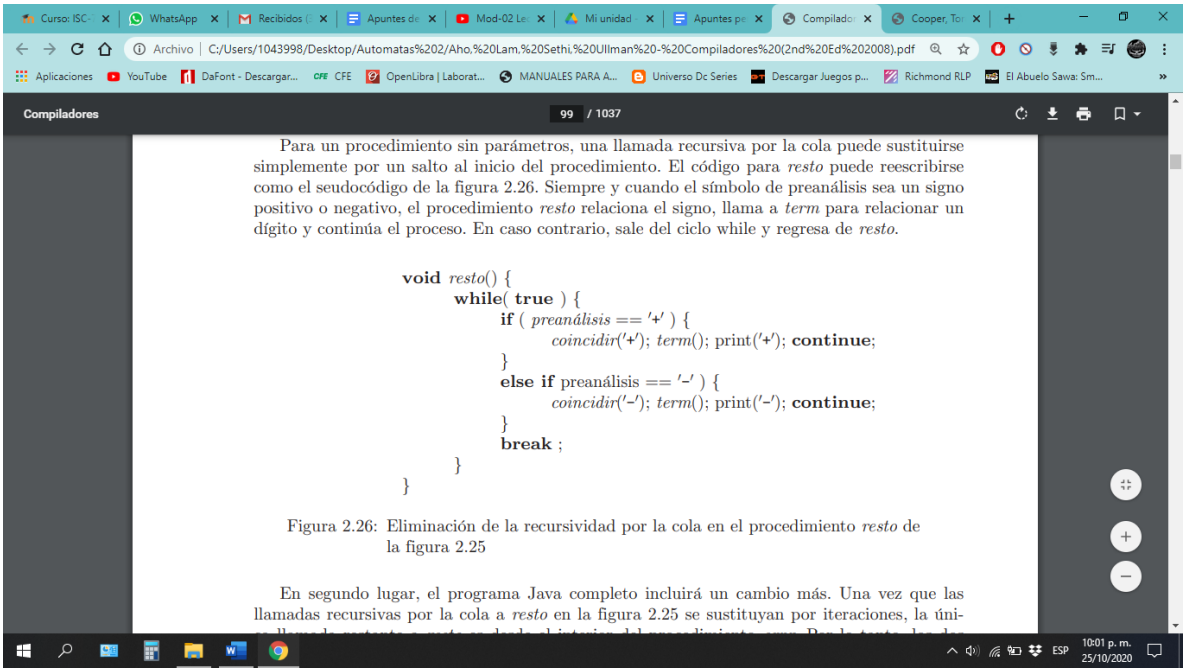
Análisis sintáctico El análisis sintáctico (parsing) es el proceso de determinar cómo puede generarse una cadena de terminales mediante una gramática. Al hablar sobre este problema, es más útil pensar en que se va a construir un árbol de análisis sintáctico, aun cuando un compilador tal vez no lo construya en la práctica. No obstante, un analizador sintáctico debe ser capaz de construir el árbol en principio, o de lo contrario no se puede garantizar que la traducción sea correcta

Recursividad a la izquierda Es posible que un analizador sintáctico de descenso recursivo entre en un ciclo infinito. Se produce un problema con las producciones “recursivas por la izquierda” como expr → expr + term

2.5.2 Adaptación del esquema de traducción La técnica de eliminación de recursividad por la izquierda trazada en la figura 2.20 también se puede aplicar a las producciones que contengan acciones semánticas. En primer lugar, la técnica se extiende a varias producciones para A. En nuestro ejemplo, A es expr y hay dos producciones recursivas a la izquierda para expr, y una que no es recursiva. La técnica transforma las producciones A → Aα | Aβ |

Simplificación del traductor

En primer lugar, ciertas llamadas recursivas pueden sustituirse por iteraciones. Cuando la última instrucción que se ejecuta en el cuerpo de un procedimiento es una llamada recursiva al mismo procedimiento, se dice que la llamada es recursiva por la cola. Por ejemplo, en la función resto, las llamadas de resto() con los símbolos de preanálisis + y − son recursivas por la cola, ya que en cada una de estas bifurcaciones, la llamada recursiva a resto es la última instrucción ejecutada por esa llamada de resto.



Compiladores 99 / 1037

Para un procedimiento sin parámetros, una llamada recursiva por la cola puede sustituirse simplemente por un salto al inicio del procedimiento. El código para *resto* puede reescribirse como el seudocódigo de la figura 2.26. Siempre y cuando el símbolo de preanálisis sea un signo positivo o negativo, el procedimiento *resto* relaciona el signo, llama a *term* para relacionar un dígito y continúa el proceso. En caso contrario, sale del ciclo while y regresa de *resto*.

```
void resto() {
    while( true ) {
        if ( preanálisis == '+' ) {
            coincidir('+'); term(); print('+'); continue;
        }
        else if preanálisis == '-' ) {
            coincidir('-'); term(); print('-'); continue;
        }
        break ;
    }
}
```

Figura 2.26: Eliminación de la recursividad por la cola en el procedimiento *resto* de la figura 2.25

En segundo lugar, el programa Java completo incluirá un cambio más. Una vez que las llamadas recursivas por la cola a *resto* en la figura 2.25 se sustituyan por iteraciones, la úni-

Análisis léxico Un analizador léxico lee los caracteres de la entrada y los agrupa en “objetos token”. Junto con un símbolo de terminal que se utiliza para las decisiones de análisis sintáctico, un objeto token lleva información adicional en forma de valores de atributos. Hasta ahora, no hemos tenido la necesidad de diferenciar entre los términos “token” y “terminal”, ya que el analizador sintáctico ignora los valores de los atributos q

Reconocimiento de palabras clave e identificadores

La mayoría de los lenguajes utilizan cadenas de caracteres fijas tales como for, do e if, como signos de puntuación o para identificar las construcciones. A dichas cadenas de caracteres se les conoce como palabras clave. Las cadenas de caracteres también se utilizan como identificadores para nombrar variables, arreglos, funciones y demás. Las gramáticas tratan de manera rutinaria a los identificadores como terminales para simplificar el analizador sintáctico, el cual por consiguiente puede esperar el mismo terminal, por decir id, cada vez que aparece algún identificador en la entrada. Por ejemplo, en la siguiente entrada: cuenta = cuenta + incrementó; (2.6) el analizador trabaja con el flujo de terminales id = id + id. El token para id tiene un atributo que contiene el lexema. Si escribimos los tokens como n-uplas, podemos ver que las n-uplas para el flujo de entrada (2.6) son: id, "cuenta" = id, "cuenta" + id, "incremento" ;

Un analizador léxico

Hasta ahora en esta sección, los fragmentos de pseudocódigo se juntan para formar una función llamada escanear que devuelve objetos token, de la siguiente manera: Token escanear () { omitir espacio en blanco, como en la sección 2.6.1; manejar los números, como en la sección 2.6.3; manejar las palabras reservadas e identificadores, como en la sección 2.6.4; /* Si llegamos aquí, tratar el carácter de lectura de preanálisis vistazo como token */ Token t = new Token(vistazo); vistazo = espacio en blanco /* inicialización, como vimos en la sección 2.6.2 */; return t; }

Las técnicas orientadas a la sintaxis que vimos en este capítulo pueden usarse para construir interfaces de usuario (front-end) de compiladores

El punto inicial para un traductor orientado a la sintaxis es una gramática para el lenguaje fuente. Una gramática describe la estructura jerárquica de los programas. Se define en términos de símbolos elementales, conocidos como terminales, y de símbolos variables llamados no terminales. Estos símbolos representan construcciones del lenguaje. Las reglas o producciones de una gramática consisten en un no terminal conocido como el encabezado o lado izquierdo de una producción, y de una secuencia de terminales y no terminales, conocida como el cuerpo o lado derecho de la producción. Un no terminal se designa como el símbolo inicial. Al especificar un traductor, es conveniente adjuntar atributos a la construcción de programación, en donde un atributo es cualquier cantidad asociada con una construcción. Como las construcciones se representan mediante símbolos de la gramática, el concepto de los atributos se extiende a los símbolos de la gramática. Algunos ejemplos de atributos incluyen un valor entero asociado con un terminal num que representa números, y una cadena asociada con un terminal id que representa identificadores.

Análisis de arriba hacia abajo usando gramáticas LL(Oct 26-30, 2020)

El análisis de arriba hacia abajo mediante el análisis predictivo, rastrea la derivación más a la izquierda de la cadena mientras se construye el árbol de análisis.

Comienza desde el símbolo de inicio de la gramática y "predice" la siguiente producción utilizada en la derivación.

Dicha "predicción" es ayudada por tablas de análisis (construidas fuera de línea).

La siguiente producción que se utilizará en la derivación se determina utilizando el siguiente símbolo de entrada para buscar la tabla de análisis (símbolo de anticipación).

Poner restricciones en la gramática asegura que ningún espacio en la tabla de análisis contenga más de una producción.

En el momento de la construcción de la tabla de análisis, si dos producciones se vuelven elegibles para colocarse en el mismo espacio de la tabla de análisis, la gramática se declara no apta para predicciones.

Gramáticas fuertes de LL (k)

Sea la gramática dada G.

La entrada se amplía con k símbolos, $k, k es el avance de la gramática.

Introducir un nuevo no terminal S', y una producción, S'S$k, donde S es el símbolo de inicio de la gramática dada.

Considere solo las derivaciones más a la izquierda y asuma que la gramática no tiene símbolos inútiles.

Una producción A en G se llama fuerte LL(k) producción, si en G.
S'* wAw* wzy
S'* w'Aw'* w'zy
|Z|=k, z *, w y w' *, entonces =

Una gramática (no terminal) es fuerte LL(k) si todas sus producciones son fuertes LL(k).

Fuerte LL(k) Las gramáticas no permiten que se utilicen diferentes producciones del mismo no terminal incluso en dos derivaciones diferentes, si los primeros k símbolos de las cadenas producidas por y son las mismas.
Construcción de la tabla de análisis sintáctico LL (1)
para cada producción A
para cada símbolo s dirsymb ()
/* s puede ser un símbolo de terminal o $ */
add A to LLPT [A.s]
Hacer que cada entrada indefinida de LLPT sea un error

para cada producción A
para cada símbolo de terminal a first ()
add A to LLPT [A, a]
if first (){
para cada símbolo de terminal b follow (A)
add A to LLPT [A, b]

if $ follow (A)
add A to LLPT [A, $]
}
Hacer que cada entrada indefinida de LLPT sea un error

- Una vez completada la construcción de la tabla LL (1) (siguiendo cualquiera de los dos métodos), si algún espacio en la tabla LL (1) tiene dos o más producciones, entonces la gramática NO es LL (1)

Análisis de Descenso Recursivo

Estrategia de análisis de arriba hacia abajo.

Una función / procedimiento para cada no terminal.

Las funciones se llaman entre sí de forma recursiva, según la gramática.

La pila de recursividad maneja las tareas de la pila del analizador LL (1).

LL (1) condiciones que deben cumplirse para la gramática.

Se puede generar automáticamente a partir de la gramática.

La codificación manual también es fácil.

La recuperación de errores es superior.

Análisis de abajo hacia arriba

Comience en las hojas, construya el árbol de análisis en segmentos pequeños, combine los árboles pequeños para hacer árboles más grandes, hasta que se alcance la raíz.

Este proceso se llama reducción de la oración al símbolo inicial de la gramática.

Una de las formas de "reducir" una oración es seguir la derivación más a la derecha de la oración al revés.

- El análisis sintáctico Shift-Reduce implementa dicha estrategia.

- Utiliza el concepto de asa para detectar cuándo realizar reducciones.

Análisis de Mayús-Reducir

Mango: Mango de una forma de oración correcta , es una producción Ay un puesto en , donde la cadena puede ser encontrado y reemplazado por A, para producir la forma oracional derecha anterior en una derivación más a la derecha de .
Es decir, si S*rm Awrm w, entonces A en la posición siguiente es un mango de w.

Un asa siempre aparecerá en la parte superior de la pila, nunca sumergida dentro de la pila.

En el análisis sintáctico S-R, ubicamos el mango y lo reducimos por el LHS de la producción repetidamente, para llegar al símbolo de inicio.

Estas reducciones, de hecho, trazan una derivación más a la derecha de la oración a la inversa. Este proceso se llama poda de mango.

LR-Parsing es un método de análisis sintáctico con reducción de cambios.

Actividades semana 8 nov 9-13, 20

Análisis Semántico

La consistencia semántica que no puede ser manejada en la etapa de análisis se maneja aquí.

Los analizadores no pueden manejar las características sensibles al contexto de los lenguajes de programación.

Son semánticas estáticas de los lenguajes de programación y pueden ser comprobadas por el analizador semántico.

- Las variables se declaran antes de su uso.
- Los tipos coinciden en ambos lados de las asignaciones.
- Los tipos de parámetros y el número coinciden en la declaración y el uso.

Los compiladores sólo pueden generar código para comprobar la semántica dinámica de los lenguajes de programación en tiempo de ejecución.

- Si se produce un desbordamiento durante una operación aritmética.
- Si los límites de la matriz se cruzaran durante la ejecución.
- Si la recursividad cruzara los límites de la pila.
- Si la memoria del cabezal es insuficiente.

La información de los tipos se almacena en la tabla de símbolos o en el árbol de sintaxis.

- Tipos de variables, parámetros de función, dimensiones de la matriz, etc.
- Se utiliza no sólo para la validación semántica sino también para las fases posteriores de la compilación.

La semántica estática de PL puede especificarse utilizando gramáticas de atributos.

Los analizadores semánticos pueden generarse semi automáticamente a partir de atributos.

Las gramáticas de atributos son extensiones de las gramáticas sin contexto.

Semántica Estática

Muestras de comprobaciones semánticas estáticas en principal.

- Los tipos de p y el tipo de retorno de dot_prod coinciden.
- El número y el tipo de los parámetros de dot_prod son los mismos tanto en su declaración como en su uso.
- p es declarado antes de su uso, igual para a y b.

.

Muestras de comprobaciones semánticas estáticas en dot_prod.

- d e i se declaran antes de su uso.
- El tipo de d coincide con el tipo de retorno de dot_prod.
- El tipo de d coincide con el tipo de resultado de "*".
- Los elementos de las matrices x e y son compatibles con "*".

Semántica Dinámica

Muestras de comprobaciones semánticas dinámicas en dot_prod.

- El valor de i no excede el rango declarado de las matrices x e y (tanto inferior como superior).

- No hay desbordamientos durante las operaciones de “*” y “+” en d += x[i] *y [i].

Muestras de comprobaciones semánticas dinámicas en dot_prod.

- El valor de i no excede el rango declarado de las matrices x e y (tanto inferior como superior).

- No hay desbordamientos durante las operaciones de “*” y “+” en d += x[i] *y [i].

Muestras de comprobaciones semánticas dinámicas de hecho.

- La comprobación del programa no se desborda debido a la recursividad.

- No hay desbordamiento debido a “*” en n *fact (n-1).

Gramáticas de Atributos

Dejemos que G = (N.T.P.S) sea un CFG y dejemos V=NT.

Cada símbolo X de V tiene asociado un conjunto de atributos (denotados por X.a. X.b, etc.).

Dos tipos de atributos: heredados (denotados por Al(X)) y sintetizados (denotados por AS(X)).

Cada atributo toma valores de un dominio especificado (finito o infinito), que es su tipo.

- Los dominios típicos de los atributos son, enteros, reales, caracteres, cadenas, booleanos, estructuras, etc.

- Se pueden construir nuevos dominios a partir de dominios dados mediante operaciones matemáticas como producto cruzado, mapa, etc.

- formación: un mapa, ND, dónde, N y D son los dominios de los números naturales y los objetos dados, respectivamente.

- estructura: un producto cruzado, A1A2...An, dónde n es el número de campos en la estructura, y Ai es el dominio del campoith.

Gráfico de Dependencia de Atributos

- Que T sea un árbol de análisis generado por el CFG de un AG, G.

- El gráfico de dependencia de atributos (gráfico de dependencia para abreviar) para T es el gráfico dirigido, DG (T) = (V.E), donde

  V = {b | b es una instancia de atributo de algún nodo del árbol}, y

  E={(b, c)|b, c V, b y c son atributos de símbolos gramaticales en la misma producción p de B, y el valor de b se utiliza para calcular el valor de c en una regla de cálculo de atributos asociada a la producción p

Estrategia de Evaluación de Atributos

- Construye el árbol de análisis.

- Construye el gráfico de dependencia.

- Realizar una clasificación topológica en el gráfico de dependencia y obtener un orden de evaluación.

- Evaluar los atributos según este orden utilizando las reglas de evaluación de atributos correspondientes adjuntas a las producciones respectivas.

- Múltiples atributos en un nodo del árbol de análisis pueden hacer que ese nodo sea visitado varias veces.

    - Cada visita resulta en la evaluación de por lo menos un atributo.

Gramáticas Atribuidas por la L y la S

- Un AG con sólo atributos sintetizados en una gramática de atributos S.

    - Los atributos de los SAG pueden ser evaluados en cualquier orden de botones sobre un árbol de análisis (de una sola pasada).

    - La evaluación de los atributos puede ser combinada con el análisis de LR (YACC).

- En la gramática atribuida a la L, las dependencias atribuidas siempre van de izquierda a derecha.

- Más precisamente, cada atributo debe ser.

    - Sintetizada, o

    - Heredado, pero con las siguientes limitaciones: considerar una producción p:AX1X2...Xn. Let Xi.a Al(Xi). Xi.a sólo se puede utilizar.

        - elementos de Al(A).

- elementos de Al(Xk) or AS(Xk), k = 1,...,i - 1 (es decir, los atributos de X1,..., Xi-1)

- Nos concentramos en los SAG, y 1 - pasar los LAG, en los que la evaluación de atributos puede combinarse con el análisis de LR, LL o RD.

Mod-04 Lec-14 Análisis Semántico

Gramática de Traducción Atribuida

.
Aparte de las reglas de cálculo de atributos, se añade al AG algún segmento de programa que realiza el cálculo de salida o de algún otro árbol de efectos secundarios.

.
Ejemplos: operaciones de tablas de símbolos, escritura de código generado en un archivo, etc.

.
Como resultado de estos segmentos de código de acción, las órdenes de evaluación pueden verse restringidas.

.
Tales restricciones se añaden al gráfico de dependencia de atributos como bordes implícitos.

.
Estas acciones pueden ser añadidas tanto a los SAG como a los LAG (haciéndolos, SATG y LATG resp.).
.
Nuestra discusión sobre el análisis semántico utilizará LATG (1 - paso) y SATG.

LATG para Sem. Análisis de declaraciones variables - 2

La gramática no es LL(1) y por lo tanto no se puede construir un analizador LL(1) a partir de ella.

Asumimos que el árbol de análisis está disponible y que la evaluación de atributos se realiza sobre el árbol de análisis.

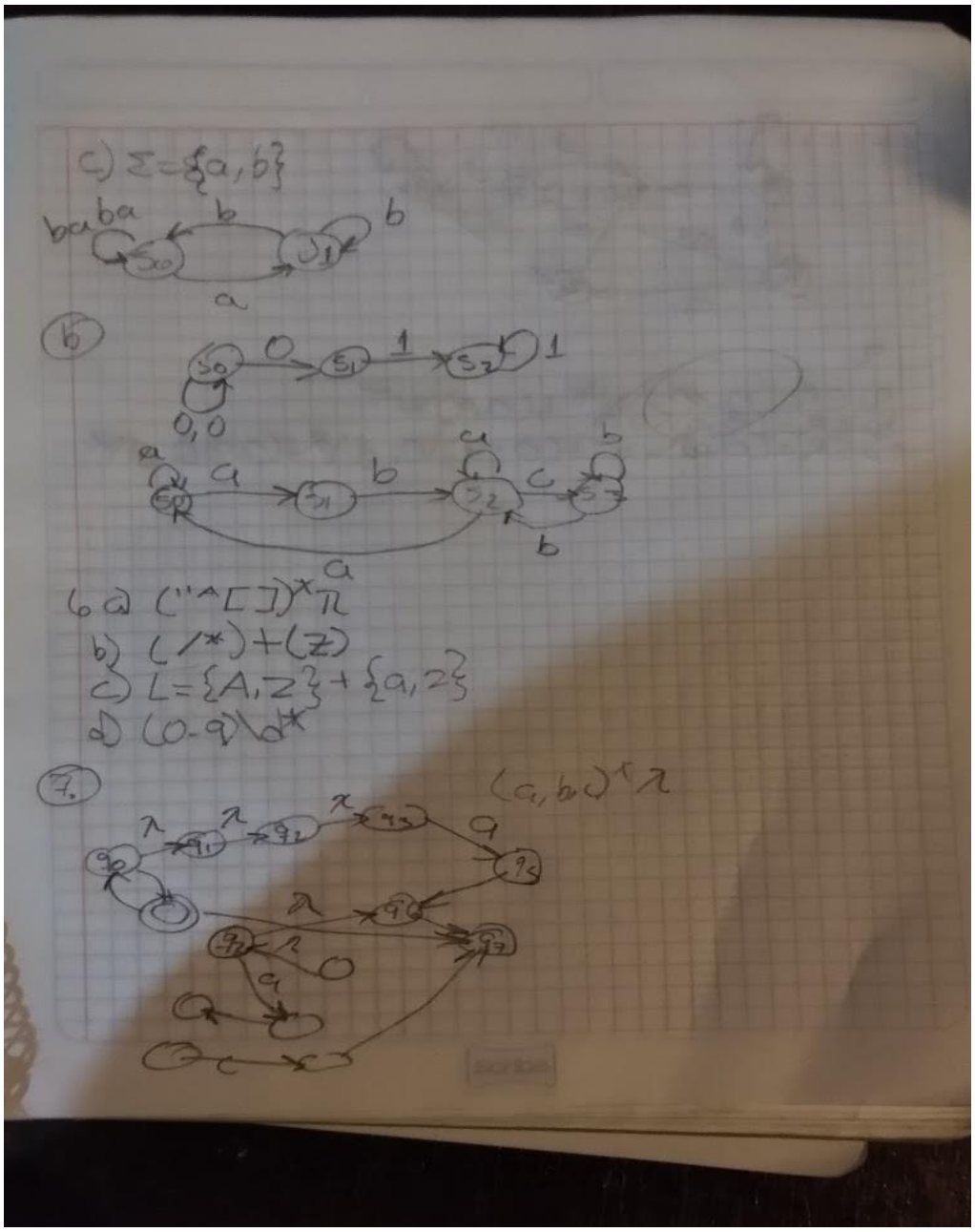
Las modificaciones del CFG para convertirlo en LL(1) y los correspondientes cambios en el AG se dejan como ejercicios.

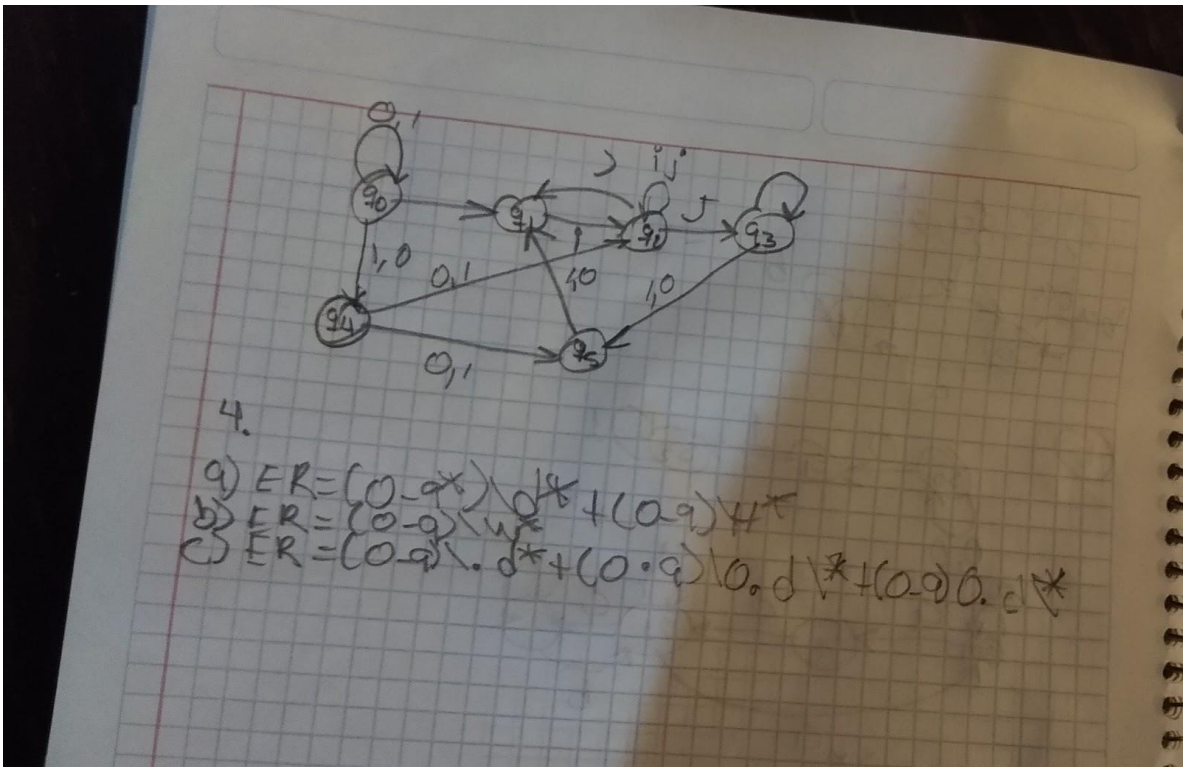
Los atributos y sus reglas de cálculo para las producciones 1 - 4 son como antes y los ignoramos.

Proporcionamos el AG sólo para las producciones 5 - 7; el AG para la regla 8 es similar al de la regla 7.

El manejo de las declaraciones constantes es similar al de las declaraciones variables.

Ejercicios del examen





4.

a) ER=(0-9)*1*+(0-9)1*

b) ER=(0-9)*1*

c) ER=(0-9)*.1*+(0-9)*0.1*+(0-9)0.1*