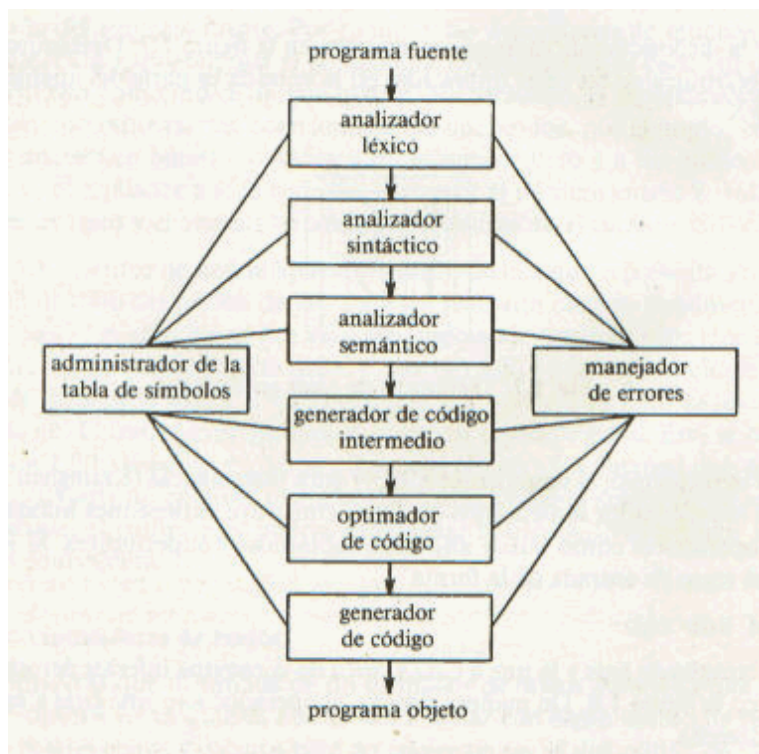


FASES DE UN COMPILADOR

Conceptualmente, un compilador opera *en fases*, cada una de las cuales transforma al programa fuente de una representación en otra. En la siguiente figura se muestra una descomposición típica de un compilador. En la práctica, se pueden agrupar algunas fases y las representaciones intermedias entre las fases agrupadas no necesitan ser construidas explícitamente.

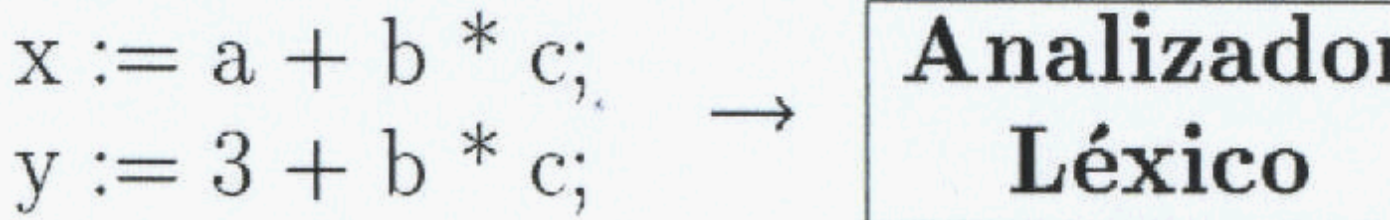


Como se pueden ver dos fases del compilador, la administración de la tabla de símbolos y el manejo de errores, se muestran en interacción con las seis fases de análisis léxico, análisis sintáctico, análisis semántico, generación de código intermedio, optimización de código y generación de código. De modo informal, también se llamarán "fases" al administrador de la tabla de símbolos y al manejador de errores.

FASE DE ANÁLISIS LÉXICO

En un compilador, el análisis lineal se llama *análisis lineal o exploración*. Por *análisis lineal* entendemos la cadena de caracteres que constituye el programa fuente se lee de izquierda a derecha y se agrupan en

componentes léxicos, que son secuencias de caracteres que tiene un significado colectivo. Por ejemplo, en el análisis léxico los caracteres de las siguientes proposiciones de asignación



```
x := a + b * c;  
y := 3 + b * c;
```

→ **Analizador
Léxico**

Los espacios en blanco (delimitadores) que separan los caracteres de estos componentes léxicos normalmente se eliminan durante el análisis léxico.

FASE DE ANÁLISIS SEMÁNTICO

La fase de análisis semántico revisa el programa fuente para tratar de encontrar errores semánticos y reúne la información sobre los tipos para la fase posterior de generación de código. En ella se utiliza la estructura jerárquica determinada por la fase de análisis sintáctico para identificar los operadores y operandos de expresiones y proposiciones.

Un componente importante del análisis semántico es la verificación de tipos. Aquí, el compilador verifica si cada operador tiene operandos permitidos por la especificación del lenguaje fuente. Por ejemplo, las definiciones de muchos lenguajes de programación requieren que el compilador indique un error cada vez que se use un número real como índice de una matriz. Sin embargo, la especificación del lenguaje puede permitir ciertas coerciones a los operandos, por ejemplo, cuando un operador aritmético binario se aplica a un número entero y a un número real. En este caso, el compilador puede necesitar convertir el número entero a real.

Ejemplo 1.1. Dentro de una máquina el patrón de bits que representa un entero es en general distinto del patrón de bits para un real, aun cuando el número entero y el real tengan el mismo valor. Por ejemplo, supóngase que todos los identificadores de la figura siguiente se han declarado reales y que tan solo 60 se supone entero. La verificación de tipos de la figura a revela que * se aplica a un real, velocidad, y a un entero, 60. El tratamiento general es convertir el entero a real. Esto se ha logrado en la figura b creando un nodo extra para el operador entereal que de manera explícita convierte un entero a real. Por otra parte, como el operando

de entereal es una constante, el compilador podría reemplazar la constante entera por una constante real equivalente.

EXPRESIÓN REGULAR

Una expresión regular denota un conjunto de secuencias de símbolos válidas que se construyen en base al alfabeto de un lenguaje. Generalmente estos conjuntos se representan como:



Es una expresión regular que denota el conjunto vacío (el conjunto no contiene secuencias de símbolos).



Es una expresión regular que denota al conjunto que contiene la secuencia vacía.



Una secuencia de símbolos S es una expresión regular que denota a un conjunto que contiene a S.

Operaciones sobre Lenguajes

Construir una expresión regular es realizar operaciones sobre el alfabeto de un lenguaje. Podemos decir que el lenguaje, en su aspecto de léxico, está conformado por las operaciones que realizamos sobre su alfabeto para definir cada unidad de léxico. Las operaciones posibles son:

Unión de L y M.

$$L \cup M = \{ S \mid S \text{ está en } L \text{ ó } S \text{ está en } M \}$$

Concatenación de L y M.

$$LM = \{ st \mid s \text{ está en } L \text{ y } t \text{ está en } M \}$$

Cerradura Kleene.

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

(L se repite de 0 a infinito)

Cerradura Positiva.

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

(L se repite de 1 a infinito).

Como se ve, las expresiones regulares son operaciones que se realizan sobre conjuntos de símbolos que pertenecen al alfabeto de un lenguaje. Estos conjuntos se describen colocando a los símbolos que pertenecen a ellos entre llaves. Por ejemplo, los caracteres que ubicamos como letras podrían conformar el conjunto

{ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz}

Si los elementos del conjunto tienen un orden conocido o asociado al código ASCII se puede describir colocando el primer símbolo y el último separados por un guión (–) como

{A–Za–z}

Para hacer más simple la construcción de una expresión regular, se designa bajo un nombre al conjunto con el cual se trabajará. así el conjunto letra se puede describir como

L = {A–Za–z}

Cuando a un conjunto o a una expresión regular se le da un nombre, ésta recibe el nombre de definición regular. Así, la expresión regular para un identificador será:

ID=L (L | D | G) *

si las definiciones sobre las cuales está construida son:

L= {A–Za–z}

D = {0123456789}

G = {_}

Por ejemplo, en Ada los identificadores están formados por letras, dígitos o guiones, pero el guión no puede ser el último caracter. La expresión regular con la cual se puede conformar el patrón para esta unidad de léxico puede ser:

ID_ada = L (L | D) * (G (L | D) +) *

Una expresión regular es un método formal para describir un patrón y puede ser empleada para construir un analizador léxico que lo reconozca en una computadora. De hecho, la expresión sufre una serie de transformaciones para llegar a ser explotada como tabla:

Expresión Regular
se convierte en
Autómata Finito Nodeterminístico (NFA)
se convierte en
Autómata Finito Determinístico (DFA)
que se representa como
Tabla

AUTÓMATAS FINITOS

Un autómata finito es un conjunto de nodos y aristas que representan trayectorias para generar una expresión bajo un alfabeto. Un diagrama de transición es un autómata finito. Los autómatas finitos se clasifican en:

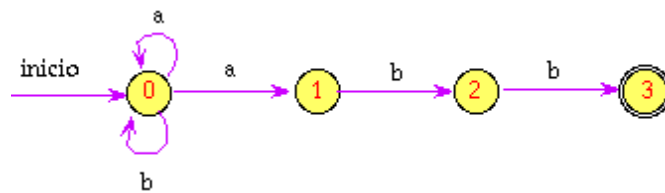
- ♦ Autómatas finitos no determinísticos. NFA
- ♦ Autómatas finitos determinísticos. DFA

Autómata Finito NoDeterminístico

Un NFA es un modelo matemático que consiste de:

- 1.– Un conjunto de estados.
- 2.– Un conjunto de símbolos de entrada.
- 3.– Una función de transición que corresponde pares estado–símbolo a conjuntos de estados.
- 4.– Un estado S_0 que denota como el estado inicial.
- 5.– Un conjunto de estados F que denotan los estados de aceptación o finales.

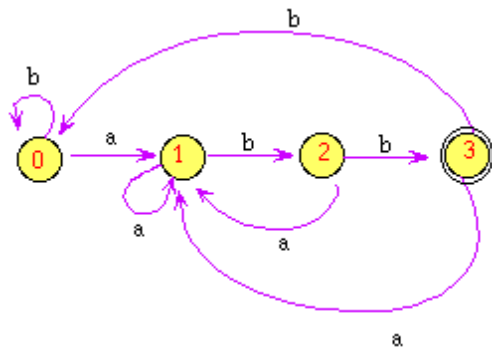
Un NFA no tiene restricciones para que exista más de una transición con el mismo nombre a diferentes estados, por lo que en una representación tabular no es posible determinar de manera única el estado destino para un símbolo determinado. Por ejemplo, el siguiente diagrama representa la expresión $(a \mid b)^* a b b$



no podemos determinar a qué estado se avanza del estado 0 con el símbolo a ; puede ser al propio 0 o al 1 pero ¿bajo qué condición? No está determinado. Incluso podría existir transiciones sin ninguna restricción, que se denominan transiciones epsilon porque no requieren un símbolo determinado ocurra para realizarse. Nuevamente, esto se vuelve indeterminado en una representación tabular.

Autómata Finito Determinístico

Un DFA es un caso especial de NFA en el que ningún estado tiene transiciones para diversos estados bajo el mismo símbolo; no se permiten transiciones epsilon. Los diagramas de transición son autómatas determinísticos. Por ejemplo, el DFA de $(a \mid b)^* a b b$ puede ser:



que podría ser muy aproximado al diagrama de transición que construiríamos para la expresión.

Para llegar de la expresión regular al autómata determinístico se emplea el método de Thompson que permite hacerla transformación y preparar la construcción del analizador de léxico.