Compiladores: Sesión 6. Optimización

Prof. Gloria Inés Alvarez V.

Departamento de Ciencias e Ingeniería de la Computación
Pontificia Universidad Javeriana Cali

7 de febrero de 2008

Optimización

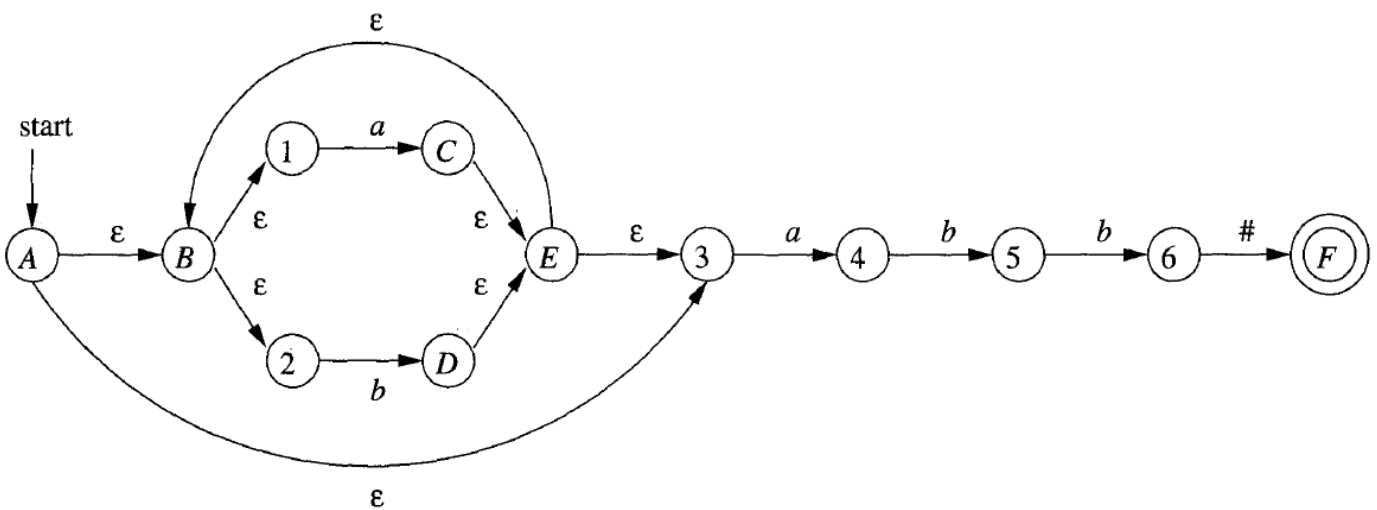
Se proponen tres maneras de optimizar el analizador léxico:

- Construir el DFA directamente a partir de la expresión regular, sin pasar por un NFA. El autómata resultante suele tener menos estados que el que se obtendría a partir de un NFA.
- Minimizar el DFA, es decir, eliminar estados mientras se sigue reconociendo el mismo lenguaje.
- Hacer una representación que permita un acceso más rápido a la tabla de transición del DFA.

Estados Importantes

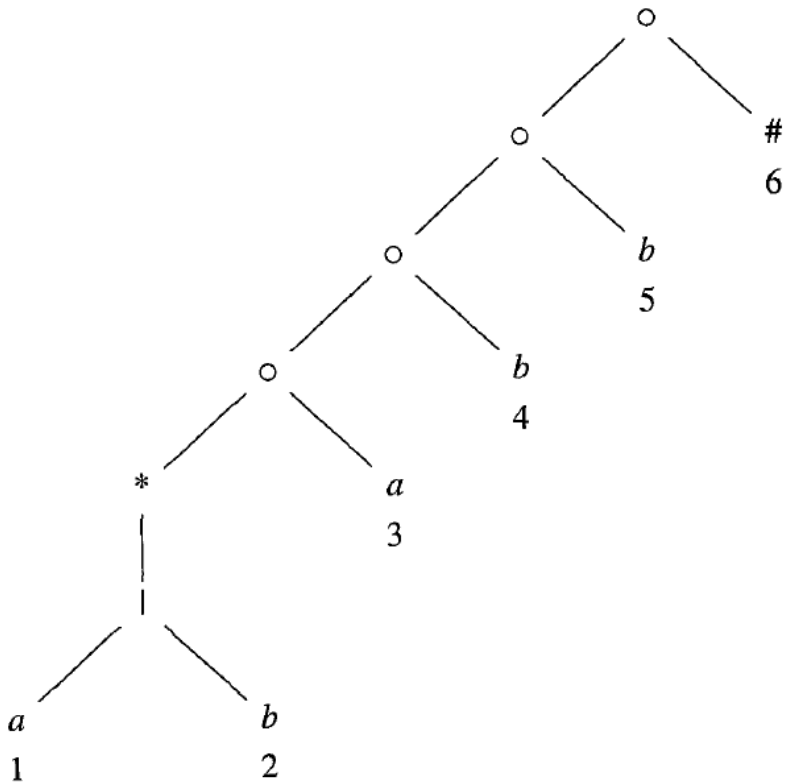
- Son aquellos estados de los que parten arcos etiquetados con símbolos diferentes a la cadena vacía.
- Corresponden a las hojas del árbol de sintaxis de la expresion regular que no contienen ε.
- Se adiciona un símbolo que no esté en el alfabeto para hacer que el estado final sea importante

Ejemplo para la expresión regular $(a + b) * abb\#$



Estados Importantes

Ejemplo de árbol de sintaxis asociado a la expresión regular $(a + b) * abb\#$



Estados Importantes

- Los valores enteros asociados a las hojas se llaman *posiciones* y permiten identificar los estados importantes.
- Dos estados del DFA generado por construcción de subconjuntos son iguales si contienen los mismos estados importantes y ambos o ninguno contienen estados finales del NFA.

Construcción del DFA a partir de la expresión regular

Definición

Si i es una posición, $followpos(i)$ es el conjunto de posiciones j tales que existe alguna cadena de entrada de la forma $\ldots cd \ldots$ donde i corresponde a c y j a d.

Ejemplo:
En el árbol de sintaxis anterior: $followpos(1) = \{1, 2, 3\}$

Definición

Si n es un estado en el árbol de sintaxis, $nullable(n)$ es verdadero si el lenguaje de la subexpresión con raíz en n contiene la cadena ε.

Construcción del DFA a partir de la expresión regular

Definición

Si n es un estado en el árbol de sintaxis, firstpos(n) es el conjunto de posiciones en el subárbol con raíz en n tales que existe alguna cadena del lenguaje asociado a la subexpresión con raíz en n que comienza por esa posición.

Definición

Si n es un estado en el árbol de sintaxis, lastpos(n) es el conjunto de posiciones en el subárbol con raíz en n tales que existe alguna cadena del lenguaje asociado a la subexpresión con raíz en n que termina por esa posición.

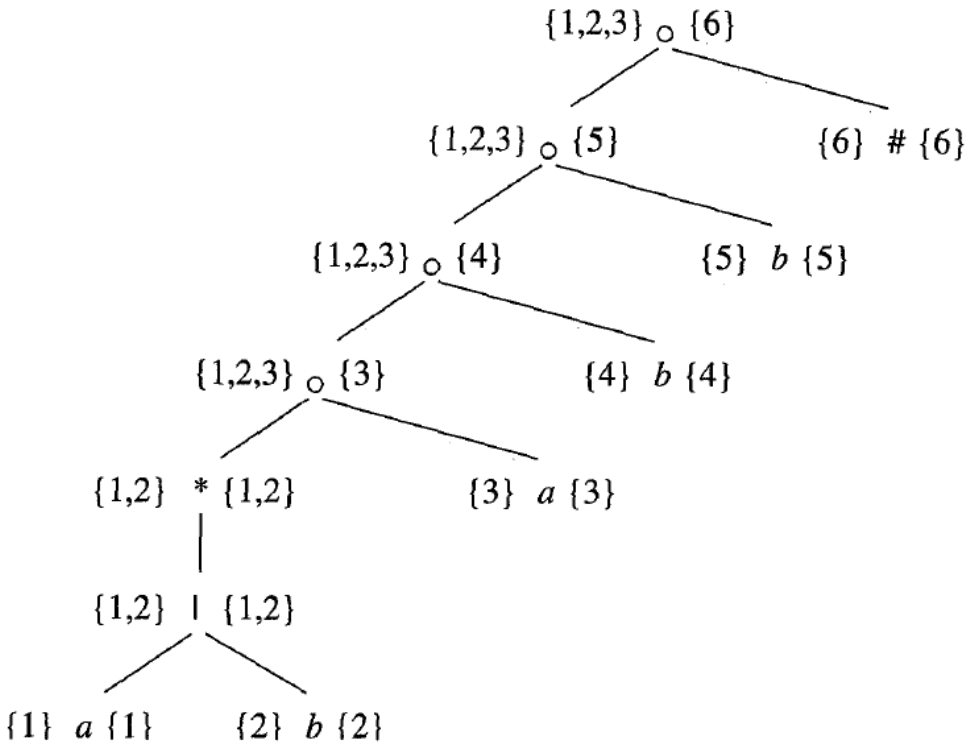
Construcción del DFA a partir de la expresión regular

Ejemplo de cálculo de las funciones *firstpos*(), *lastpos*()

NODE n	*nullable*(n)	*firstpos*(n)
A leaf labeled ϵ	**true**	$\emptyset$
A leaf with position i	**false**	$\{i\}$
An or-node $n = c_1 \mid c_2$	*nullable*(c_1) **or** *nullable*(c_2)	*firstpos*(c_1) $\cup$ *firstpos*(c_2)
A cat-node $n = c_1 c_2$	*nullable*(c_1) **and** *nullable*(c_2)	**if** (*nullable*(c_1)) *firstpos*(c_1) $\cup$ *firstpos*(c_2) **else** *firstpos*(c_1)
A star-node $n = c_1{}^*$	**true**	*firstpos*(c_1)

Construcción del DFA a partir de la expresión regular

Ejemplo de cálculo de las funciones *firstpos*(), *lastpos*()



Construcción del DFA a partir de la expresión regular

Cálculo de la función *followpos*(i):

Definición

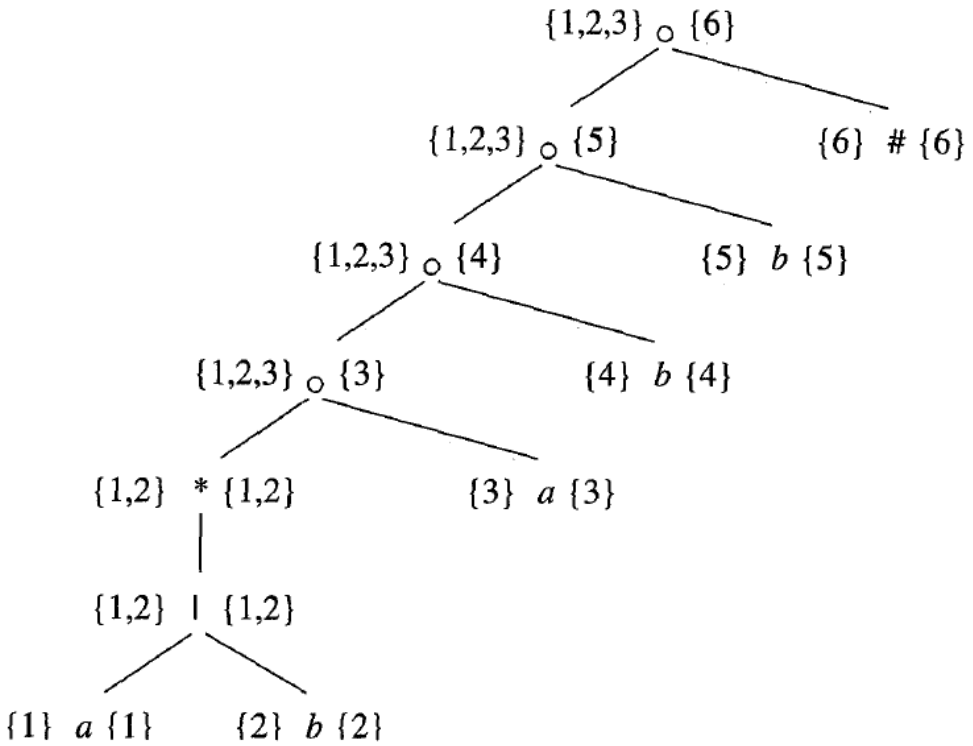
Si n es un nodo de concatenación con hijo izquierdo c_1 e hijo derecho c_2, e i es una posición en lastpos(c_1), entonces todas las posiciones en firstpos(c_2) están en followpos(i).

Definición

Si n es un nodo estrella, e i es una posición en lastpos(n), entonces todas las posiciones en firstpos(n) están en followpos(n).

Construcción del DFA a partir de la expresión regular

Ejemplo de cálculo de la función *followpos*()



NODE n	*followpos*(n)
1	$\{1,2,3\}$
2	$\{1,2,3\}$
3	$\{4\}$
4	$\{5\}$
5	$\{6\}$
6	$\emptyset$

Construcción del DFA a partir de la expresión regular

Algoritmo para construir un DFA a partir de una expresión regular (r) de modo que ambos reconozcan el mismo lenguaje.

1. Construir un árbol de sintaxis para $(r)\#$
2. Calcular las funciones *nullable*(), *firstpos*(), *lastpos*() y *followpos*() haciendo recorridos en profundidad sobre el árbol de sintaxis.
3. Construir el DFA a partir de esa información.

Construcción del DFA a partir de la expresión regular

A = firstpos(root)
Dstates = A
while (Exista un conjunto de estados T no marcado en Dstates) do
    marcar T
    for $a \in \Sigma$ do
        for $p \in T$ do
            if (a es el símbolo en la posición p) then
                $U = U \cup followpos(p)$
        if ($U \neq \emptyset$ and $U \notin Dstates$)
            $Dstates = Dstates \cup \{U\}$
        $Dtran[T, a] = U$
Estado inicial: $firstpos(root)$
Estados finales: todos los que contengan la posición asociada al símbolo #

Minimización de DFAs

Definición

Una cadena w distingue a dos estados s y t si al avanzar por el autómata a partir de s siguiendo la cadena w se llega a un estado final, pero a partir de t se llega a un estado de no aceptación o viceversa.

Si un DFA tiene algún estado del cual no parte arco con alguno de los símbolos del alfabeto, el autómata se puede completar sin afectar el lenguaje que reconoce adicionando un estado sifón.

Minimización de DFAs

Entrada: un DFA M con estados S, alfabeto Σ, estado inicial s_0, estados finales F y todas las transiciones definidas
Salida: Un DFA M' que acepta el mismo lenguaje que M y tiene el menor número de estados posible.

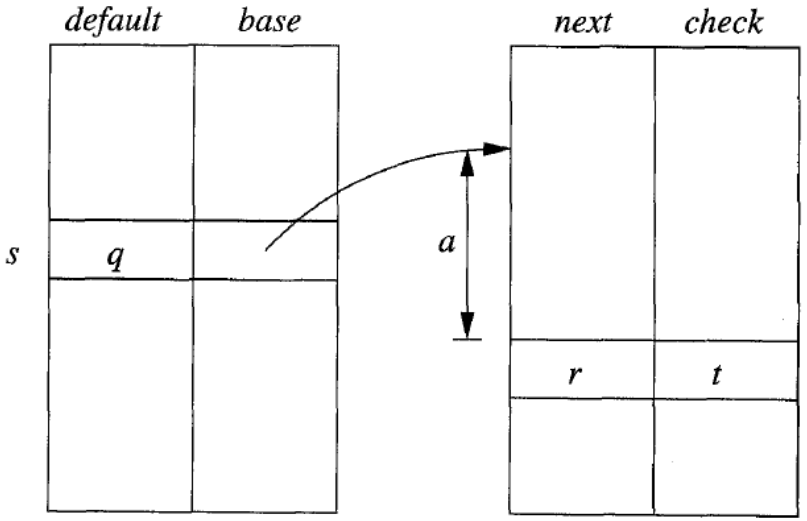
1. Construir una partición inicial Π separando los estados finales del resto
2. Construir una siguiente partición Π_{new} a partir de Π
3. Si $\Pi_{new} = \Pi$ entonces $\Pi_{final} = \Pi_{new}$ e ir al paso 4, sinó ir al paso 2
4. Escoger un estado en cada grupo de Π_{final} como representante
5. Eliminar los estados sifón

Minimización de DFAs

Método para construir la nueva partición Π_{new}
for $G \in \Pi$ do
Partir G en subgrupos tales que dos estados $s, t \in G$ pertenecen al mismo subgrupo si y sólo si $\forall a \in \Sigma, \delta(s, a)$ pertenece al mismo grupo en Π que $\delta(t, a)$.
Adicionar el conjunto de subgrupos obtenidos en Π_{new} en el lugar de G.

Método de compresión de la tabla de transición

- Velocidad vs. Espacio
- Mayor velocidad sin desperdiciar espacio usando 4 arreglos indexados por número de estado: *base* determina la ubicación de las entradas en *next* y *check* para un estado. *default* provee una base alternativa cuando la base actual sea inválida.



Método de compresión de la tabla de transición

siguienteEstado(s, a)
if ($check[base[s] + a] = s$)
    return $next[base[s] + a]$
else
    return $siguienteEstado(default[s], a)$

Bibliografía

Los esquemas de estas transparencias fueron tomados del libro *Compilers: principles, techniques and tools* de A. Aho, R. Sethi, J. Ullman Primera edición. 1988. Sección 3.9