Generadores de analizadores léxicos

4.1 Conversión de una expresión regular en un AFD

Con los métodos de los capítulos anteriores ya es posible realizar este tipo de conversión.

En este capítulo se presenta un método de conversión directo para ir de la expresión regular al autómata finito. El proceso que se utiliza esta mejor adaptado para ser programado ya que se basa en la construcción de un árbol de evaluación para la expresión regular y en el cálculo de varias funciones definidas para el proceso. El tema de la construcción de este tipo de generadores culmina en la siguiente sección, analizando un método para la minimización de autómatas finitos deterministas. Este ultimo método sería él ultimo paso del generador y tiene la finalidad de disminuir el número de estados del modelo.

Descripción general del método

Para construir directamente un AFD a partir de una expresión regular se procede de la siguiente manera.

Paso 1.

Agregar el símbolo # al final de la expresión regular:

r#

Paso 2.

Construir un árbol de evaluación T para la expresión r# y numerar las hojas, considerando las siguientes convenciones:

1. El operador de cerradura (*) tiene la mayor precedencia y es asociativo por la izquierda.

2. El operador de concatenación (·) tiene la segunda mayor precedencia y es asociativo por la izquierda.

3. El operador de unión (+) tiene la menor precedencia y es asociativo por la izquierda.

Paso 3.

Se calculan las funciones anulables(), primera() y ultima() para cada uno de los nodos del árbol, recorriéndolo en profundidad. Un recorrido en profundidad consiste en primero visitar todos los hijos de un nodo para visitar después al nodo.

Recorrido en profundidad

Sea n un nodo de T

```
visita (n)
{
Para cada hijo m de n de izquierda a derecha
{
visita (m)
}
visita (n)
}
```

Paso 4.

Se construye la función siguiente(), definida sobre las hojas de T.

Paso 5.

Se construye el AFD calculando las transiciones a partir del estado inicial

q0 = primera (raíz)

utilizando los valores de la función siguiente().

Definición de las funciones requeridas por el proceso

Función anulable().

Esta función se define sobre los nodos de T. Indica si es o no posible derivar la cadena vacía desde dicho nodo.

Si n es un nodo del árbol T y n1, n2 sus nodos hijos (solo n1 sin n = *), entonces anulable() es una función bolean que se calcula mediante las reglas siguientes:

n	Anulable(n)
Ɛ	1
a € ∑	0
+	anulable(n1) or anulable(n2)
·	anulable(n1) and anulable(n2)
*	1

Función primera()

Esta función se define sobre los nodos de T. Es el conjunto de todos los símbolos que ocupan la primera posición en las cadenas derivadas desde el nodo para el que se calcula.

Si n es un nodo del árbol T y n1, n2 sus nodos hijos (solo n1 sin n = *), entonces primera() es el conjunto de símbolos que se calcula mediante las reglas siguientes:

N	Anulable(n)
Ɛ	Φ
a €∑	{ a }
+	primera(n1) U primera(n2)
·	If anulable(n1) then primera(n1) U primera(n2) else primera(n1)

*	Primera(n1)

Función ultima().

Esta función se define sobre los nodos de T. Es el conjunto de todos los símbolos que ocupan la ultima posición en las cadenas derivadas desde el nodo para el que se calcula.

Si n es un nodo del árbol T y n1, n2 sus nodos hijos (solo n1 si n = *), entonces ultima() es el conjunto de símbolos que se calcula mediante las reglas siguientes:

n	Ultima(n)
Ɛ	ϕ
a Є ∑	{ a }
+	ultima(n1) U ultima(n2)
·	If anulable(n2) then ultima(n1) U ultima(n2) else ultima(n2)
*	Ultima(n1)

Función siguiente()

Esta función se define sobre el conjunto de los números asociados a las hojas del árbol. Es el conjunto de los números asociados a las hojas que pueden ocupar la posición siguiente en las cadenas válidas, con respecto a la posición de la hoja para la que se calcula la función.

La función se define de manera siguiente:

Si i es el número asociado a una hoja del árbol, se recorre en profundidad el árbol a partir de dicha hoja agregando hojas al conjunto siguiente (i) de acuerdo a las siguientes reglas.

1. Los únicos nodos donde posiblemente se añadan hojas son los nodos concatenadores y cerradura.

2. Para nodos concatenadores

Sean n = · con hijo izquierdo n1 e hijo derecho n2

Si i esta en ultima(n1), se colocan en siguiente (i) todos los elementos de primera(n2)

3. Para nodos cerradura

Sea n = *

Si i esta en ultima(n), se colocan en siguiente (i) todos los elementos de primera(n)

Algoritmo para convertir una expresión regular en un AFD

Entrada: Expresión regular r.

Salida: AFD que reconoce a L(r)

Proceso:

1. Construya un árbol T para evaluar la expresión regular aumentada r#.

2. Numere las hojas y mantenga esta relación.

3. Recorra en profundidad el árbol y calcule las funciones anulable(), primera() y ultima() para cada nodo de T.

4. Calcule la función siguiente (i) para cada hoja del árbol.

5. Construya el AFD.

5.1. Sean ∑ el conjunto de símbolos diferentes de r y q0 = primera (Raíz de T)

5.2. Para cada a € ∑ calcule ∫(q0, a) considerando que:

Si el símbolo a esta asociado a los números: i1, i2, ..., i1 incluidos en q, entonces, d se define por

∫ (q, a) = siguiente (i1) U siguiente (i2) U ..... U siguiente (i1)

5.3. Agregar los nuevos estados que aparecen en cada una de las transiciones calculadas y proceder a calcular sus transiciones.

5.4. Realizar el paso 5.3 hasta que ya no aparezcan nuevos estados.

5.5. Los estados finales son aquellos que incluyen el número asociado con el símbolo