Instituto Politécnico Nacional
Escuela Superior de Cómputo
Ingeniería en Sistemas Computacionales

Compiladores

Práctica 3: Implementación de un algoritmo para convertir un AFN a un AFD

Profesor: Rafael Norman Saucedo Delgado

Alumno: Ayala Segoviano Donaldo Horacio

Boleta: 2019630415

Introducción

Autómata Finito Determinista

Un autómata finito determinista, que es aquel que sólo puede estar en un único estado después de leer cualquier secuencia de entradas. El término "determinista" hace referencia al hecho de que para cada entrada sólo existe uno y sólo un estado al que el autómata puede hacer la transición a partir de su estado actual.

Un autómata se define como la quíntupla:

$$A = (Q, \Sigma, \delta, q0, F)$$

Donde:

1. Un conjunto finito de estados, a menudo designado como Q.
2. Un conjunto finito de símbolos de entrada, a menudo designado como Σ.
3. Una función de transición que toma como argumentos un estado y un símbolo de entrada y devuelve un estado. La función de transición se designa habitualmente como δ. En nuestra representación gráfica informal del autómata, δ se ha representa mediante arcos entre los estados y las etiquetas sobre los arcos. Si q es un estado ya es un símbolo de entrada, entonces δ (q,a) es el estado p tal que existe un arco etiquetado a que va desde q hasta p.2
4. Un estado inicial, uno de los estados de Q.
5. Un conjunto de estados finales o de aceptación F. El conjunto F es un subconjunto de Q.

Autómata Finito No Determinista

Un autómata finito "no determinista" (AFN) tiene la capacidad de estar en varios estados a la vez. Esta capacidad a menudo se expresa como la posibilidad de que el autómata "conjeture" algo acerca de su entrada.

Un AFN se representa esencialmente como un AFD:

$$A = (Q, \Sigma, \delta, q0, F)$$

Donde:

1. Q es un conjunto finito de estados.
2. Σ es un conjunto finito de símbolos de entrada.
3. q0, un elemento de Q, es el estado inicial.
4. F, un subconjunto de Q, es el conjunto de estados finales (o de aceptación).
5. δ , la función de transición, es una función que toma como argumentos un estado de Q y un símbolo de entrada de Σ y devuelve un subconjunto de Q. Observe que la única diferencia entre un AFN y un AFD se encuentra en el tipo de valor que devuelve δ : un conjunto de estados en el caso de un AFN y un único estado en el caso de un AFD.

Conversión de un AFN a un AFD

La expresión regular es la notación de elección para describir analizadores léxicos y demás software de procesamiento de patrones. No obstan te, la implementación de ese software requiere la simulación de un AFD, o tal vez la simulación de un AFN. Como, por lo general, un AFN tiene la opción de moverse sobre un símbolo de entrada o sobre ε, o incluso la opción de realizar una transición sobre e o sobre un símbolo de entrada real, su simulación es menos simple que la de un AFD. Por ende, con frecuencia es importante convertir un AFN a un AFD que acepte el mismo lenguaje.

Durante esta práctica se implementará el algoritmo de construcción de subconjuntos, el cual sirve para obtener el AFD que acepta el mismo lenguaje que un AFN. Es necesario mencionar que para iniciar el algoritmo se debe agregar ϵ-cerradura(estado_inicial_del_AFN) a $\mathrm{D}_{\mathrm{estados}}$, y ponerlo como *sin marcar.*

```
while ( hay un estado sin marcar T en Destados ) {
        marcar T;
        for ( cada símbolo de entrada a ) {
                U = ε-cerradura(mover(T, a));
                if ( U no está en Destados )
                        agregar U como estado sin marcar a Destados;
                Dtran[T, a] = U;
        }
}
```

Figura 1. Algoritmo de construcción de subconjuntos.

En la figura 1 se puede observar el algoritmo de construcción de subconjuntos que será implementado en esta práctica ya que nos permite obtener el AFD equivalente de un AFN. A continuación, en la figura 2, se describen las operaciones sobre los estados usadas en el algoritmo y su descripción.

OPERACIÓN	DESCRIPCIÓN
ϵ-*cerradura*(s)	Conjunto de estados del AFN a los que se puede llegar desde el estado s del AFN, sólo en las transiciones ϵ.
ϵ-*cerradura*(T)	Conjunto de estados del AFN a los que se puede llegar desde cierto estado s del AFN en el conjunto T, sólo en las transiciones ϵ; $= \cup_{s \text{ en } T}$ ϵ-*cerradura*(s).
mover(T, a)	Conjunto de estados del AFN para los cuales hay una transición sobre el símbolo de entrada a, a partir de cierto estado s en T.

Figura 2. Operaciones sobre los estados del AFN.

Desarrollo

Para el desarrollo de la práctica se usará la metodología en cascada.

Requisitos

Implementar un algoritmo que convierta un AFN (Autómata Finito No Determinista) a un AFD (Autómata Finito Determinista).

Entrada: Un autómata finito no determinista.

Salida: Un autómata finito determinista que acepta el mismo lenguaje que el AFN de entrada.

Diseño

A continuación, se muestra el diagrama de clases a seguir para la implementación del algoritmo.

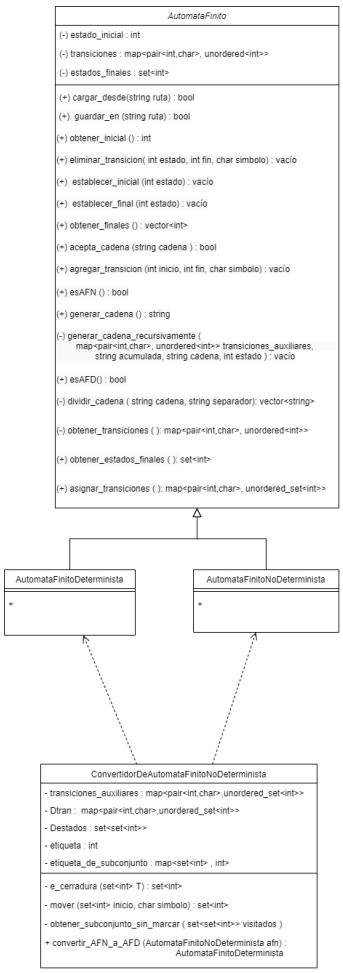

Figura 3. Diagrama de clases.

Implementación

La implementación del algoritmo, se llevó a cabo en el lenguaje de programación C++.

Verificación

Ahora se procederá a probar la clase implementada que es la clase "*ConvertidorDeAutomataFinitoNoDeterminista*", para esto usaremos la clase desarrollada anteriormente para poder proporcionar como entrada el ejemplo del libro de compiladores de Aho. Del dragón morado de la página 155.

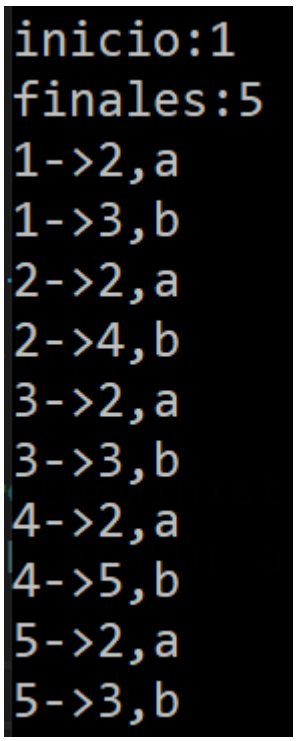

Figura 4. Pruebas de la clase implementada.

En la figura 4 se pueden observar las transiciones del autómata finito determinista producido. Se puede comprobar que el resultado es el mismo al obtenido en el libro de compiladores de Aho del dragón morado, solo que con diferentes etiquetas de estados.

Mantenimiento

Se han corregido algunos errores en las clases desarrolladas anteriormente y de las que depende el correcto funcionamiento de la clase implementada durante la práctica.

Conclusión

Durante el desarrollo de la práctica presente se pudieron adquirir nuevos conocimientos acerca de la implementación de algoritmos, también se conocío la metodología en cascada, sus partes y funciones. Se concluyó que el seguir una metodología bien establecida puede facilitar el desarrollo de un sistema de software.

También se pudo concluir que la práctica funcionó correctamente y que se pudo implementar el algoritmo correctamente.

También se pudo observar que la implementación, que en un principio resultó algo compleja, ahora está resultando en una interfaz más cómoda de usar.

Referencias

- Aho A., Lam M. (2008) *Compiladores, principios técnicas y herramientas. Segunda edición*. México: Pearson Education.
- Hopcroft J., Motwani R. (2007) *Introducción a la teoría de autómatas, lenguajes y computación.* Madrid : Pearson Education.