Universidad del Valle de Guatemala Pablo Díaz 13203

Diseño de Lenguajes de Programación

Proyecto Fase 1

Ing. Tomás Gálvez

**Diseño**

El proyecto incluye:

1. Construcción de un AFN (autómata finito no determinista) por medio del algoritmo de Thomson.

2. Conversión de un AFN a un AFD (autómata finito determinista) por medio de la agrupación de subconjuntos.

3. Creación directa de un AFD por medio de un árbol sintáctico y agrupación.

4. Minimización de un AFD directo y AFD por medio de subconjuntos.

Construcción de un AFN.

La lógica para realizar esta construcción fue primer tener una estructura de datos para modelar un autómata, un estado y una transición. Se sabe que un autómata tiene un estado inicial, un alfabeto, un conjunto de estados, una tabla de transiciones y un conjunto de estados de aceptación. Para la construcción se utilizó una pila de autómatas que va metiendo el resultado y después se va modificando con cada operación. Primer se comienza creando el autómata más simple hasta las operaciones conocidas como cerradura de kleene, unión y concatenación.

Conversión de un AFN a AFD.

Se sabe por teoría que un AFN siempre tiene un equivalente AFD. Este método se basa en crear subconjuntos de estados utilizando operaciones de e-closure (todos los estados posibles que se pueden llegar con épsilon) y move (estado a que se llega por medio de una transición). Básicamente con estas dos operaciones se van creando los subconjuntos de estados que tiene que tener el AFD. La clave para realizar esta implementación fue utilizar una cola (FIFO) y no un stack (LIFO) porque si no se ponían las transiciones al revés. Otro punto clave fue hacer un merge del resultado del e-closure de cada estado al realizar dicha operación. La creación de los estados se basó en cada subconjunto diferente creado y las transiciones se pusieron cuando el subconjunto ya existía.

Creación Directa de un AFD

n Directa de un AFD diferente creado y las transiciones se pusieron cuando el subconjunto ya exist operaciLa clave para realiza

La creación directa de un AFD significa que se crea el autómata a partir una expresión regular. Para la creación directa primero se tuvo que realizar un árbol sintáctico con la versión extendida de la expresión regular. Para futuras implementaciones se recomienda leer la expresión de derecha a izquierda debido a que la expresión se encuentra en notación postfix y los nodos hoja (terminales ) se encuentran al inicio de la expresión. Para la creación directa se tiene que calcular una tabla llamada follow pos, que indica los estados que se deben crear. La creación de esta tabla incluye calcular otros conjuntos llamados first pos , last pos y nullable a partir de los nodos del árbol sintáctico.

Minimización de un AFD

La minimización consiste en eliminar los estados inalcanzables y los estados indistinguibles. Los estados inalcanzables son los estados que no se pueden alcanzar desde el estado inicial del autómata con alguna cadena. Los estados indistinguibles son los estados que se pueden unir en uno solo porque tienen las mismas transiciones. El algoritmo implementado es el de particiones, el cual empieza con colocar en conjuntos diferentes los estados de aceptación y los de no aceptación en un mismo conjunto. Luego se calculan los grupos a los que pertenecen cada estado dependiendo si tienen transiciones a otro grupo dentro la misma partición. Después, se vuelven a realizar particiones sobre estos grupos hasta que ya no se pueda.

**Problemas Encontrados**

1. Tuve el problema que en el e-closure se me había olvidado incluir el estado del cual se realiza e-closure y también tuve el problema que no había verificado si el estado ingresado al resultado del e-closure no estuviera repetido en la pila de estados, esto me generaba un ciclo infinito.

2. Tuve que realizar la concatenación de estados por medio de una transición épsilon.

3. Armé el árbol sintáctico mezclando una pila y recursividad porque tenía una expresión en postfix y leí la cadena de izquierda a derecha, por lo que los primeros nodos creados eran hojas y hasta después creaba los estados que tendrían hijos que serían hojas, por lo que tuve que meter las hojas a una pila de nodos y cuando creaba los nodos operadores le ponía de hijos lo que tuviera en la pila.

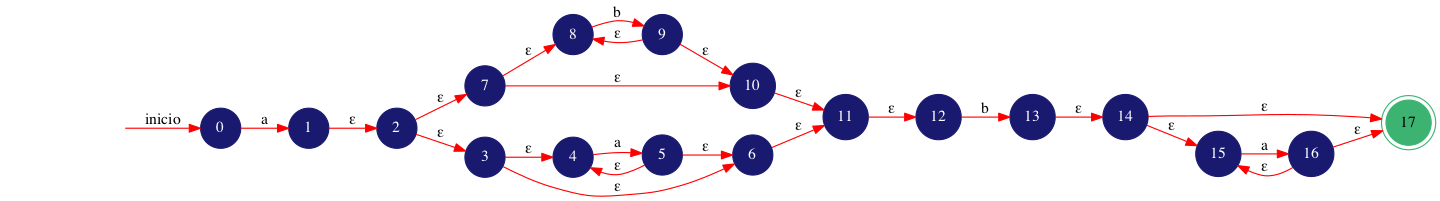
4. Se tuvo un problema a la hora de calcular follow pos porque realizo un merge del first pos anterior pero cuando cambia de iteración, se quedaba el resultado anterior entonces la solución era utilizar una lista temporal o re-calcular follow pos en cada iteración para no alterar el resultado en las siguientes iteraciones.

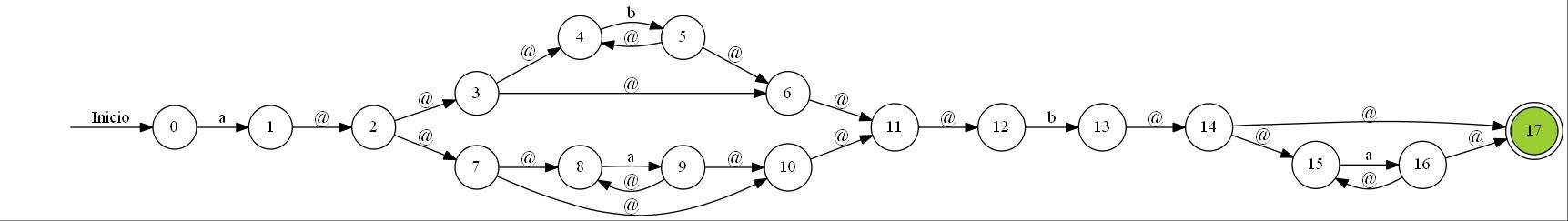
5. El problema en la minimización fue que saber que estructuras de datos utilizar para implementar el algoritmo de hopcroft. Se recomienda utilizar mejor un ciclo while true porque se modifica la partición dentro este ciclo porque si se utiliza un ciclo for no se puede cambiar el arreglo que se está recorriendo.

**Pruebas**

Regex de prueba: a(b\*|a\*)ba\*

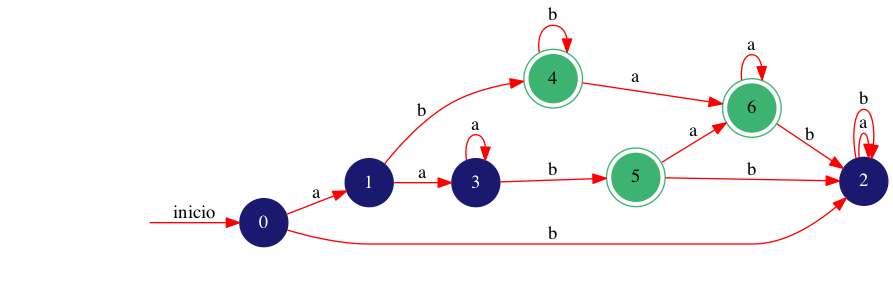
**AFN**

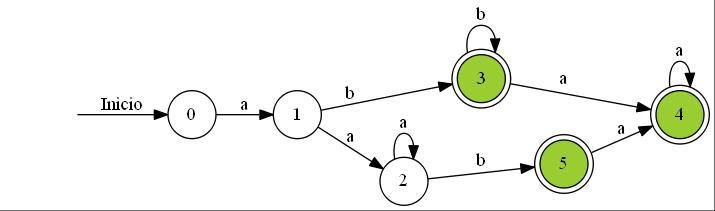




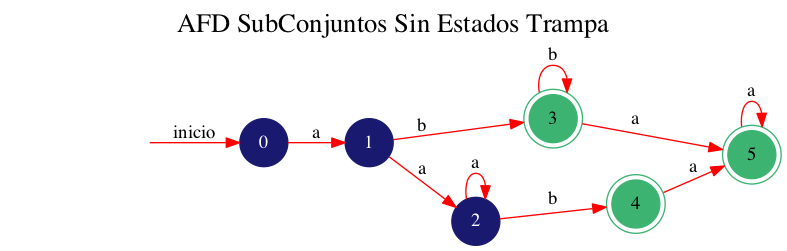
Comparación con otro compañero: Estos autómatas son exactamente iguales

**AFD Subconjuntos**

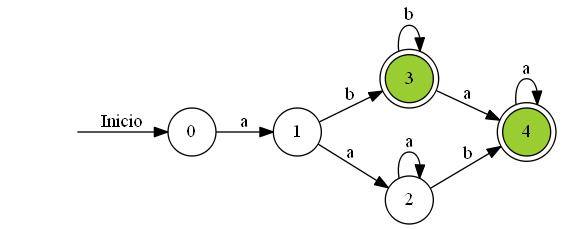
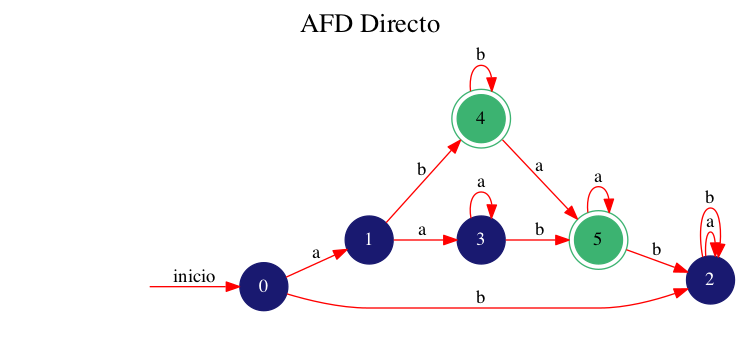




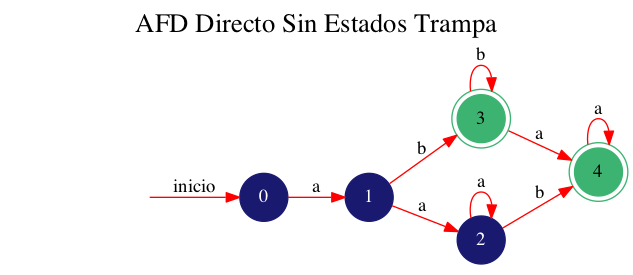
La diferencia entre estos autómatas es que daniel no sigue la definición formal de un AFD, es decir que no todos los nodos tienen transiciones con todo el alfabeto. Por lo tanto, mi AFD tiene un estdo más, que es el considerado el de trampa. Esto puede ser ineficiente para autómatas muy grandes por lo que se implementó un método para quitarlos. Al quitarlo queda de la siguiente manera.

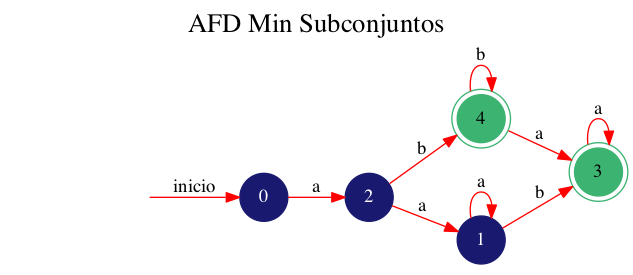
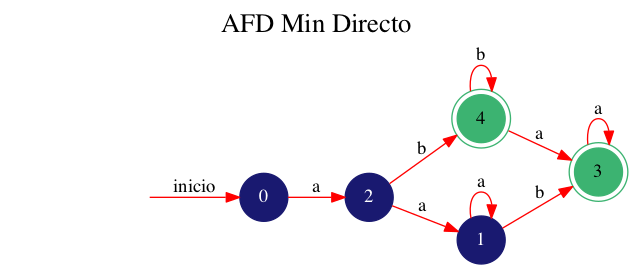


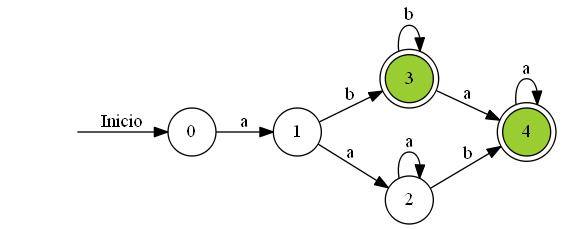
**AFD Directo**

****

En este caso se puede apreciar que ocurre lo mismo que el caso anterior, debido a que el nodo inicial tiene transición con b a un estado de trampa, a cambio daniel no tiene transición b en el estado inicial, por eso mi autómata tiene un estado más. Al quitar los estado de trampa quedan así.



**AFD minimizado Directo y Subconjuntos**

****

Aquí se puede ver que los AFD minimizados son únicos.