Programación Declarativa
Proyecto
Automatás Traductores. Ejemplos, traducción y visualización.

Diego Méndez Medina 420004358

Índice

Como proyecto final de la materia Programación Declarativa se realizo una herramienta (*software*) escrita en `HASKELL` que nos permite trabajar con automátas finitos traductores definidos en archivos y cambiarlos entre sus tipos (Moore y Mealy). Se visualiza la equivalencia y cuando es mejor uno sobre el otro.

1. Antecedentes

El automáta finito(AF), con sus múltiples variantes, es el modelo aceptador más *primitivo*. Pero si modificamos, agregando una función, el modelo lo convertimos, en no solo una maquína aceptadora, sino también en una maquina traductora.

La definición de la función(de traducción) y sus posibles variaciones es lo que nos permite, para AF, obtener dos traductores[1].

Como experiencía personal y hablando con compañeros que cursaron con otrxs profesores, no se suele hablar de automátas traductores en los cursos de Automátas y Lenguajes Formales que se dan en la Facultad de Ciencias.

El presente proyecto surgio como idea para familiarizarme más con el concepto y también para tener una herramienta que nos permita mostrar la traducción entre automátas traductores.

2. Analísis del Problema

El uso de automátas traductores es más *popular* en *aplicaciones* que el de los automatás de aceptación. Esto por que los primeros nos suelen permiter trabajar con toda la cadena, dar más información de esta y así **extender** el poder de computo sobre las mismas cadenas.

Un uso común es en el de compiladores para lexers pero también tienen un uso muy conocido en el desarollo de circuitos y encriptación.

[1] A pesar de no ser aceptadores para que se lleve acabo la traducción el tipo de cadenas se debe de poder aceptar para algún AF.[1]

No todo automáta traductor es de aceptación[1], el presente se enfoca solo en automatás traductores que también son de aceptación. Si bien los no aceptadores también tienen aplicaciones para la generación de circuitos, lexers y encriptación es necesario tener un inicio.

Como bien ya mencionamos para AF hay dos tipos de traductores y su diferencia radica en la forma en la que la función de respuesta o traducción esta definida. Veremos ahora la definición de cada uno.

2.1. Automata de Moore

La respuesta de estos automátas depende **exclusivamente** del estado en el que se encuentra el automáta, así es independiente de la entrada. . Un automáta de Moore, también llamada *de respuesta asignada por estado*, es una tupla $M = (Q, \Sigma, O, \delta, \lambda, q_0, F)$, donde:

Q : Conjunto de estados
Σ : Alfabeto de entrada
O : Alfabeto de salida
δ : Función de transición
: $Q \times \Sigma \to Q$
λ : Función de respuesta
: $Q \times \Sigma \to O^*$
q_0 : Estado inicial
F : Conjunto de estados Finales

2.2. Automata de Mealy

Por otro lado en estos automátas dan su respuesta durante el proceso de la cadena.

También llamados automátas con respuesta asignada durante la transición es un tupla $M = (Q, \Sigma, O, \delta, \lambda, q_0, F)$, donde:

Q : Conjunto de estados
Σ : Alfabeto de entrada
O : Alfabeto de salida
δ : Función de transición
: $Q \times \Sigma \to Q$
λ : Función de respuesta
: $Q \to O^*$
q_0 : Estado inicial
F : Conjunto de estados Finales

Como podemos ver son muy similares pues son una extensión de los AF salvo por como lo hemos dicho la función de traducción o respuesta. Existe una relación entre estos automátas y es que es posible hablar de la equivalencía entre traductores. Decimos que un automáta de Moore y uno de Mealy son equivalentes cuando podemos simular la conducta del otro modelo.

3. Implementación / Solución

En `HASKELL` es común trabajar con automátas definiendo la función de transición(delta), así ya incluimos los estados y el alfabeto y por último damos la función de aceptación, donde en realidad solo estamos determinando quienes son finales. Con ese acercamiento, si bien eficaz, se pierde la noción conjuntista de los automátas como tuplas.

Lo que buscamos con el desarrollo del proyecto no es simplemente generar automátas en un archivo con extensión *hs* y transformarlo, sino crear una herramienta qie nos permita leer cualquier tipo de automáta traductor y que mediante un formato especifico (archivo txt) trabajar con este: aceptando cadenas del automáta traductor recibido, traduciendo cadenas en este, pasar del tipo automáta traductor recibido al otro tipo y hacer las mismas operaciones con este, por último también guardar el automáta generado en un archivo para su posterior consulta.

Lo siguiente es desarrollar el formato para guardar automátas en un archivo para que nuestra herramienta lo lea y dar un panorama de la implementación de nuestra herramienta.

3.1. Definiendo el formato

Vamos a trabajar con archivos de texto plano, donde se busca esten las siguientes palabras reservadas:

Moore	Mealy
Finales	Estados
Entrada	Salida
Transición	Respuesta
Inicial	

Cada archivo que represente un traductor debe tener todos de los anteriores salvo que solo debe escojer entre *Moore* o *Mealy*, dependiendo del tipo que sea. Para cada una de las palabras reservadas(salvo el tipo) se sigue un signo de igual que describe el conjunto.

- Estados: Los estados son cadenas y estan separados por coma
- Entrada: Es el alfabeto de entrada son caracteres separados por ','
- Salida: Es el alfabeto de entrada son caracteres separados por ','
- Finales: Similar a Estados.
- Transición: Son de la forma Estado"-¿"Estado, estan separados por ','
- Respuesta: Función de traducción y depende del tipo.
  - Para Moore: Son de la forma: Estado-¿Salida
  - Para Mealy: Son de la forma: Estado-¿Entrada-¿Salida
- Inicial: Un único estado.

Al finalizar la descripción de cada conjunto se finaliza con un '.'. Se permite que haya saltos de línea entre descripciones para no tener líneas muy largas. Es decir es valido tanto:

$$Respuesta : q0- > 0- > 0, q0- > 1- > 1, q4- > 0- > 0.$$

como:

$$\begin{aligned} Respuesta :& q0- > 0- > 0, q0- > 1- > 1, \\ & q4- > 0- > 0. \end{aligned}$$

Así mostramos a continuación la representación de dos automátas:

$$\begin{aligned} & Moore. \\ Estados &= A, B, C, D. \\ Entrada &= 0, 1. \\ Salida &= 0, 1, 2, 3, 4. \\ Transicion &= A- > 0- > A, A- > 1- > B, \\ & B- > 0- > B, B- > 1- > C, \\ & C- > 0- > C, C- > 1- > D, \\ & D- > 0- > D, D- > 1- > A. \\ Respuesta &= A- > 0, B- > 1, C- > 2, D- > 3. \\ Inicial &= A. \\ Finales &= A, B, C, D. \end{aligned}$$

Automáta de Mealy que dada una cadena de bits de multiplos de cuatro traduce la cadena de tal forma que para cuada cuatro bits que lea los reproduce y genera un quinto bit de tal forma que el número de unos es impar.

$$\begin{aligned} & Mealy. \\ Inicial &= q0. \\ Finales &= q0. \\ Estados &= q0, q1, q2, q3, q4, q5, q6. \\ Entrada &= 0, 1. \\ Salida &= 0, 1. \\ Respuesta &= q0- > 0- > 0, q0- > 1- > 1, \\ & q4- > 0- > 0, q4- > 1- > 1, \\ & q1- > 0- > 0, q1- > 1- > 1, \\ & q5- > 0- > 0, q5- > 1- > 1, \\ & q2- > 0- > 0, q2- > 1- > 1, \\ & q6- > 0- > 01, q6- > 1- > 10, \\ & q3- > 0- > 00, q3- > 1- > 11. \\ Transicion &= q0- > 0- > q4, q0- > 1- > q1, \\ & q4- > 0- > q5, q4- > 1- > q2, \\ & q1- > 0- > q2, q1- > 1- > q5, \\ & q5- > 0- > q6, q5- > 1- > q3, \\ & q2- > 0- > q3, q2- > 1- > q6, \\ & q6- > 0- > q0, q6- > 1- > q0, \\ & q3- > 0- > q0, q3- > 1- > q0. \end{aligned}$$

Para las palabras reservadas no hay distinción entre mayuscuas y minusculas, es decir *Moore, moore,*

MoOre, MOre y demás combinación son válidas. No confundir con el lado derecho de la igualdad pues ahi si hay diferencia entre el estado A y a, sucede lo mismo con alfabetos. Tambíen no importa el uso de espacios, saltos de línea ni tabulaciónes siempre y cuando esten las ocho palabras del automaáta finalizadas con su respectivo punto. Por último no importa el orden en el que aparezcan. Como se vio en los ejemplos. Estos automátas y otros ejemplos se encuentran en la carpeta *automatas/* del repositorio.

3.2. Sobre la Implementación

Vimos a cada traudctor como una extensión de los *AF*, y es así que definimos los automátas de la siguiente manera:

```
module Automata where

type Estado = [Char]
type Alfabeto = Char
type Transicion = (Estado, Alfabeto,
                   Estado)
type Respuesta = (Estado, [Alfabeto])
type RespuestaMealy =
  (Estado, Alfabeto, [Alfabeto])

data Automata = Automata{
  estados :: [Estado],
  alfabetoEntrada :: [Alfabeto],
  alfabetoSalida :: [Alfabeto],
  transiciones :: [Transicion],
  inicial :: Estado,
  finales :: [Estado]
  } deriving Show

data Moore = Moore{
  fRespuestas :: [Respuesta],
  automataMoore :: Automata
  } deriving Show

data Mealy = Mealy{
  fRespuestasM :: [RespuestaMealy],
  automataMealy :: Automata
```

Así definimos las funciones normales para automátas que nos permiten procesar cadenas:

Transita sobre el autómata dada la cadena

```
transita :: Automata -> [Alfabeto] -> [Estado]
```

Acepta el autómata

```
acepta :: Automata -> [Alfabeto] -> Bool
```

Su implementación como las funciones auxiliares se encuentran en la carpeta src del repositorio.

3.3. Traduciendo Traductores

4. Conclusión

Es por eso que si bien estos traducen no suelen ser efectivos para un método de encriptación general, pues como veremos a la hora de traducir necesitan un mayor numero de estados

5. Revisión Bibliográfica

Referencias

[1] Elisa Viso, E. “Introducción a Autómatas y Lenguajes Formales”, UNAM, CDMX, 2015.

[2] Guillermo Morales, L. “Máquinas secuenciales”, CINVESTAV, CDMX, Junio del 21. Recuperado en Junio del 2023 de el siguiente enlace.