

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

Diego Méndez Medina 420004358

## Índice

<b>1. Antecedentes</b>	<b>1</b>
<b>2. Análisis del Problema</b>	<b>1</b>
2.1. Automata de Moore	2
2.2. Automata de Mealy	2
<b>3. Implementación / Solución</b>	<b>2</b>
3.1. Definiendo el formato	3
3.2. Sobre la Implementación	4
3.3. Traduciendo Traductores	4
<b>4. Conclusión</b>	<b>4</b>
4.1. Trabajo Futuro	4
<b>5. Revisión Bibliográfica</b>	<b>5</b>

Como proyecto final de la materia Programación Declarativa se realizo una herramienta (*software*) escrita en HASKELL que nos permite trabajar con automatás 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 automatá 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 máquina 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 experiencia personal y hablando con compañeros que cursaron con otrxs profesores, no se suele hablar de automatás traductores en los cursos de Automatás y Lenguajes Formales que se dan en la Facultad de Ciencias.

El presente proyecto surgio como idea para familia-

rizarme más con el concepto y también para tener una herramienta que nos permita mostrar la traducción entre automatás traductores.

## 2. Análisis del Problema

El uso de automatás traductores es más *popular* en *aplicaciones* que el de los automatás de aceptación. Esto por que los primeros nos suelen permitir 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 desarrollo de circuitos y encriptación.

No todo autómatá traductor es de aceptación<sup>1</sup>, el presente se enfoca solo en autómatás 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.

$Q$  : Conjunto de estados  
 $\Sigma$  : Alfabeto de entrada  
 $O$  : Alfabeto de salida  
 $\delta$  : Función de transición  
 $\quad : Q \times \Sigma \rightarrow Q$   
 $\lambda$  : Función de respuesta  
 $\quad : Q \rightarrow O^*$   
 $q_0$  : Estado inicial  
 $F$  : Conjunto de estados Finales

## 2.1. Automata de Moore

La respuesta de estos autómatás depende **exclusivamente** del estado en el que se encuentra el autómatá, así es independiente de la entrada. . Un autómatá 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  
 $\Sigma$  : Alfabeto de entrada  
 $O$  : Alfabeto de salida  
 $\delta$  : Función de transición  
 $\quad : Q \times \Sigma \rightarrow Q$   
 $\lambda$  : Función de respuesta  
 $\quad : Q \times \Sigma \rightarrow O^*$   
 $q_0$  : Estado inicial  
 $F$  : Conjunto de estados Finales

## 2.2. Automata de Mealy

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

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

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 autómatás y es que es posible hablar de la equivalencia entre traductores. Decimos que un autómatá 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 autómatás 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 autómatás como tuplas.

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

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

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

### 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 escoger 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:

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

Así mostramos a continuación la representación de dos autómatas:

*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* =  $A, B, C, D.$

Autómata 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.

*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* =  $q0- > 0- > q4, q0- > 1- > q1,$

$q4- > 0- > q5, q4- > 1- > q2,$

$q1- > 0- > q2, q1- > 1- > q5,$

$q5- > 0- > q6, q5- > 1- > q3,$

$q2- > 0- > q3, q2- > 1- > q6,$

$q6- > 0- > q0, q6- > 1- > q0,$

$q3- > 0- > q0, q3- > 1- > q0.$

Para las palabras reservadas no hay distinción entre mayúsculas y minúsculas, es decir *Moore*, *moore*, *MoOre*, *MORe* y demás combinación son válidas. No confundir con el lado derecho de la igualdad pues ahí si hay diferencia entre el estado *A* y *a*, sucede lo mismo con alfabetos. También no importa el uso de espacios, saltos de línea ni tabulaciones siempre y cuando estén las ocho palabras del automata finalizadas con su respectivo punto. Por último no importa el orden en el que aparezcan. Como se vio en los ejemplos. Estos autómatas y otros ejemplos se encuentran en la carpeta *automatas/* del repositorio.

### 3.2. Sobre la Implementación

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

```
module Automata where

import Data.List
import Data.Set

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
```

Observamos que en efecto en lo único que varían es en la función de respuestas. También observamos que tratamos a las funciones como tuplas donde los primeros elementos son el dominio y el último el regreso.

Así definimos las funciones normales para au-

tomátas 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

Me apoyé principalmente en el libro de Elisa Viso, pues es el material que conocía y se me hace conciso. Esta sección es de muy pocas páginas y solo da ejemplos y los algoritmos de traducción. No hablaré mucho de la traducción pues creo el código lo hace por sí mismo. Sí haré mención que la traducción de Moore a Mealy es mucho más fácil e intuitiva de entender y programar.

## 4. Conclusión

Traducir de Mealey a Moore resulta costoso, pues generamos muchos más estados con su respectiva traducción, a los cuales es necesario añadir las transiciones necesarias.

Basta con ver el ejemplo de la máquina que traduce la paridad de Bits, se encuentra como *paridad-BitsMealy.txt*, el cual cuenta con 10 funciones de respuesta/transición y 7 estados. Su traducción genera 11 estados y 14 traducciones/transiciones.

Es por eso que los de Moore no suelen ser efectivos para un método de encriptación físico, pues necesitan un mayor número de estados i.e de recursos y complejidad.

Es posible que se nos facilite hacer más un autómata de Moore que de Mealey, o viceversa, con nuestra herramienta logramos la traducción y además podemos trabajar con ambos autómatas.

Se descubrió un typo en el autómata de la figura 2.10[1], a la traducción le hace falta la transición B0-¿B1.

### 4.1. Trabajo Futuro

- No se logró hacer una interfaz gráfica para desplegarlos.
- Se puede extender a diferentes autómatas, con sus respectivos traductores.

- Implementar el algoritmo de reducción.

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

## 5. Revisión Bibliográfica

### Referencias

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