Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Minimización de mezcladores

Tesis de Licenciatura en Ciencias de la Computación

Pablo Leonardo Cremona

pcremona@dc.uba.ar
LU: 360/98

Directora: Verónica Becher

Buenos Aires, 3 de diciembre de 2021

MINIMIZACIÓN DE MEZCLADORES

Los mezcladores son autómatas finitos con dos cintas de entrada y una cinta de salida, es decir, son transductores finitos con dos cintas de entrada y unas de salida. La salida intercala los símbolos de cada una de las secuencias de entrada, conservando el órden en el que aparecen en las entradas. La teoría clásica de minimización de autómatas finitos no resuelve el problema de minimización de mezcladores porque los mezcladores admiten una noción de indistinguibilidad entre estados más amplia, que no se obtiene por refinamientos sucesivos. En esta tesis damos un algoritmo para obtener a partir de un mezclador determinístico otro equivalente, pero irreducible.

Palabras claves: Mezcladores, Traductores, Automatas, Minimización, Indistinguibilidad.

MIXERS MINIMIZATION

Mixers are finite automata with two input tapes and one output tape, i.e., they are transducers with two input tapes and one output tape. The output interleaves the symbols of each input sequence, preserving the order in which they appear in the inputs. The classical theory of minimization of finite automata does not solve the problem of minimization of mixers because the mixers admit a wider notion of indistinguishability between states, which is not obtained by successive refinements. In this thesis we present an algorithm that, given a deterministic mixer, produces an equivalent, but irreducible, mixer.

Keywords: Mixers, Transducers, Automata, Minimization, Indistinguishability.

CONTENIDOS

1. MEZCLADORES

Los *mezcladores*, en inglés *shufflers*, son autómatas finitos con dos cintas de entrada y una cinta de salida de modo que la salida contiene todos los símbolos de las dos de entrada, pero mezclados. Es decir, la salida intercala los símbolos de cada una de las secuencias de entrada, conservando el orden en el que aparecen en las palabras de entrada. La Figura 1.1 ilustra un mezclador. Los mezcladores aparecen explícitamente en [9] y en [8].

En este trabajo nos proponemos dar un algoritmo de minimización de mezcladores determinísticos.

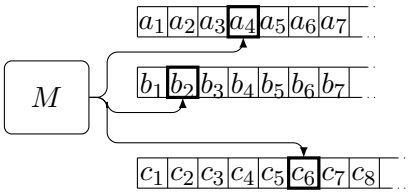


Fig. 1.1: Un mezclador.

Formalmente:

Definición 1. Un *mezclador determinístico* es una tupla de la forma $M =< Q, \Sigma, \delta, \tau, q_0, F >$ donde:

- Q conjunto de estados
- Σ alfabeto o conjunto de símbolos
- $\delta : Q \times \Sigma \cup \{\lambda\} \times \Sigma \cup \{\lambda\} \to Q$ función de transición
- $\tau : Q \times \Sigma \cup \{\lambda\} \times \Sigma \cup \{\lambda\} \to \Sigma$ función de traducción
- $q_0 \in Q$ estado inicial
- $F \subseteq Q$ conjunto de estados finales

Y vale que:

$\forall (q, a, b) \in Q \times \Sigma \cup \{\lambda\} \times \Sigma \cup \{\lambda\}, \delta(q, a, b) \neq \emptyset$ implica $(a = \lambda$ o $b = \lambda)$ y $a \neq b$
$\forall (q, a, b) \in Q \times \Sigma \cup \{\lambda\} \times \Sigma \cup \{\lambda\}, c \in \Sigma, \tau(q, a, b) = c$ implica $a = c$ y $b = \lambda$ o $b = c$ y $a = \lambda$
$\forall (q, a, b), \delta(q, a, b)$ está definido si y solo si $\tau(q, a, b)$ está definido

Es decir que solo se puede consumir de una de las cintas de entradas por transición. Cada una de sus transiciones es de tipo $p \xrightarrow{a,\lambda|a} q$ o del tipo $p \xrightarrow{\lambda,a|a} q$.

Definición 2. La configuración instantánea de un *mezclador* es una tupla

$$(q, \alpha, \beta, \eta).$$

con $q \in Q$, α la entrada por consumir de la cinta 1, β la entrada por consumir de la cinta 2 y η la salida.

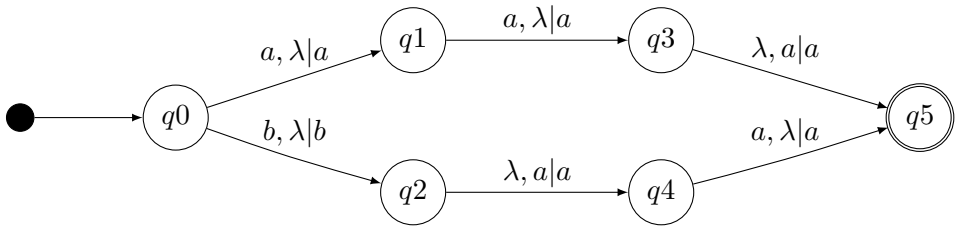


Fig. 1.2: Un *mezclador* no mínimo.

Definición 3. Relación entre configuraciones instantáneas. Sea $q_i, q_j \in Q$, $a \in \Sigma$, $\alpha, \beta, \eta \in \Sigma^*$ y . como la concatenación entre símbolos y cadenas

$$(q_i, a.\alpha, \beta, \eta) \vdash (q_j, \alpha, \beta, \eta.a) \text{ si y solo si } \delta(q_i, a, \lambda) = q_j \wedge \tau(q_i, a, \lambda) = a$$

$$(q_i, \alpha, a.\beta, \eta) \vdash (q_j, \alpha, \beta, \eta.a) \text{ si y solo si } \delta(q_i, \lambda, a) = q_j \wedge \tau(q_i, \lambda, a) = a$$

Definición 4. Dado un mezclador $M =< Q, \Sigma, \delta, \tau, q_0, F >$ definimos la relación calculada por M (traducción) y lo notamos $R(M)$:

$$R(M) = \{(\alpha, \beta, \eta) \in (\Sigma^*, \Sigma^*, \Sigma^*) | \exists q_f \in F, (q_0, \alpha, \beta, \lambda) \vdash^* (q_f, \lambda, \lambda, \eta)\}$$

Por comodidad definimos $\hat{\delta}$ como la aplicación sucesiva de δ .

Definición 5. Sea $\hat{\delta} : Q \times \Sigma^* \times \Sigma^* \to Q$:

$$\hat{\delta}(q, \lambda, \lambda) = q$$

$$\hat{\delta}(q, a.\alpha, \lambda) = \hat{\delta}(\delta(q, a, \lambda), \alpha, \lambda)$$

$$\hat{\delta}(q, \lambda, a.\alpha) = \hat{\delta}(\delta(q, \lambda, a), \lambda, \alpha)$$

$$\hat{\delta}(q, a.\alpha, b.\beta) = \begin{cases} \hat{\delta}(\delta(q, a, \lambda), \alpha, b.\beta) & \text{si} \quad \delta(q, a, \lambda) \neq \emptyset \\ \hat{\delta}(\delta(q, \lambda, b), a.\alpha, \beta) & \text{si} \quad \delta(q, \lambda, b) \neq \emptyset \end{cases}$$

Definición 6. Sea $\hat{\tau} : Q \times \Sigma^* \times \Sigma^* \to \Sigma^*$:

$$\hat{\tau}(q, \lambda, \lambda) = \lambda$$

$$\hat{\tau}(q, a.\alpha, \lambda) = a.\hat{\tau}(\delta(q, a, \lambda), \alpha, \lambda)$$

$$\hat{\tau}(q, \lambda, a.\alpha) = a.\hat{\tau}(\delta(q, \lambda, a), \lambda, \alpha)$$

$$\hat{\tau}(q, a.\alpha, b.\beta) = \begin{cases} a.\hat{\tau}(\delta(q, a, \lambda), \alpha, b.\beta) & \text{si} \quad \delta(q, a, \lambda) \neq \emptyset \\ b.\hat{\tau}(\delta(q, \lambda, b), a.\alpha, \beta) & \text{si} \quad \delta(q, \lambda, b) \neq \emptyset \end{cases}$$

En adelante usaremos τ_C y $\hat{\tau}_C$ para la restricción de τ y $\hat{\tau}$ al conjunto de estados en el camino C.

Definición 7. Decimos que el *mezclador* $M =< Q, \Sigma, \delta, \tau, q_0, F >$ es mínimo si para cualquier otro *mezclador* M', tal que $R(M) = R(M')$ vale que $|Q_M| \leq |Q_{M'}|$

El mezclador Figura 1.2 calcula la misma relación que el mezclador de la Figura 1.3 y que el mezlador de la Figura 1.4, que es el mínimo. En el ejemplo de la Figura 1.2 se puede observar que no alcanza con la minimización clásica de autómatas ya que los caminos q_1, q_3, q_5 y q_2, q_4, q_5 no son iguales pero calculan la misma traducción $(a, a|aa)$ en ambos casos. Si intentaramos transformar el mezclador en un autómata determinístico el algoritmo clasico de minización de automatas no reduciría la cantidad de estados ya que q_3 y q_4 serian distinguibles. Va a ser necesario reordenar algunas transiciones, manteniendo la traducción (Figura 1.3), para luego poder minimizar clasicamente.

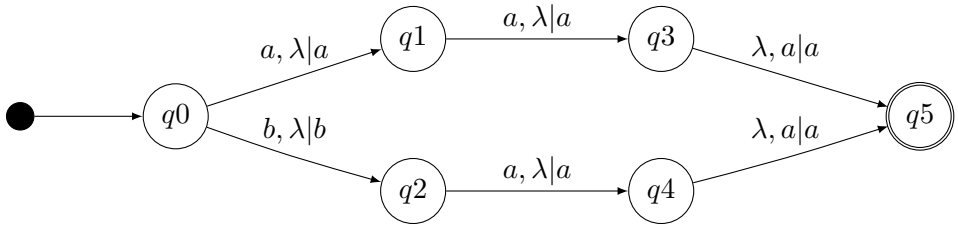


Fig. 1.3: Un *mezclador* no mínimo reordenado.

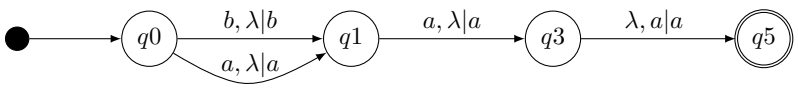


Fig. 1.4: Un *mezclador* mínimo.

2. NUESTRO ALGORITMO DE MINIMIZACIÓN

2.1. El algoritmo clásico de minimización no alcanza

Un autómata finito es mínimo si tiene el mínimo número de estados entre todos los autómatas que reconocen el mismo lenguaje. En el caso de los autómatas finitos determinísticos este autómata mínimo es único, salvo isomorfismos.

Consideremos un autómata determinístico $A =< Q, \Sigma, \delta, q_0, F >$ sobre el alfabeto Σ con el conjunto de estados Q, función de transición δ, estado inicial q_0, y conjunto de estados finales F. La función de transición debe ser completa.

Para cada estado q corresponde un subautómata de A obtenido cuando q es elegido como estado inicial. Llamamos a este subautómata basado en q o simplemente autómata en q. Usualmente solo consideramos el autómata podado, es decir la parte del autómata que es accesible desde q. Para cada estado q corresponde un lenguaje $L_q(A)$ que es el conjunto de cadenas reconocidas por el subautómata en q, esto es

$$L_q(A) = \{w \in \Sigma^* : \hat{\delta}(q_0, w) \in F\}$$

Este lenguaje es llamado el futuro del estado q.

El autómata A es mínimo si $L_p(A) \neq L_q(A)$ para cada par distinto de estados p, q. La relación de equivalencia $\equiv$ definida por

$$p \equiv q \text{ si y solo si } L_p(A) = L_q(A)$$

es un congruencia, es decir $p \equiv q$ implica $\delta(p, a) \equiv \delta(q, a)$ para todos los símbolos a. Se denomina La congruencia de Nerode o relación de indistinguibilidad. Notar que la congruencia de Nerode satura el conjunto de estados finales. Así, un autómata determinista de estado finito es mínimo si y solo si su equivalencia de Nerode es la identidad.

Minimizar un autómata de estados finito es el problema de computar la equivalencia de Nerode. De hecho, el cociente del autómata $A\backslash \equiv$ se obtiene tomando como conjunto de estados el conjunto de clases de equivalencia de la equivalencia de Nerode, como estado inicial el la clase de equivalencia del estado inicial q_0, como estado final el conjunto de clases de equivalencia de los estados en F y definiendo las función de transición como $\delta([p], a) = [\delta(p, a)]$. Acepta el mismo lenguaje y la equivalencia de Nerode es la identidad. El autómata mínimo que reconoce un lenguaje dado es único.

Para los autómatas finitos no determinísticos no es posible definir el autómata mínimo en base a lenguaje cociente. Tampoco es posible para los transductores [1, 6, 3].

Otra manera de ver a los mezcladores es reformular las dos cintas de entrada en una

sola cinta de entrada con un alfabeto ampliado y usando una parte del alfabeto para la cinta 1 y la otra parte del alfabeto para la cinta 2.

Nuestros mezcladores son transductores determinísticos y no es posible definir el autómata mínimo en base al lenguaje cociente. Hay varias obstrucciones. Una es que la definición de mezclador necesariamente produce una función no completa y el algoritmo de minimización clásica requiere que la función de transición del autómata sea completa.

La segunda razón es más sutil e insalvable. Los mezcladores leen de dos cintas de entrada, pero de a un símbolo a la vez, imponiendo un orden en la lectura de los símbolos. Cuando los contenidos de ambas cintas coinciden en una porción, hay dos maneras de computar la misma función: una manera es con un mezclador que lee primero de la primera cinta y luego de la segunda; la otra es con un mezclador que lee primero de la segunda y luego de la primera. El problema de minimizar estos mezcladores determinísticos es en realidad equivalente a un problema de identificar caminos equivalentes en un autómata finito no determinístico, y elegir el más corto.

En el autómata (que no es un mezclador válido) que mostramos en la Figura 2.1 se ve el tipo de problema. Y en la Figura 2.3 se ve el mismo fenómeno en un mezclador.

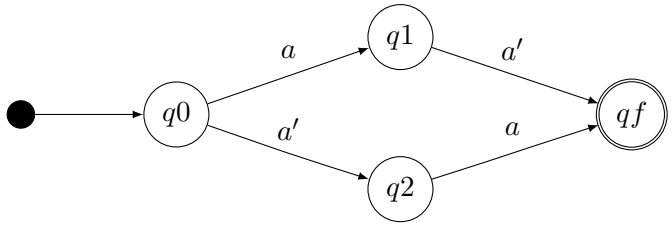


Fig. 2.1: Aútomata no deterministico

Esta dificultad responde a que para minimizar un mezclador determinístico no alcanza con minimizar de manera clásica el autómata determinístico asociado ya que podría haber una permutación de esas transiciones que den lugar a la misma traducción, y el autómata definido a partir de esas permutaciones sí pueda ser minimizado con el algoritmo clásico de minimización.

Entonces, el problema de minimizar un mezclador determinístico es encontrar otro que compute la misma función pero que tenga una cantidad de estados mínima.

2.2. Nuestro Algoritmo de Minimización

Damos a continuación un algoritmo que transforma un mezclador en otro irreducible que reconoce el mismo lenguaje.

La idea del algoritmo es reordenar caminos que calculan la misma traducción para poder unificarlos. Para esto es necesario primero expandir algunos caminos para poder llevar el reorden más allá de los caminos simples preexistentes. La expansión se realiza para

ampliar al máximo las posibilidades de reorden. La necesidad de expandir el autómata no está relacionada con existencia de ciclos sino con la posibilidad de reordenar. Esto se puede ver en Figura 2.2, donde podría minimizar si se expande q_{12} y q_{13}.

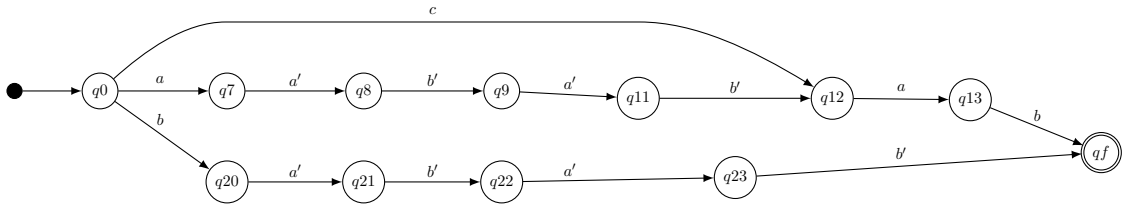


Fig. 2.2: Expandiendo q12 y q13 se podria reordenar y minimizar

Nuestro algoritmo consta de tres etapas: La primera transforma el mezclador en un autómata finito determinístico incompleto con un alfabeto ampliado para representar tanto a la transición como a la cinta de la que sería leída. Llamaremos a esta transformación T. No hace falta guardar el símbolo de la cinta de salida ya que en los *mezcladores* el símbolo que se escribe en la cinta de salida es igual a la entrada leída en alguna de las dos cintas. La primera etapa produce un autómata con la misma cantidad de estados que el que había en el mezclador original.

La segunda etapa es transformar el autómata determinístico que obtuvimos expandiendo sus caminos con el objetivo de encontrar más posibles caminos reordenables. La expansión es la única etapa que aumenta la cantidad de estados del autómata.

La tercera etapa es una iteración transformando el autómata hasta que no lo puede reducir más. En cada iteración : primero busca los estados con grado de entrada dos o más, revisa si algunos de los caminos simples que terminan en ellos se pueden juntar (reordenándolos de ser necesario), y el tal caso los junta. En el algoritmo esto se llama *pegado de caminos simples*. A continuación, el algoritmo normaliza los caminos simples de todo el autómata. Y finalmente calcula la minimización clásica de autómatas finitos determinísticos. Para esto el autómata primero se completa agregando un estado trampa que luego de minimizar se elimina.

El pegado de caminos simples compara todas las posibles factorizaciones de cada par de caminos que terminan en el mismo estado, en cambio la normalización determina una factorización para cada camino simple sin comparar con los demás, y esta factorización está incluida en las factorizaciones que compara el pegado de caminos. Es por eso que normalizar antes del pegado no aportaría nada, en cambio hacerlo después podría normalizar los nuevos caminos generados por el pegado.

Cada iteración de la tercera etapa reduce la cantidad de estados del autómata. El algoritmo termina cuando no puede hacer más reducciones.

Algoritmo 1: Algoritmo de minimización

Data: M - Mezclador

Result: Mezclador irreducible

Construir $A = T(M)$ (transformar de mezclador en AFD);

Expandir los caminos de A obteniendo A_1;

repeat

  Reordenar y pegar camino *simples* de A_1 obteniendo A_2;

  Normalizar caminos *simples* de A_2 obteniendo A_3;

  Correr el algoritmo clásico de minimización sobre A_3 obteniendo A_4;

  Hacer $A_1 = A_4$;

until *Autómata* A_1 *no cambie*;

Reconstruir un el traductor M' haciendo $M' = T^{-1}(A)$

Nota. Es importante destacar que no siempre puede alcanzar con una iteración porque puede ser necesario reordenar caminos que aparecen luego correr el algoritmo clásico de minimización y a su vez estos caminos no fueron minimizados en la primera iteración.

Un ejemplo de corrida del algoritmo: partiendo del mezclador Figura 1.2 que se transforma en:

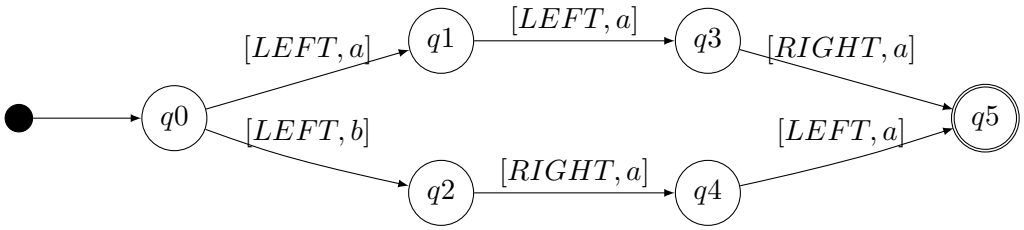


Fig. 2.3: Autómata finito determinístico resultante de aplicar la transformación T al mezclador de la Figura 1.2

Una minimización tradicional de AFD no conseguiría dejar un autómata más chico, sin embargo, si reordenamos algunas transiciones, manteniendo la traducción calculada:

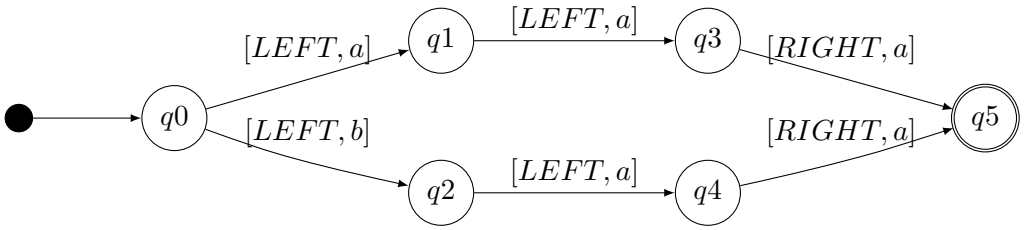


Fig. 2.4: Un *mezclador* no mínimo.

Aplicar el algoritmo clásico de minimización obteniendo el automata de la Figura 2.5

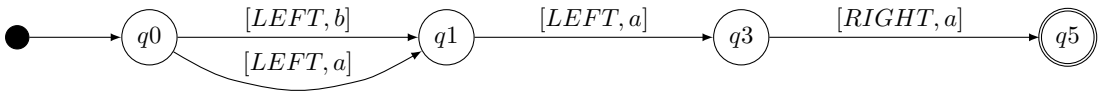


Fig. 2.5: Autómata minimizado

Y aplicando la transformación inversa a T, que llamamos T^{-1}, nos queda el mezclador de Figura 1.4

2.3. Transformación T (de mezclador a automata finito determinístico)

Definición 8. Definimos la transformación T que dado un *mezclador* M construye un autómata finito determinístico A codificando las distintas transiciones en un nuevo alfabeto.

Dado $M =< Q, \Sigma, \delta_M, \tau, q_i, F >$ construimos

$$A =< Q, \Sigma', \delta_A, q_i, F >$$

Donde

$$\Sigma' = \{(s, a) | s \in (l|r), a \in \Sigma\}$$

y

$$\delta_A(q_i, (l, a)) = q_j \ si \ \delta_M(q_i, a, \lambda) = \{q_j\}$$

$$\delta_A(q_i, (r, a)) = q_j \ si \ \delta_M(q_i, \lambda, a) = \{q_j\}$$

Nota. En las figuras usamos $\Sigma' = \{[s, a] | s \in \{LEFT, RIGHT\}, a \in \Sigma\}$

Es fácil ver que si calculamos la transformación inversa T^{-1} para volver a obtener un *mezclador* la relación calculada será la misma que antes de aplicar T.

2.4. Expansión

La expansión es una función que toma un AFD A y define un AFD $E(A)$ que reconoce el mismo lenguaje que A. En la expansión cada estado de A con grado de entrada mayor a 1 se reemplaza por nuevos estados, uno para cada transición de entrada, y estas transiciones se conectan una a una con estos estados, por ejemplo el autómata 2.6

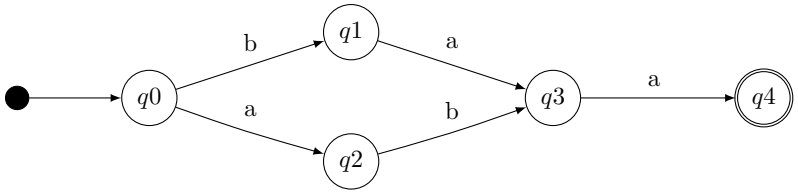


Fig. 2.6: AFD sin ciclos

Se expande como

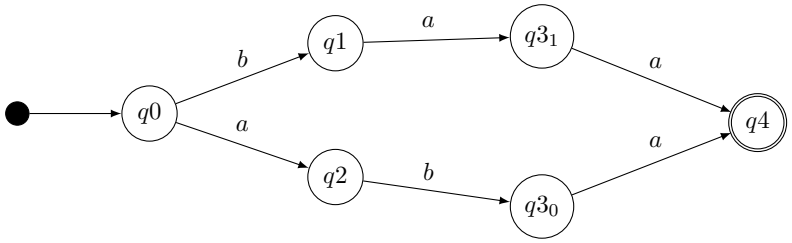


Fig. 2.7: Expansión de AFD de la Figura 2.6

No expandimos el estado inicial porque perderíamos el determinismo y no expandimos estados finales ya que no tendría sentido el reorden antes y después de un final porque cambiaría el resultado de la traducción.

Para evitar que el autómata se expanda indefinidamente no expandimos los nodos provenientes de otra expansión, por ejemplo el AFD de la Figura 2.11

La expansión agranda el AFD para que el pegado pueda achicar. Por ejemplo en el autómata de la Figura 2.8 no se pueden reordenar los caminos más allá de q_6 para juntarlos con el camino que va de q_8 a q_f, sin embargo, expandiendo como se ve en la Figura 2.9 quedaran tres caminos independientes y con la posibilidad de reordenarlos según sea más conveniente.

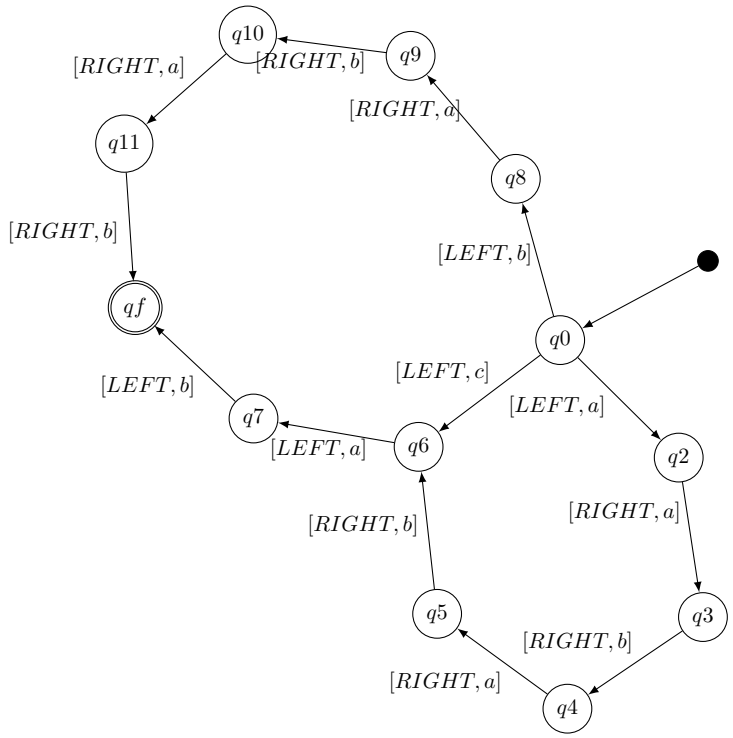


Fig. 2.8: Autómata expansible

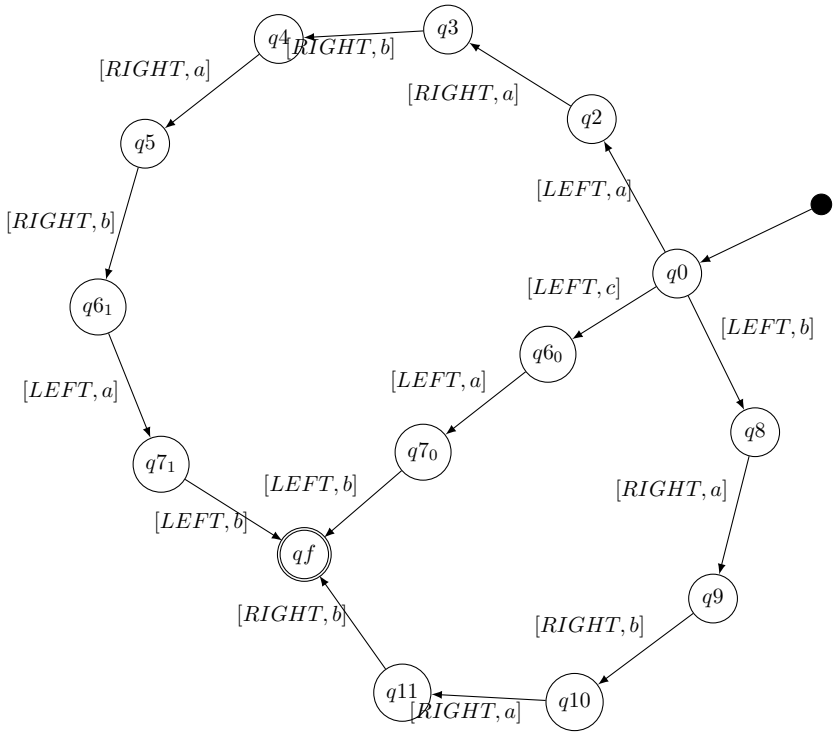


Fig. 2.9: Expansión de del autómata de la Figura 2.8

Escribimos $d_{in}(q)$ y $d_{out}(q)$ para el grado de entrada y el grado de salida del estado q.

Algoritmo 2: Expansión

Data: AFD $A =< Q, \Sigma', \delta, q_0, F >$
Result: AFD expandido
$visitados \leftarrow \{q_0\} \cup F$
while $Q \backslash visitados \neq \emptyset$ **do**
  Tomar un $q \in Q$ tal que $d_{in}(q) > 1$ y $d_{out}(q) > 0$
  Agregar q a $visitados$
  Para cada transición $p \xrightarrow{a} q$, crear un nuevo estado q_p y reemplazar las transiciones $p \xrightarrow{a} q$ por $p \xrightarrow{a} q_p$, si $q \in F$ entonces agregar q_p a F
  Para cada transición $q \xrightarrow{a} r$ eliminarla y reemplazarla por transiciones de la forma $q_p \xrightarrow{a} r$, para cada q_p agregado en el paso anterior
end

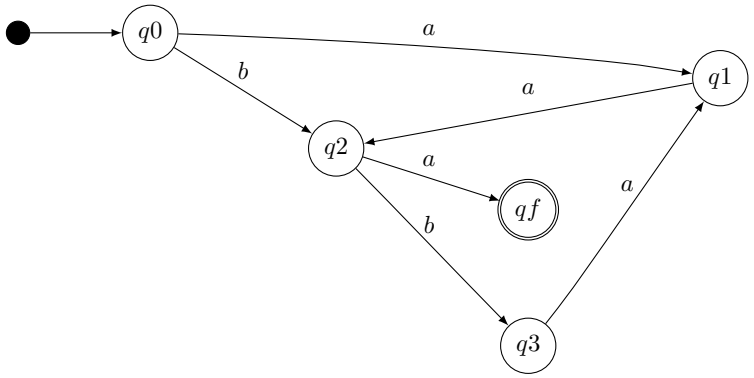


Fig. 2.10: AFD con ciclos

En el caso de que el autómata tenga ciclos tenemos que decidir una condición de corté porque sino la expansión sería infinita. Decidimos no volver a expandir un nodo que provenga de una expansión. Por ejemplo el AFD de la Figura 2.10 se expande como Figura 2.11.

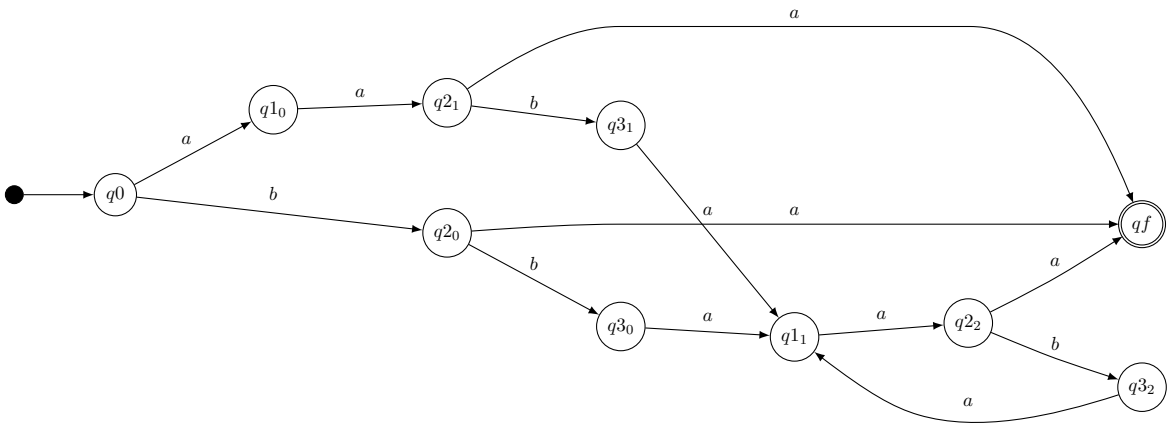


Fig. 2.11: Expansión de del AFD de la Figura 2.10

Nota. Hay que tener en cuenta que el el autómata resultante de la expansion depende del orden en que se analizan los estados para la expansión, otra expansión posible para el mismo autómata es la que se ve en Figura 2.12), esta vez empezando desde $q2$.

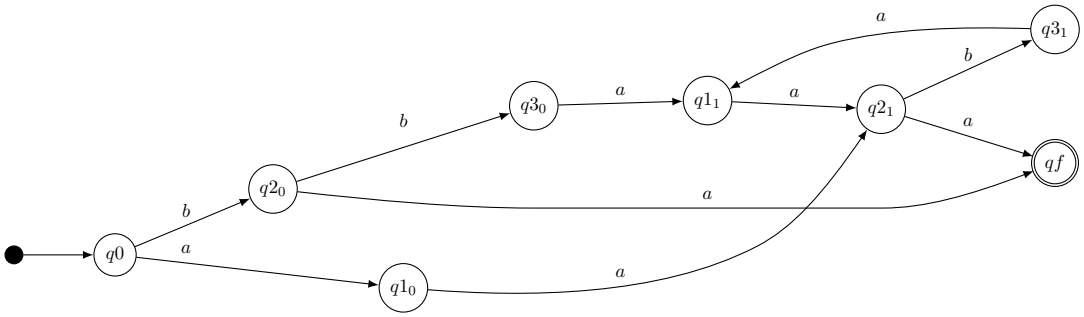
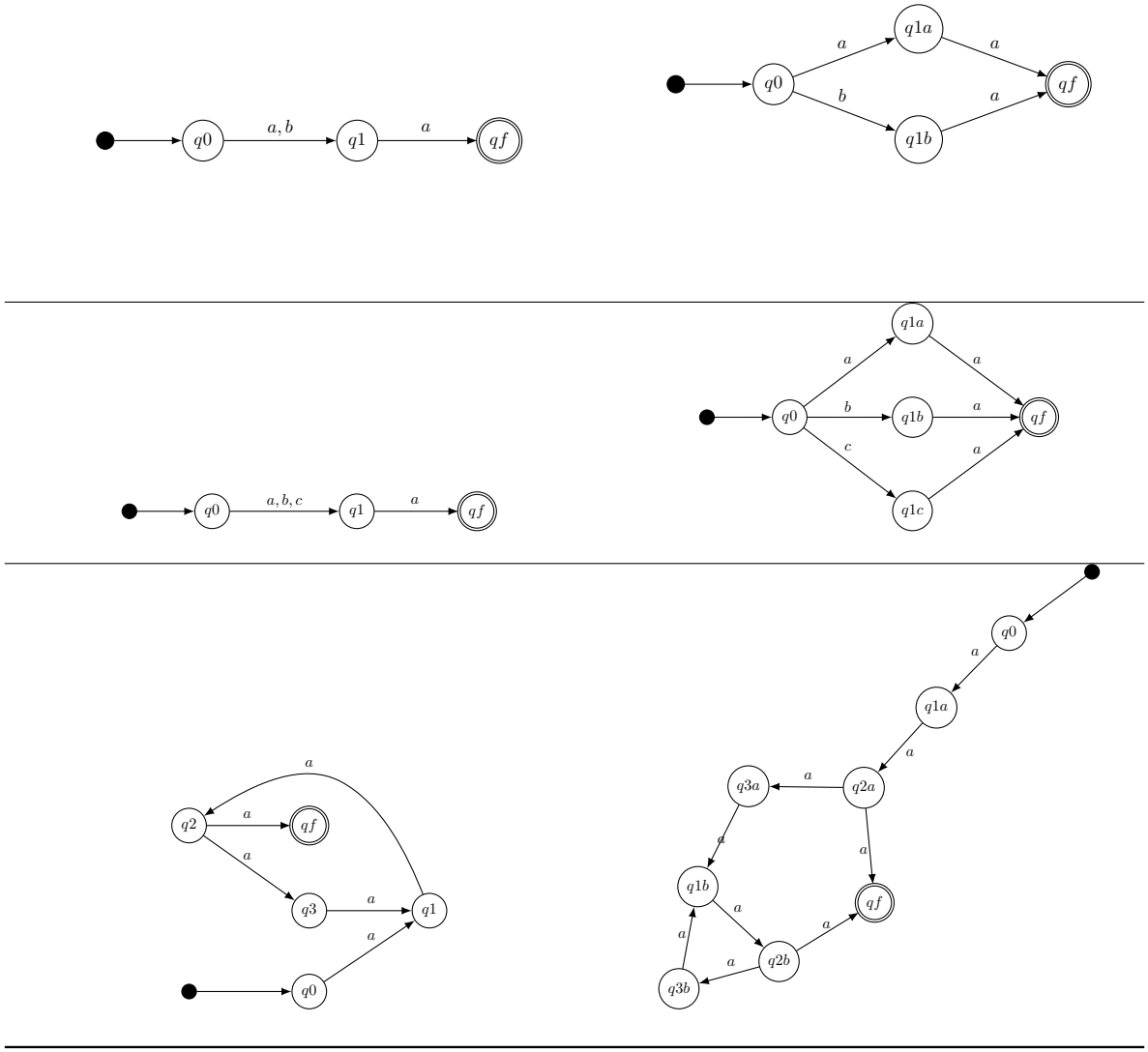


Fig. 2.12: Expansión de Figura 2.10 iniciando en q_2.

Por una cuestión de unicidad de los resultados, proponemos seleccionar los nodos a expandir en orden BFS, pero no parece ser una ventaja sobre cualquier otro orden.

Observar que que el autómata resultante de la expansion tiene a lo sumo $D * |Q|$ estados, donde D es el máximo grado de entrada de cada uno de los estado de Q.

La expansión de un autómata previamente expandido puede dar como resultado un autómata de mayor tamaño. Terminamos está sección con esta serie de ejemplos de expansiones:



2.5. Pegado de caminos simples

El pegado de caminos simples es lo que permite transformar el autómata en otro que computa la misma función pero con menos estados. Para esto buscamos los caminos simples que terminan en el mismo estado y para cada par buscamos la factorización que nos permita reordenar dejando los caminos con sufijos idénticos lo más largo posible.

Necesitamos las siguientes definiciones:

Definición 9 (Camino)**.** Dado un autómata donde las transiciones poseen lado y símbolo, llamamos camino a la secuencia de transiciones $estado \xrightarrow{(lado,símbolo)} estado, \ldots, estado \xrightarrow{(lado,símbolo)} estado$ que corresponden a transiciones sucesivas en el autómata.

Por ejemplo en Figura 2.3 $q1, (left, a), q3, (right, a)q5$ es un camino. Cuando las etiquetas de los estados no nos interesen nos referiremos a camino como la sucesión de los pares (lado, símbolo).

Definición 10 (Camino simple)**.** Un camino $q_1, (s_1, a_1), q_2 \ldots, q_{n-1}, (s_{n-1}, a_{n-1}), q_n$ es simple si los estados $q1$ a q_{n-1} tienen grado de salida 1 y los estados q_2 a q_n tienen grado de entrada 1 y $\forall i = 1 \ldots n-1$ vale que $q_i \notin F$.

Definición 11 (Under)**.** Dado un camino C, $under(C)$ es la secuencia de símbolos sin tener en cuenta el lado

$$under((s_1, a_1)(s_2, a_2) \ldots (s_n, a_n)) = a_1 a_2 \ldots a_n$$

Definición 12 (Side)**.** Dado un camino C, $side(C)$ es el conjunto de lados sin tener en cuenta los símbolos.

$$side((s_1, a_1)(s_2, a_2) \ldots (s_n, a_n)) = \{s_1, s_2, \ldots, s_n\}$$

como solo hay dos posibles valores para s_i vale que $|side(\alpha)| \leq 2$

Definición 13 (Reordenamiento de caminos)**.** Un camino simple C' es un reorden válido de un camino simple C si $under(C) = under(C')$ y $\hat{\tau}_C = \hat{\tau}_{C'}$ (calculan la misma traducción).

Por ejemplo, un posible reorden para el camino simple

$$\xrightarrow{(r,a_1)} \xrightarrow{(r,a_2)} \ldots \xrightarrow{(r,a_j)} \xrightarrow{(l,a_1)} \xrightarrow{(l,a2)} \ldots \xrightarrow{(l,a_j)}$$

es reescribir respetando el orden relativo para los pares provenientes de la misma cinta (r o l), pero escribiendo primero la subcadena que viene de la cinta l y luego la de r.

$$\xrightarrow{(l,a_1)} \xrightarrow{(l,a2)} \ldots \xrightarrow{(l,a_j)} \xrightarrow{(r,a_1)} \xrightarrow{(r,a2)} \ldots \xrightarrow{(r,a_j)}$$

Definición 14 (Factorización)**.** Dado un camino simple $C = (s_1, a_1)(s_2, a_2) \ldots)(s_m, a_m)$, decimos que $\alpha, \beta_1, \beta_2, \ldots, \beta_n$, con $|\alpha| \geq 0$ y $\forall i, |\beta_i| > 0$ es una factorización de C si $C = \alpha\beta_1\beta_2 \ldots \beta_n$ y vale que $\forall i, 1 \leq i \leq n$, $under(\beta_i) = under(\beta_1) \wedge |side(\beta_i)| = 1$.

Ej: el camino

$$\xrightarrow{(l,c)} \xrightarrow{(r,c)} \xrightarrow{(l,a)} \xrightarrow{(l,b)} \xrightarrow{(r,a)} \xrightarrow{(r,b)} \xrightarrow{(r,a)} \xrightarrow{(r,b)}$$

admite la factorización $\alpha = (l, c)(r, c)$, $\beta_1 = (l, a)(l, b)$, $\beta_2 = (r, a)(r, b)$ y $\beta_3 = (r, a)(r, b)$.

Notar que hay muchas posibles factorizaciones para un camino, dependiendo no solo de $under(\beta_i)$ sino también de la longitud de α. Llamamos a este conjunto: *Conjunto de factorizaciones de un camino simple.*

Definición 15 (m-pegable)**.** Dados dos caminos simples C_1, C_2 y sus respectivas factorizaciones F_{C_1}, F_{C_2} decimos que son $m - pegables$ si teniendo en cuenta las factorizaciones es posible *reordenar* los caminos simples C_1 y C_2 tal que se obtenga un sufijo común a ambos de longitud m.

Por ejemplo si tenemos los caminos que se ven en Figura 2.13 podemos reordenar los factores *ab* del primer camino para obtener dos caminos con sufijo indistinguible, Figura 2.14. Los caminos serán entonces 2-pegables.

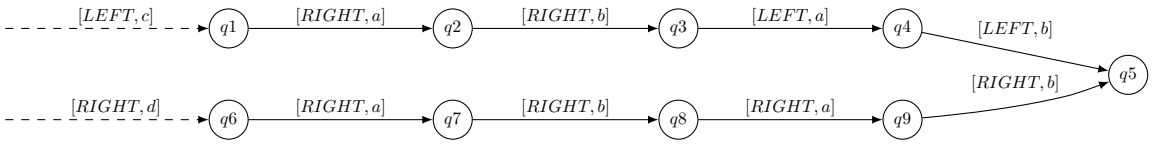


Fig. 2.13: Dos caminos *2-pegables*

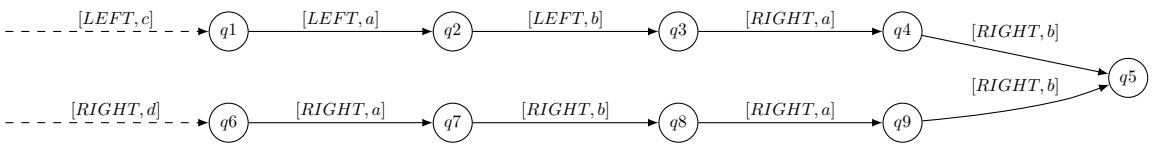


Fig. 2.14: Caminos de Figura 2.13 reordenados

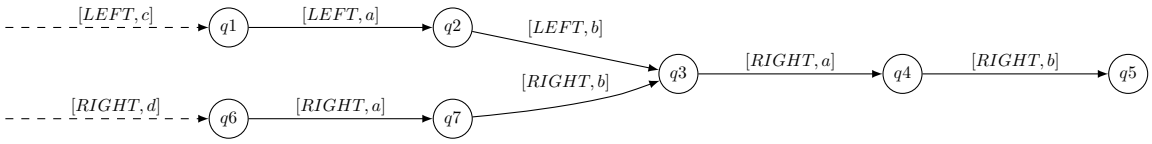


Fig. 2.15: Caminos de Figura 2.13 pegados

Algoritmo 3: Pegado de caminos simples

Data: AFD $A =< Q, \Sigma, \delta, q_0, F >$

Result: AFD A modificado

while *repetir hasta que A no cambie* **do**

  Hacer $Q_{candidato} \leftarrow \{q \in Q / d_{in}(q) > 1\}$

  while $Q_{candidato} \neq \emptyset$ **do**

    Tomar q_c como un elemento de $Q_{candidato}$

    Hacer $C \leftarrow \{$camino simple terminado en $q_c\}$

    forall $c \in C$ **do**

      computar F_c (conjunto de factorizaciones)

    end

    Encontrar $f \in F_{c_i}, g \in F_{c_j}, c_i, c_j \in C, i \neq j$ (dos factorizaciones de dos caminos distintos) $m - pegables$ con el máximo m posible.

    Reordenar c_i y c_j para lograr $m - pegado$

    Fusionar sufijo común en autómata A

    Si $d_{in}(q_c) = 1$ o ninguno par de caminos terminados en q_c son $m - pegables$ (para alguna factorización), sacarlo de $Q_{candidato}$

  end

end

2.6. Normalización

Es posible realizar un reordenamiento de caminos simples para que estados que en el autómata son distinguibles (de acuerdo a la relación de equivalencia de Nerode) pasen a ser indistinguibles y por lo tanto se pueda aplicar la minimización clásica. Para resolver esto agregamos una etapa de *normalización* donde los caminos simples son reescritos de forma *normalizada*, es decir primero todos los factores de un lado y luego todos los del otro, teniendo en cuenta el lado del primer estado del camino, si este tuviera grado de salida mayor a uno, o primero el lado izquierdo y luego el derecho en otro caso. De existir más de una factorización se elije una que minimiza la longitud de α y posee al menos dos ocurrencias de β.

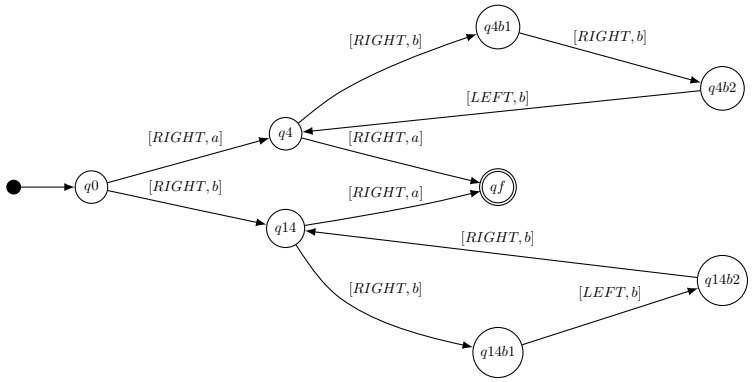


Fig. 2.16: Un *mezclador* con ciclos, sin normalizar.

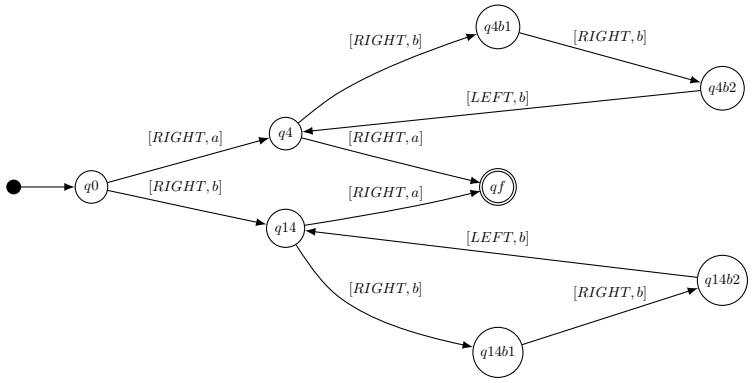


Fig. 2.17: El *mezclador* de la Figura 2.16 luego de normalizarlo.

Algoritmo 4: Normalizacion

Data: AFD $A =< Q, \Sigma, \delta, q_0, F >$

Result: AFD normalizado

forall *c camino simple* **do**

computar F_c (conjunto de factorizaciones, donde cada factorización es de la forma $\alpha\beta_1\beta_2 \dots \beta_n$ y vale que para todo $i, j, under(\beta_i) = under(\beta_j)$

buscar $f \in F_c, f = \alpha\beta_1\beta_2 \dots \beta_n$ tal que $|\alpha|$ sea mínimo.

Reordenar C según f dejando primero todos los factores con *side* igual al primer estado del camino (sin tener en cuenta el camino para calcular el lado del estado). Si el primer estado del camino tiene grado de salida 1 se normaliza dejando primero los factores de lado izquierdo y luego los de lado derecho.

end

2.7. Minimización clásica de Moore de AFD

Como última etapa, en cada iteración realizamos una minimización clásica de autómatas finitos deterministicos ya que el pegado de caminos simples y en la normalización pueden haber dejado estados indistinguibles y por lo tanto el autómata resultante quedará aun más chico.

El algoritmo más simple de minimización es el de Moore [7], computa la equivalencia de Nerode haciendo un refinamiento iterativo desde una equivalencia inicial. Todos los autómatas se asumen determinísticos. Aunque en el peor caso es de orden cuadrático, es el caso promedio es $O(n \log n)$, ver [4] . El algoritmo de minimización de un autómata finito determinístico dado en Hopcroft [5] tiene un orden en peor caso de $O(n \log n)$. Es, al menos por ahora, el algoritmo más eficiente para el caso general. Los algoritmos de minimización clásicos asumen que la función de transición es completa. Recientemente fue extendido para autómatas finitos determinísticos incompletos [10, 2]. No se conoce el orden promedio para el algoritmo de Hopcroft.

Algoritmo 5: Algoritmo de cálculo de equivalencia de Nerode

Data: AFD $M =< Q, \Sigma, \delta, q_0, F >$
Result: $Q/\equiv$
$P = \{Q\}$;
$i = 0$;
while $\left(P \neq \bigcup_{X \in P} X/\overset{i}{\equiv}\right)$ **do**
  $P = \bigcup_{X \in P} X/\overset{i}{\equiv}$; $i = i + 1$;
end
return P;

Sea $M =< Q, \Sigma, \delta, q_0, F >$ un AFD sin estados inaccesibles y con función de transición completa.

El AFD mínimo equivalente $M_{min} =< Q_{min}, \Sigma, \delta_{min}, q_{min_0}, F_{min} >$ es

$$
\begin{aligned}
Q_{min} &= (Q \,/\, \equiv) \text{ (las clases de equivalencia de } \equiv) \\
\delta_{min}([q], a) &= [\delta(q, a)] \\
q_{min_0} &= [q_0] \\
F_{min} &= \{[q] \in Q_{min} : q \in F\}
\end{aligned}
$$

3. CORRECTITUD

Consideramos en esta sección la correctitud de nuestro algoritmo de minimización dado en Algoritmo 1.

3.1. El mezclador resultante reconoce el mismo lenguaje

Proposición 1. *La expansión de un AFD preserva el lenguaje aceptado.*

Demostración. Sea A el AFD dado y llamemos A' al AFD expandido resultante. Debemos ver que para cada camino $q_0 \xrightarrow{a_1} q_1 \dots q_{n-1} \xrightarrow{a_{n-1}} q_f$ en el autómata original existe exactamente un camino en el autómata expandido

$$p_0 \xrightarrow{a_1} p_1 \dots p_{n-1} \xrightarrow{a_{n-1}} p_f.$$

Probaremos que para cada cadena de Σ^* que representa un camino desde q_0 en A, existe un camino desde p_0 en A' por inducción en la longitud de la cadena. El camino no necesariamente termina en un estado final.

1. Si $long(\alpha) = 0$ entonces existe un un camino de longitud 0 en A' ya que el estado inicial no se expande por definición
2. Tomemos como hipótesis inductiva que vale para α que de longitud n. Debemos ver que vale para longitud $n+1$. Dada la cadena αa que es camino desde q_0 en A, por hipótesis inductiva vale que existe los estados p_0 a p_n corespondientes a la expansión de los estados q_0 a q_n que consumen α. Si existia una transición de q_n a q_j en A entonces existe una transición de p_n a p_j donde p_j proviene de la expansión de q_j por construcción de la expansión. Por lo tanto si existe en A un camino que consume αa entonces existe en A' un camino que consume αa.

Dado que los estados finales no se expanden si un camino termina en un estado final de A también va a terminar en un estado final de A', es decir que todas la cadenas aceptadas por A van a ser aceptadas por A'.

Para probar que cada cadena aceptada por A' es aceptada por A se prueba de manera similar pero teniendo en cuenta que cada estado de A', por definición del algoritmo de expansión, proviene de un estado en A que preservan los símbolos consumidos entre ellos. □

Proposición 2. *Sea T la transformación de Mezclador a AFD de la Definición 8. Dado un mezclador determinístico M, $T^{-1}(Expansión(T(M))$ es un mezclador que calcula la misma traducción.*

Demostración. Dejaría de ser un mezclador si un mismo estado tuviera transiciones salientes con distinto lado. Dado que las transiciones que se agregan en la expansión de cada estado poseen el mismo lado que las transiciones que tenía dicho estado antes de la expansión vale que los estados expandidos preservan el lado del estado que les dio origen. Dado que la transformación T es un isomorfismo (renombre) entre *mezclador* y AFD y dado que en la proposición **??** se probó que la expansión preserva el lenguaje es trivial ver que calcula la misma traducción. □

Proposición 3. *El pegado de caminos simples preserva el lenguaje aceptado.*

Demostración. El pegado de caminos simples consiste en pegados sucesivos de pares de caminos. Por definición el pegado consiste en reordenar los caminos respetando una factorización de manera que se obtenga un sufijo común (sean $m-pegables$) y luego fusionarlos. Por definición el reorden de un camino preserva la cadena consumida de cada lado. Si los caminos reordenados tienen un sufijo común es trivial ver que se puede transformar el autómata para fusionar de manera que compartan el camino correspondiente a tal sufijo preservando las cadenas consumidas de cada lado. El pegado de caminos no modifica los estados finales, luego las cadenas consumidas en el autómata original terminaran en un estado final en el autómata resultante del pegado si y solo si terminaban en un estado final en el autómata original. □

Proposición 4. *El pegado de caminos simples de un autómata proveniente de un mezclador resulta en un autómata que puede ser transformado en otro mezclador.*

Demostración. Es evidente porque el pegado de caminos no agrega transiciones de salida a ningún estado. □

Proposición 5. *Dado un* mezclador M, $T^{-1}(Normalizar(T(M)))$ *preserva el lenguaje calculado.*

Demostración. La normalización consiste en reordenar cada camino simple de manera que se preserve la cadena consumida de cada lado. Es fácil ver que esto ocurre porque para reordenar se busca una factorización del camino tal que se pueda escribir como $\alpha\beta_1\beta_2\ldots\beta_n$ y vale que para todo $i, j, under(\beta_i) = under(\beta_j)$ y el reorden se realiza entre los distintos β_i, como los $under(\beta_i)$ son iguales para todo i, la cadena consumida de cada lado no va a verse modificada. □

Proposición 6. *Da un* mezclador M, $T^{-1}(Normalizar(T(M)))$ *sigue siendo un* mezclador.

Demostración. Dado que solo se modifica el lado de nodos con grado de entrada uno no es posible que la normalización agregue no determinismo. □

Proposición 7. *Completar el autómata para minimizar y luego eliminar el estado trampa y sus transiciones preserva el mezclador.*

Demostración. Sea q_t el estado *trampa* utilizado para completar el autómata. Supongamos que luego de la minimización existe un estado q_i y dos símbolos a, b tal que $\delta(q_i, (a, l)) \neq q_t$ y $\delta(q_i, (b, r)) \neq q_t$ entonces existen dos posibilidades. Una es que las transiciones ya existían desde q_i antes de minimizar, con lo cual no partimos de un mezclador. La otra posibilidad es que q_i sea el resultado de juntar dos estados indistinguibles, pero entonces ambos tenían que aceptar la cadena que comenzaba por (a, l) y la que comenzaba por (b, r). Absurdo que viene de suponer que el autómata original era un mezclador. □

El resultado de aplicar el Algoritmo 1 sigue siendo un mezclador. Es fácil verlo porque el resultado de cada etapa sigue siendo mezclador. También es fácil ver que el algoritmo termina, cada etapa termina y en cada iteración o bien el autómata no cambia, en cuyo caso se termina el ciclo o bien el autómata resultante tiene menos estados.

Teorema 1. *El Algoritmo 1 transforma un mezclador en otro irreducible que realiza la misma traducción.*

3.2. ¿Es mínimo?

Proposición 8. *La cantidad de estados que se obtienen luego de aplicar el Algoritmo 1 al mezclador M es menor o igual a la cantidad que se obtiene aplicando la minimización clásica de AFD sobre el autómata transformado $T(M)$.*

Demostración. El algoritmo termina con un autómata que ya no puede continuar reduciendo. Veamos la cantidad de estados es menor o igual a la que obtiene el algoritmo clásico de minimización de AFD. La expansión es la única etapa que agrega estados. Notemos que si las otras etapas no actúan la minimización clásica reconstruiría el autómata original (antes de expandir), ya todos los estados provenientes de la expansión de un estado son indistinguibles entre ellos. La cantidad de clases de equivalencia de la relación de indistinguibilidad del autómata expandido coincide con las del autómata original.

El pegado de caminos compara las factorizaciones posibles de cada par de caminos simples que terminan en el mismo nodo. Si la minimización hubiera podido fusionarlos quiere decir que hay una factorización y reorden que consiste en no modificar estos caminos y por lo tanto el pegado de caminos los fusionaría. Si en cambio el pegado de caminos reordena dos caminos para fusionarlos entonces la minimización no hubiera podido juntarlos. De todos los caminos simples que terminan en un mismo nodo el pegado selecciona el par de caminos que más estados logra fusionar. Si hubiera una subgrafo indistinguible de este subgrafo en otra parte del autómata (que termina en otro estado) el pegado tomaría las mismas decisiones allí y por lo tanto seguirían siendo indistinguibles y la minimización clásica podría fusionarlos.

Si después de la normalización existen dos caminos simples que calculan la misma traducción y no se pudieron fusionar en la minimización entonces no podían hacerlo tampoco antes de normalizar ya que o bien los caminos quedan ordenados de la misma manera o bien tienen nodos iniciales de lados distintos, lo que los hace distinguibles.

Dado que cada etapa no afecta la minimización clásica haciendo que deje más estado se puede concluir que nuestro algoritmo no es peor que la minimización clásica. □

BIBLIOGRAFÍA

[1] Jean Berstel, Luc Boasson, Olivier Carton, and Isabelle Fagnot. Minimization of automata. In *Handbook Automata: from Mathematics to Applications*. European Mathematical Society, 2021.

[2] M.-P. Béal and M. Crochemore. Minimizing incomplete automata. In B. J. Piskorski, W. Watson, and A. Yli-Jyra editors, editors, *Proc. 7th Int. Workshop on Finite-State Methods and Natural Language Processing*. Ispra, 2008.

[3] C. Choffrut. Minimizing subsequential transducers: a survey. *Selected papers in honor of Jean Berstel. Theoretical Computer Science*, 292:131–143, 2003.

[4] F.Bassino, J.David, and C.Nicaud. On the average complexity of moore's state minimization algorithm. In S. Albers and editors J.-Y. Marion, editors, *STACS 2009, Proc. 26th Symp. Theoretical Aspects of Comp. Sci.*, volume 09001 of Dagstuhl Seminar Proceedings, pages 123–134. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009.

[5] J.E.Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z.Kohavi and editors A. Paz, editors, *Theory of machines and computations*, pages 189–196. Academic Press, 1971.

[6] Mehryar Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, 2000.

[7] E. F. Moore. Gedanken experiments on sequential machines. In C. E. Shannon and editors J. Mc-Carthy, editors, *Automata Studies*, pages 129–153. Princeton Universty Press, 1956.

[8] V. Becher N. Alvarez and O. Carton. Finite-state independence and normal sequences. *Journal of Computer and System Sciences*, 103:1–17, 2019.

[9] J.-E. Pin. *Relational morphisms, transductions and operations on languages*, pages 34–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 1989.

[10] A. Valmari and P. Lehtinen. Efficient minimization of dfas with partial transition. In S. Albers and editors P. Weil, editors, *STACS 2008, Proc. 25th Symp. Theoretical Aspects of Comp. Sci.*, volume 08001 of Dagstuhl Seminar Proceedings, pages 645–656. Schloss Dagstuhl - Leibniz- Zentrum für Informatik, 2008.