Estudio de la Mezcla de Estados Determinista y No Determinista en el Diseño de Algoritmos para Inferencia Gramatical de Lenguajes Regulares

Gloria Inés Alvarez Vargas
Director: Dr. D. Pedro García Gómez

Tesis Doctoral.

Programa Doctoral en Reconocimiento de Formas e Inteligencia Artificial
Departamento de Sistemas Informáticos y Computación, DSIC
Universidad Politécnica de Valencia

Valencia, Octubre de 2007

Resumen

Esta investigación aborda el tema del diseño de algoritmos de inferencia gramatical para lenguajes regulares, particularmente en lo relacionado con la mezcla de estados como elemento fundamental del proceso de inferencia. Se estudia la mezcla de estados en sus variantes determinista y no determinista desde el punto de vista teórico. Como resultado se propone una manera eficiente de realizar la mezcla de estados no determinista y se demuestra que la inferencia gramatical de lenguajes regulares basada en la mezcla de estados (tanto determinista como no determinista) converge en el límite independientemente del orden en que se realizan las mezclas. La demostración es de interés ya que entre otras consecuencias, permite afirmar la convergencia en el límite de la estrategia EDSM (Evidence Driven States Merging) que es ampliamente conocida en la literatura como un heurísico. Dado que la demostración considera también la inferencia de autómatas no deterministas, el resultado abre la puerta al desarrollo de algoritmos convergentes que infieren autómatas no deterministas.

El aspecto experimental de esta investigación propone un conjunto de algoritmos de inferencia gramatical para lenguajes regulares, todos ellos convergentes en el límite. Estos algoritmos surgen de aplicar diferentes variantes de mezcla de estados determinista y no determinista; ellos buscan aprovechar la información que se puede obtener a partir de las relaciones de inclusión entre los lenguajes por la derecha asociados a los estados de todo autómata. Se proponen cuatro algoritmos que hacen mezcla determinista y dos que hacen mezcla no determinista de estados. Los resultados obtenidos al comparar estos nuevos algoritmos con algoritmos de referencia como RPNI, red-blue o DeLeTe2 muestran que se logra disminuir significativamente el tamaño de las hipótesis que se producen, al tiempo que se consiguen tasas de reconocimiento comparables o ligeramente inferiores. También se han obtenido algunas mejoras en la complejidad temporal de los algoritmos nuevos con respecto a los de referencia.

Abstract

This research deals with the design of new grammatical inference algorithms for regular languages; of particular interest are strategies using state merging as their main feature. Both deterministic and non-deterministic state merging variations have been studied from a theoretical point of view. As a result, an efficient non-deterministic merging states method has been proposed; besides, it has been proved that grammatical inference for regular languages based on state merging (both deterministic and non-deterministic) converges in the limit independently of the order in which the merging is done. This proof is very interesting, since among other reasons, it allows to state the convergence in the limit of the EDSM (Evidence Driven States Merging) strategy, which is broadly known in the literature as a heuristic. Given that the proof also considers non-deterministic merging, this result may be effective in the design of new convergent inference algorithms for non-deterministic automata.

On the practical side, this research proposes several grammatical inference algorithms for regular languages, all of which converge in the limit. These algorithms explore both deterministic and non-deterministic state merging methods and take advantage of the information obtained from inclusion relations between derivatives of each state of an automaton. Six new algorithms are proposed; four of them yield deterministic automata while the remaining two produce non-deterministic automata. The results of experimental comparisons between the new algorithms and well known reference ones like RPNI, red-blue or DeLeTe2, show that the proposed algorithms are able to get significantly smaller hypothesis than its counterparts while the recognition rates are comparable or slightly lower. Furthermore, for some of the proposed algorithms, these results are obtained using methods of a lower temporal complexity than those used in the reference algorithms.

Resum

Aquesta investigació aborda el tema del diseny d'algoritmes d'inferència gramatical per a llenguatges regulars, particularment el relacionat amb l'agrupament d'estats com element fonamental del procés d'inferéncia. Des del punt de vista teòric, s'estudia la fusió d'estats en les seues variants determinista y no determinista. Com a resultat d'aquest estudi es proposa una manera eficient de realitzar l'agrupament no determinista d'estats i es demostra que la inferència gramatical de llenguatges regulars basada en la fusió d'estats, tant determinista com no determinista, convergeix en el límit independentment de l'ordre en que es realitcen les fusions. La demostració es d'interés ja que entre altres conseqüéncies, permet afirmar la convergència en el límit de l'estratégia EDSM (Evidence Driven States Merging) que es coneguda en la literatura com un heurístic. Donat que la demostració considera també la inferéncia d'autómates no deterministes, el resultat obri la porta al desenvolupament d'algoritmes convergents que infereixen autómates no deterministes.

L'aspecte experimental d'aquesta investigació proposa un conjunt d'algoritmes d'inferéncia gramatical per llenguatges regulars, tots ells convergents en el límit. Aquestos algoritmes sorgeixen al aplicar diferents variants d'agrupament determinista o no determinista d'estats; els algoritmes busquen aprofitar l'informació que es pot obtindre a partir de les relacions d'inclusió entre els llenguatges per la dreta asociats als estats de tot autómata. Es proposen quatre algoritmes que realitcen fusió determinista i dos que realitcen fusió no determinista d'estats. Els resultats obtesos al comparar aquests nous algoritmes amb altres de referéncia com RPNI, red-blue o DeLeTe2 mostren que s'aconsegueix reduir significativament la mida de les hipótesis que es produeixen, al mateix temps s'aconseguix tases de reconeixement comparables o lleugerament inferiors. També s'han obtes millores en la complexitat temporal dels nous algoritmes respecte als de referéncia.

Agradecimiento

Quiero agradecer a la Pontificia Universidad Javeriana de Cali por haberme brindado la posibilidad de venir a Valencia a estudiar y a los miembros del grupo en Teoría de Lenguajes Computabilidad y Criptografía y en especial a mi director el Doctor Pedro García porque a su lado he aprendido mucho durante estos años. En lo personal, agradezco a mis seres queridos por su constante apoyo y por creer en mí. Pero sobre todo, agradezco a la Vida, a la Fuerza que nos anima a todos los seres humanos, por haber tejido con habilidad la madeja de las circunstancias para traerme hasta el presente y haberme concedido el disfrutar y padecer esta enriquecedora experiencia personal.

Índice general

Índice de figuras

Índice de cuadros

Índice de algoritmos

Introducción

El problema de la inferencia gramatical consiste en recibir como entrada cadenas de símbolos etiquetadas como pertenecientes (positivas) o no (negativas) a un lenguaje formal y producir como salida una representación del lenguaje formal que etiquetó dichas cadenas. El diseño de algoritmos de inferencia gramatical, que es el tema fundamental en esta investigación, pretende desarrollar métodos eficaces y eficientes para realizar dicho proceso. La característica más importante de un algoritmo de inferencia es su convergencia en el límite; es decir, su capacidad para identificar, luego de un número finito de intentos, en forma correcta y definitiva el lenguaje representado por las muestras. En la medida que el algoritmo recibe mayor cantidad de cadenas de entrada es más factible que realice la identificación exacta, cuando no lo logra el algoritmo propone una respuesta aproximada.

La inferencia gramatical es un problema que ha venido despertando mucho interés en la comunidad académica, debido no sólo a sus características teóricas sino a la amplia gama de posibles aplicaciones que tiene, de la Higuera [dlH05] menciona entre otras: la robótica, los sistemas de control, el reconocimiento estructural de patrones, la lingüística computacional, el modelamiento del habla, la traducción automática, la biología computacional, el manejo de documentos, la compresión de datos y la búsqueda de conocimiento en bases de datos (en inglés data mining).

Aunque ya existe mucho terreno adelantado gracias a años de estudio, todavía queda trabajo por hacer, sobre todo si se tiene en cuenta que el problema de encontrar el DFA más pequeño que reconoce un conjunto de muestras positivas y negativas es un problema NP-Completo y que en la práctica es necesario realizar la inferencia en el menor tiempo posible, necesitando la menor cantidad de datos posible y generando hipótesis tan pequeñas y exactas como sea posible.

Existen varias aproximaciones a la solución del problema de la inferencia gramatical; por ejemplo, se hace inferencia mediante mezcla de estados [OG92, LPP98]; se usan autómatas deterministas estocásticos [CO94]; se aplica aprendizaje activo, es decir, aquel que aporta información adicional al proceso de inferencia en la forma de un oráculo o maestro informado que participa en el proceso cuando el algoritmo así lo requiere [PH97, PH00, Ang87, Ang90, Ang92]; se incorpora información sobre el contexto del que provienen las muestras [CFKdlH04]; y también se usan técnicas de inteligencia computacional como redes neuronales, algoritmos genéticos [Dup94] o programación evolutiva [LR05].

Este trabajo centra su interés en el estudio de la mezcla de estados como elemento principal de los algoritmos de inferencia gramatical, considerando tanto la inferencia de modelos deterministas como no deterministas, buscando proponer algoritmos con el mejor desempeño sobre muestras de prueba y que produzcan hipótesis lo más pequeñas posible.

En el Capítulo 1 se describe en más detalle el problema y la hipótesis de investigación, se proponen definiciones básicas y la notación a utilizar, se presenta el modelo de aprendizaje y se describen los RFSA (Residual Finite State Automata) que son una clase de autómata no determinista que será utilizada intensivamente en esta investigación. Luego, en el Capítulo 2, se hace un recuento de los trabajos más importantes que se conocen en el ámbito de la inferencia gramatical de lenguajes regulares haciendo énfasis en los trabajos relacionados con algoritmos de mezcla de estados, también se reportan antecedentes en cuanto a modelos de aprendizaje y a la evaluación empírica de algoritmos de inferencia gramatical. A continuación, en el Capítulo 3 se describen los diferentes tipos de mezcla de estados que se han considerado en este trabajo y también se presenta la demostración de que la convergencia de un proceso de mezcla de estados es independiente del orden en que ellas se realizan, tanto en el caso determinista como no determinista. En el Capítulo 4 se describen algoritmos de inferencia gramatical, en la primera sección se presentan los algoritmos de referencia, ellos no sólo han servido como base para algunos de los nuevos algoritmos propuestos, sino que también se utilizan en la parte experimental como punto de referencia para evaluarlos; en las secciones posteriores se describen los algoritmos de inferencia gramatical que han surgido de este estudio. En el siguiente Capítulo, el 5, se describen los córpora utilizados para evaluar dichos algoritmos, los resultados de las experimentaciones y se analizan dichos resultados. Finalmente, en el Capítulo 6 se concluye y se proponen líneas de trabajo futuro.

Capítulo 1

El Problema de la Inferencia Gramatical

El problema de la inferencia gramatical se puede plantear como el de aprender una función desconocida a partir de algunos de los valores que produce. Más exactamente, consiste en construir, ojalá en forma eficiente, una representación de un lenguaje objetivo a partir de un conjunto muestras de entrada. El lenguaje objetivo es un lenguaje formal, definido sobre un alfabeto finito, que en este estudio será siempre un lenguaje regular. La entrada del problema está formada por un conjunto no vacío de muestras positivas y un conjunto de muestras negativas, que en la definición general puede ser vacío pero que en este estudio en concreto no lo será. Las muestras positivas son cadenas de símbolos del alfabeto del lenguaje, que pertenecen al lenguaje objetivo y las negativas cadenas que pertenecen al complemento de dicho lenguaje objetivo. La representación resultante de la inferencia puede expresarse de diversas maneras, en este estudio dicha representación será un autómata finito.

Este estudio asume el modelo de aprendizaje de Gold (ver Sección 1.2) y lo aplica al diseño de algoritmos de inferencia gramatical para lenguajes regulares, con el objetivo de lograr mejores tasas de acierto en muestras de prueba y menores tamaños de las hipótesis que los obtenidos con los mejores algoritmos actuales. Se procede bajo la hipótesis de que utilizar NFAs como modelo de representación, puede mejorar la velocidad de convergencia, la tasa de acierto y/o disminuir el tamaño de las hipótesis obtenidas durante el proceso de inferencia gramatical de algunas subclases de lenguajes regulares.

La hipótesis surge de un estudio inicial en el que se implementa una versión del algoritmo *RPNI* con mezcla no determinista, allí se logra constatar que la generación de hipótesis no deterministas abre nuevas posibilidades con respecto a la inferencia de autómatas deterministas. Algunos antecedentes cercanos, como los trabajos de Denis et. al. [DLT04], que muestran resultados exitosos, refuerzan el interés inicial. Esta investigación se interesa en particular por el estudio de los autómatas residuales y las relaciones de inclusión entre estados; esta teoría se aprovecha tanto para construir algoritmos de inferencia que obtienen autómatas deterministas como algoritmos que obtienen autómatas no

deterministas. A partir de estos algoritmos y de manera fundamentalmente experimental se busca validar o refutar la hipótesis.

Existen importantes resultados teóricos que establecen la dificultad de la tarea de la inferencia gramatical: Gold demostró que es posible identificar en el límite lenguajes regulares a partir de muestras positivas y negativas [Gol67]; en cambio si sólo se dispone de muestras positivas la identificación es imposible para cualquier clase super-finita de lenguajes [Gol67]. Para lenguajes en categorías más altas de la jerarquía de Chomsky no se garantiza la identificación [Gol67].

En cuanto a la complejidad de las soluciones para el problema de la inferencia gramatical, también se conocen resultados importantes: se puede encontrar un autómata determinista consistente con un conjunto de muestras positivas y negativas dadas en tiempo polinomial, pero encontrar el autómata determinista más pequeño posible es un problema NP-Completo [Gol67]. Buscar un autómata no determinista consistente con un conjunto de muestras positivas y negativas es un problema tratable, pero encontrar un autómata no determinista con el menor número de estados posible es de nuevo un problema NP-Completo [Gol67].

Aunque en la última década han aparecido algunos trabajos sobre la inferencia de lenguajes regulares mediante autómatas no deterministas (NFAs), este no es un tema en el que abunden las aportaciones. La escasez de trabajos se puede explicar en la mayor complejidad de este enfoque con respecto a la inferencia de autómatas deterministas (DFAs); y el hecho que a pesar de ello estén surgiendo propuestas se puede explicar en que los NFAs son un modelo de representación que puede llegar a ser exponencialmente más pequeño que los DFAs. A continuación se profundiza en los problemas que surgen en la inferencia de NFAs y en la importancia de buscarles una solución.

Se sabe que inferir una representación mínima de un lenguaje regular es NP-Completo, y también se sabe que existen algoritmos que, provistas ciertas condiciones en cuanto a la cantidad y calidad de las muestras de entrenamiento, pueden resolver dicho problema en tiempo polinomial, obteniendo como representación un DFA. Dichos algoritmos proceden iterativamente, parten de una hipótesis muy particular y la van generalizando en tanto las muestras de entrenamiento lo permitan. Este proceso se realiza de manera convergente, es decir, garantizando que después de un número finito de hipótesis equivocadas, se llega a la hipótesis correcta. Para que la convergencia tenga sentido, debe existir una representación "ideal" del lenguaje objetivo, hacia la cual debe fluir el proceso. En el caso de los DFAs, esta representación es el DFA mínimo, es decir, aquel autómata determinista que identifica el lenguaje objetivo y que tiene el menor número de estados; la teoría dice que cada lenguaje regular tiene asociado un único DFA mínimo. Al pasar a la inferencia de NFAs, y procediendo también de forma iterativa, se observa que no existe un único NFA mínimo asociado a cada lenguaje regular sino varios NFAs minimales diferentes entre sí, esto hace que no haya un único punto hacia el cual enfocar el proceso de convergencia. En resumen, se tiene que a la dificultad ya conocida de la inferencia en sí misma, se suma la dificultad de trabajar con un modelo que no facilita la convergencia.

Conocidas las dificultades, alguien podría preguntarse si vale la pena insistir

en la inferencia de NFAs, y efectivamente hay razones para hacerlo: se sabe que un mismo lenguaje puede ser representado mediante muchos DFAs (o NFAs); de todos ellos, normalmente interesa el más pequeño, el que tiene menor número de estados. El hecho que se obtenga la representación de tamaño mínimo no significa que dicha representación sea pequeña, y cuando estos tamaños crecen mucho, hacen que la inferencia no sea viable en la práctica, lo cual ocurre desafortunadamente en la mayoría de los problemas reales en los que teóricamente sería de utilidad aplicar inferencia. Dado un lenguaje regular, se sabe que el tamaño de los NFAs minimales que lo representan es menor o igual que el tamaño del DFA mínimo que lo representa; en ocasiones, el número de estados de un NFA minimal puede ser exponencialmente menor que el del mencionando DFA mínimo. Obtener representaciones precisas y a la vez pequeñas es una condición necesaria para poder aplicar la inferencia en problemas de tamaño real.

Una estrategia que se viene siguiendo en los últimos años para realizar inferencia de NFAs, consiste en proponer subclases de NFAs que posean propiedades que faciliten la inferencia. Por ejemplo, en los trabajos de Coste [CF00, CF03b, CF03a], se propone la subclase de los UFAs (Unambiguous Finite Automata) y Denis propone la subclase de los RFSAs (Residual Finite State Automata) [DLT02, DLT04, AGR06]. Estas aproximaciones han sido evaluadas experimentalmente y reportan mejor comportamiento que la inferencia de DFAs para ciertos tipos de lenguajes regulares. El mejor comportamiento puede significar: mayor precisión en la aproximación al lenguaje objetivo ó menor tamaño en las hipótesis obtenidas. Estas estrategias proponen algoritmos polinomiales, pero de mayor complejidad temporal que los ya existentes para la inferencia de DFAs.

Se sabe que existen lenguajes regulares cuya inferencia con NFAs no mejora los resultados obtenidos con DFAs, pero también se sabe que hay otros lenguajes regulares para los cuales es posible hacer mejoras importantes [DLT04, CF03a].

Durante el resto de este capítulo se presentan elementos teóricos básicos a los que se hará referencia en el resto del documento. En la sección 1.1 se presentan las definiciones fundamentales y algunos aspectos de notación; en la sección 1.2 se hace explícito el modelo de aprendizaje que sirve de base a este trabajo y en la sección 1.3 se presenta la teoría fundamental acerca de los autómatas finitos residuales (RFSA).

1.1. Definiciones Básicas

A continuación se proponen algunas definiciones importantes y también algunas explicaciones sobre la notación utilizada, para mayor detalle sobre definiciones básicas se puede ver [HMU01].

Sea Σ un alfabeto finito y sea Σ^* el monoide libre generado por Σ con la concatenación como operación interna y ε como elemento neutro. Un *lenguaje* L sobre Σ es un subconjunto de Σ^*. Los elementos de L se llaman *palabras*. La longitud de una palabra $w \in \Sigma^*$ se denota $|w|$. Dado $x \in \Sigma^*$, si $x = uv$ con $u, v \in \Sigma^*$, entonces u (resp. v) se denomina *prefijo* (resp. *sufijo*) de x. $\Pr(L)$

(resp. Suf(L)) denota el conjunto de prefijos (resp. sufijos) de L. El producto de dos lenguajes $L_1, L_2 \subseteq \Sigma^*$ se define como: $L_1 \cdot L_2 = \{u_1u_2 | u_1 \in L_1 \wedge u_2 \in L_2\}$. Algunas veces, $L_1 \cdot L_2$ se denotará simplemente como L_1L_2. A lo largo de este documento, el *orden lexicográfico* en Σ^* se denotará como $\ll$. Suponiendo que Σ está totalmente ordenado por $<$ y dados $u, v \in \Sigma^*$ con $u = u_1 \dots u_m$ y $v = v_1 \dots v_n$, $u \ll v$ si y sólo si ($|u| < |v|$) ó ($|u| = |v|$ y $\exists j$, $0 \leq j < n, m$ tal que $u_1 \dots u_j = v_1 \dots v_j$ y $u_{j+1} < v_{j+1}$).

Definición 1 *Un* Autómata No Determinista *(NFA por su sigla en inglés) es una 5-tupla $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ donde Q es el conjunto (finito) de estados, Σ es un alfabeto finito, $I \subseteq Q$ es el conjunto de estados iniciales, $F \subseteq Q$ es el conjunto de estados finales y δ es una función parcial que va de $Q \times \Sigma$ en 2^Q. La extensión de esta función a palabras también se denota δ, va de $2^Q \times \Sigma^*$ en 2^Q y se define en la Ecuación 1.1.*

$$\begin{aligned} \delta(\{q\}, \varepsilon) &= \{q\} \\ \delta(\{q\}, a) &= \delta(q, a) \\ \delta(Q', u) &= \cup\{\delta(\{q\}, u) \mid q \in Q'\} \\ \delta(\{q\}, ua) &= \delta(\delta(\{q\}, u), a) \end{aligned} \tag{1.1}$$

Donde $Q' \subseteq Q, a \in \Sigma, q \in Q, u \in \Sigma^*$. Una palabra x es aceptada por $\mathcal{A}$ si $\delta(I, x) \cap F \neq \emptyset$. El conjunto de palabras aceptadas por $\mathcal{A}$ se denota $L(\mathcal{A})$.

Definición 2 *Un* Autómata Determinista *(DFA por su sigla en inglés) es un NFA tal que $|I| = 1$ y $|\delta(q, a)| \leq 1$, $\forall q \in Q$, $\forall a \in \Sigma$. Un DFA se denota entonces como la tupla $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, donde Q es el conjunto (finito) de estados, Σ es un alfabeto finito, δ es una función de $Q \times \Sigma$ en Q, q_0 es el único estado inicial y $F \subseteq Q$ es el conjunto de estados finales.*

Definición 3 *Se dice que un NFA está* recortado *si y sólo si $\forall q \in Q, \exists w_1 \in \Sigma^*$ tal que $q \in \delta(q_0, w_1)$ y $\exists w_2 \in \Sigma^*$ tal que $\delta(q, w_2) \cap F \neq \emptyset$.*

Definición 4 *Un estado q es* alcanzable *mediante una palabra u si $q \in \delta(q_0, u)$.*

Definición 5 *Dos autómatas son* equivalentes *si ellos reconocen el mismo lenguaje.*

Definición 6 *Dado un autómata $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ y un conjunto $P \subseteq Q$, la* restricción *de $\mathcal{A}$ a P es el autómata $\mathcal{A}_P = (P, \Sigma, \delta', I', F')$, donde $I' = I \cap P$, $F' = F \cap P$ y δ' es la restricción de δ que va de $P \times \Sigma$ en 2^P.*

Definición 7 *Dado un conjunto finito de palabras D_+, el* árbol aceptor de prefijos de D_+ *se define como el autómata $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ donde $Q = Pr(D_+)$, $q_0 = \varepsilon$, $F = D_+$ y $\delta(u, a) = ua$, $\forall u, ua \in Q$.*

Definición 8 *La* derivada *de un lenguaje L por una palabra u, también llamada* lenguaje residual *de L con respecto a u, ó* lenguaje por la derecha *de L con respecto a u se denota $u^{-1}L$ y se define como $u^{-1}L = \{v \in \Sigma^* | uv \in L\}$. Un lenguaje residual $u^{-1}L$ es* compuesto *si $u^{-1}L = \cup_{(v^{-1}L \subsetneq u^{-1}L)} v^{-1}L$. Un lenguaje residual es* primo *si no es compuesto, es decir, si no puede construirse a partir de la unión de otros lenguajes residuales de L.*

Definición 9 *Si $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ es un NFA y $q \in Q$, se define el* lenguaje aceptado por $\mathcal{A}$ *desde el estado q como $L(\mathcal{A}, q) = \{v \in \Sigma^* | \delta(q, v) \cap F \neq \emptyset\}$.*

Definición 10 *El conjunto de los* Prefijos más cortos *de un lenguaje [OG92] $Pc(L) = \{w | w \in Pr(L), \nexists v : w^{-1}L = v^{-1}L, v \ll w\}$.*

Definición 11 *El* núcleo *de un lenguaje $N(L) = \{ua | ua \in Pr(L), u \in Pc(L)\} \cup \{\varepsilon\}$.*

Definición 12 *Sea $D \subset \Sigma^*$ finito. El* autómata maximal *para D es el NFA $AM(D) = (Q, \Sigma, \delta, I, F)$ donde $Q = \cup_{x \in D}\{(u, v) \in \Sigma^* \times \Sigma^* | uv = x\}$, $I = \{(\lambda, x) | x \in D\}$, $F = \{(x, \lambda) | x \in D\}$ y para $(u, av) \in Q$, $\delta((u, av), a) = (ua, v)$. De modo que $L(AM(D)) = D$.*

Definición 13 *Sea $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ un autómata, sea π una partición de Q y sea $B(q, \pi)$ el bloque de π que contiene q. El* autómata cociente *es $\mathcal{A}/\pi = (Q', \Sigma, \delta', I', F')$, donde $Q' = Q/\pi = \{B(q, \pi) | q \in Q\}$, $I' = \{B \in Q' | B \cap I \neq \emptyset\}$, $F' = \{B \in Q' | B \cap F \neq \emptyset\}$ y la función de transición es $B' \in \delta'(B, a)$ si y sólo si $\exists q \in B, \exists q' \in B'$ con $q' \in \delta(q, a)$.*

Definición 14 *Una* máquina de Moore *es una 6-tupla $M = (Q, \Sigma, B, \delta, q_0, \Phi)$, donde Q es el conjunto de estados, Σ (resp. B) es el alfabeto de entrada (resp. salida), δ es una función parcial que va de $Q \times \Sigma$ en Q, q_0 es el estado inicial y Φ es una función que va de Q en B llamada* función de salida*. Una máquina de Moore no determinista, se define de manera similar, excepto que δ va de $Q \times \Sigma$ en 2^Q y puede tener más de un estado inicial, siendo I el conjunto de ellos. El autómata asociado a una máquina de Moore $M = (Q, \Sigma, \{0, 1\}, \delta, I, \Phi)$ es $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ donde $F = \{q \in Q | \Phi(q) = 1\}$. La* restricción *de M a $P \subseteq Q$ es la máquina M_P definida análogamente al caso de los autómatas. El comportamiento de M está dado por la función parcial $t_M | \Sigma^* \rightarrow B$ definida como $t_M(x) = \Phi(\delta(q_0, x))$, para todo $x \in \Sigma^*$ tal que $\delta(q_0, x)$ está definida.*

Al mostrar en forma gráfica una máquina de Moore con alfabeto de salida $B = \{0, 1, ?\}$, el valor de la función de salida asociada a cada estado, se representa variando el grosor y la forma del borde del círculo que dibuja un estado x: una línea continua gruesa significa que el estado es de aceptación ($\Phi(x) = 1$), si la línea es continua pero delgada, el estado es de rechazo ($\Phi(x) = 0$), si la línea no es continua, significa que el valor de salida está indefinido ($\Phi(x) =?$). La Figura 1.1 ilustra esta convención de dibujo: los estados 1,4,6,7,9 tienen valor de salida 1, los estados 2, 3 y 8 tienen valor de salida 0 y el estado 5 tiene valor de salida indefinido.

Definición 15 *Dados dos conjuntos disjuntos de palabras D_+ y D_-, se define el (D_+, D_-)-árbol de* Prefijos de Moore*, denotado ($APM(D_+, D_-)$), como la máquina de Moore que tiene $B = \{0, 1, ?\}$, $Q = \Pr(D_+ \cup D_-)$, $q_0 = \varepsilon$ y $\delta(u, a) = ua$ si $u, ua \in Q$ y $a \in \Sigma$. Para todo estado u, el valor de la función de salida asociado a u es 1, 0 ó ? dependiendo si u pertenece a D_+, a D_- o a $\Sigma^* - (D_+ \cup D_-)$ respectivamente. El tamaño de la muestra (D_+, D_-) es $\sum_{w \in D_+ \cup D_-} |w|$.*

Definición 16 *Una máquina de Moore $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$ es* consistente *con (D_+, D_-) si $\forall x \in D_+$ se tiene que $\Phi(\delta(q_0, x)) = 1$ y $\forall x \in D_-$ se tiene que $\Phi(\delta(q_0, x)) = 0$.*

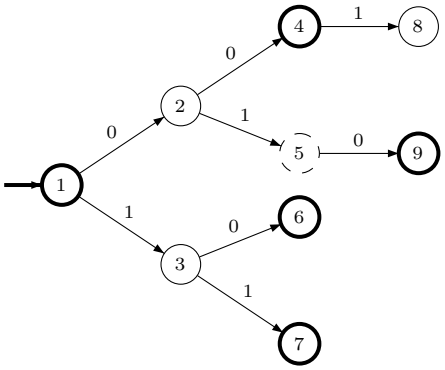


Figura 1.1: Ejemplo de la Notación Usada para Dibujar Máquinas de Moore

Definición 17 *Se entiende por* tamaño *de un autómata finito ó de una máquina de Moore, el número de estados* $|Q|$ *que posee.*

Definición 18 *Se dice que una muestra positiva* (D_+) *de un lenguaje regular* L *es* estructuralmente completa *si existe una partición* π *sobre los estados del árbol aceptor de prefijos de* D_+ *tal que el autómata cociente* $AAP(D_+)/\pi$ *es determinista y acepta* L *(AAP significa árbol aceptor de prefijos). En otras palabras, una muestra positiva es estructuralmente completa con respecto a un autómata si el autómata utiliza todas sus transiciones y todos sus estados finales para aceptar dicha muestra.*

Definición 19 *Para el algoritmo RPNI, una* muestra característica *[OG92]* (D_+, D_-) *con respecto a un lenguaje regular* L *es aquella que cumple las siguientes dos condiciones:*

- $\forall u \in N(L)$ *si* $u \in L$ *entonces* $u \in D_+$*, si no* $\exists v \in \Sigma^* | uv \in D_+$
- $\forall u \in Pc(L), \forall v \in N(L) | \delta(q_0, u) \neq \delta(q_0, v)$ *y ocurre una de dos cosas, ó* $\exists uu' \in D_+ | \exists vu' \in D_-$ *ó* $\exists vv' \in D_+ | \exists uv' \in D_-$

La primera condición garantiza que la muestra positiva sea estructuralmente completa y la segunda asegura que si se intenta mezclar dos estados distintos, existirá al menos una muestras negativa que lo impida.

1.2. Modelo de Aprendizaje

La inferencia gramatical se puede considerar una técnica inductiva para realizar aprendizaje automático, por eso es fundamental dejar claro qué se entiende por aprendizaje. Existen varios modelos de aprendizaje que vienen siendo utilizado en este ámbito: el modelo de Gold, el aprendizaje activo, el modelo PAC y el modelo basado en la teoría de Bayes, a continuación se presentan brevemente los elementos que componen el modelo de Gold, en tanto que los otros modelos mencionados son comentados en la Sección 2.2.

Se hace mayor énfasis en el modelo de Gold, ya que esta investigación se interesa especialmente por esta forma de definir el aprendizaje automático. Un modelo de aprendizaje, según Gold [Gol67], consta de tres elementos:

- Una definición de aprendizaje
- Un método de presentación de información
- Una relación de nombramiento, es decir, que asigna nombres (posiblemente más de uno) a lenguajes. El aprendiz identifica un lenguaje indicando uno de sus nombres

Con el fin de proponer un modelo de aprendizaje, Gold [Gol67] procede a definir cada uno de estos tres elementos:

- *Definición de aprendizaje.*
  Recibe el nombre de identificación en el límite y funciona de la siguiente forma: se tiene un aprendiz, que se encarga de formular una hipótesis q_t de un nombre del lenguaje L basándose en la información que ha recibido hasta el momento. El aprendiz es una función G que va de conjuntos de cadenas de unidades de información en nombres de lenguajes. $q_t = G(i_1, \ldots, i_t)$. Se dice que el aprendiz ha identificado L en el límite si pasado un tiempo finito, las hipótesis que produce son todas la misma y son un nombre de L.

  Una clase de lenguajes es identificable en el límite con respecto a un modelo de aprendizaje, si existe un aprendiz efectivo; es decir, un algoritmo que genere hipótesis y que cumpla la siguiente propiedad: dado cualquier lenguaje de la clase y dada cualquier secuencia de entrenamiento permitida para dicho lenguaje, el lenguaje va a ser identificado en el límite [Gol67].

- *Método de presentación de información.*
  Se refiere al esquema con el cual se presenta la información de entrada al aprendiz, puede ser en forma de texto: en donde la entrada se compone únicamente de cadenas que pertenecen al lenguaje objetivo; también puede ser en forma de informante: donde se proporcionan cadenas de símbolos indicando si ellas pertenecen al lenguaje objetivo ó a su complemento.

- *Relación de nombramiento.*
  Gold considera dos maneras de asignar nombres a lenguajes, las llama el probador y el generador, ambas son máquinas de Turing. El probador es un procedimiento de decisión para el lenguaje L, es decir, es una función que va de cadenas de símbolos al rango de enteros [0,1] devolviendo cero cuando la cadena de entrada no pertenece al lenguaje y uno cuando sí pertenece. El generador es una función de enteros en cadenas de símbolos, es decir, para cada entero retorna una palabra que pertenece al lenguaje L, en otras palabras el rango de esta función es exactamente L.

Gold definió varios modelos de aprendizaje de lenguajes, para ello propuso una definición de aprendizaje, dos métodos de presentación de información y seis métodos de asignación de nombres a lenguajes. El modelo es iterativo y funciona presentando unidades de información a un aprendiz que genera hipótesis a partir de ellas. Al escoger un método de presentación de información y un

método de asignación de nombres concretos, se instancia un modelo de aprendizaje particular. En el caso de esta investigación, se adopta la identificación en el límite como definición de aprendizaje, el informante como método de presentación de la información y los autómatas finitos como relación de nombramiento.

Todos los modelos de aprendizaje que se mencionan al comienzo de la sección están en desarrollo y cada uno de ellos tiene ventajas y desventajas: el modelo de Gold es teóricamente sólido y da origen a algoritmos convergentes, pero requiere datos abundantes y correctos, lo cual no siempre es viable en problemas reales. El modelo de aprendizaje activo logra resolver en tiempo polinomial algunos problemas que son intratables en el modelo de Gold, pero tiene el problema de la interactividad y de que en algunas tareas específicas no es posible construir el oráculo necesario para el buen funcionamiento del modelo, por ejemplo: si no se conoce el lenguaje objetivo como ocurre en muchas tareas reales. Los algoritmos probabilísticos se comportan mejor con información incompleta o ruidosa, aunque pierden su carácter exacto y hasta cierto punto son cajas negras cuyo funcionamiento interno es desconocido. Por su parte, el modelo PAC brinda un soporte teórico al desarrollo de modelos inductivos aproximados; los resultados conocidos son más teóricos que prácticos y apuntan a determinar qué clases de problemas son solucionables en tiempo polinomial y cuáles no lo son.

Conociendo las ventajas y desventajas de los diferentes modelos y siguiendo también la línea marcada por los trabajos que preceden y dan origen a esta investigación, se ha adoptado el modelo de aprendizaje de Gold, que permite obtener algoritmos con propiedades teóricas demostrables tales como la convergencia.

1.3. Autómatas Finitos Residuales

Un autómata de estado finito residual [DLT02] ó RFSA (por su sigla en inglés) es un autómata $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ tal que, para cada $q \in Q$, $L(\mathcal{A}, q)$ es un lenguaje residual (ver Definición 8) del lenguaje L reconocido por $\mathcal{A}$. Así $\forall q \in Q, \exists u \in \Sigma^*$ tal que $L(\mathcal{A}, q) = u^{-1}L$. En otras palabras, un *Autómata de Estado Finito Residual* (RFSA) $\mathcal{A}$ es un autómata no determinista tal que todos sus estados definen lenguajes residuales de $L(\mathcal{A})$. Todo DFA es un RFSA ya que cumple con la definición.

Dos relaciones definidas sobre el conjunto de estados de un autómata relacionan los RFSAs con la inferencia gramatical. Sea $D = (D_+, D_-)$ un conjunto de muestras, sea $u, v \in Pr(D_+)$. Se dice que $u \prec v$ si no existe una palabra w tal que $uw \in D_+$ y $vw \in D_-$. Se dice que $u \simeq v$ [1] si $u \prec v$ y $v \prec u$.

Se sabe que todo lenguaje regular tiene un conjunto finito de lenguajes residuales distintos [Myh57, Ner58], también se sabe que todo lenguaje regular puede representarse mediante DFAs y dado que todo DFA recortado (ver Definición 3) es un RFSA, se tiene que todo lenguaje regular puede ser descrito

[1] Esta relación es conocida en la terminología de Gold como *estados no obviamente distinguibles.*

mediante RFSAs [DLT02]. De los RFSA que representan un lenguaje regular, tiene particular interés el más pequeño de ellos, el que tiene el menor número de estados, este es el llamado RFSA canónico.

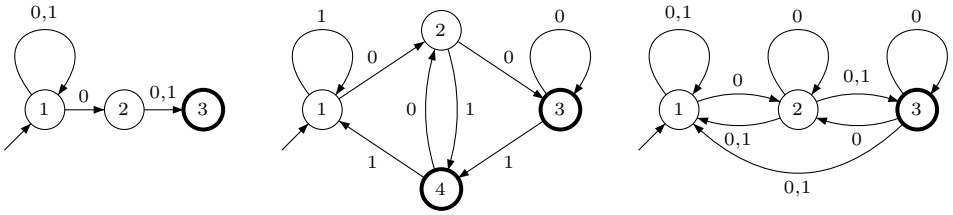
Formalmente, dado un lenguaje $L \subseteq \Sigma^*$ el RFSA *canónico* de L es el autómata $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ donde:

- $Q = \{u^{-1}L \mid u^{-1}L \text{ es primo}, u \in \Sigma^*\}$
- Σ es el alfabeto de L
- $\delta(u^{-1}L, a) = \{v^{-1}L \in Q \mid v^{-1}L \subseteq (ua)^{-1}L\}$
- $I = \{u^{-1}L \in Q \mid u^{-1}L \subseteq L\}$
- $F = \{u^{-1}L \in Q \mid \varepsilon \in u^{-1}L\}$

Observando la definición formal, se puede apreciar que el RFSA canónico es único, porque la forma de conectar los estados consiste en relacionarlos *siempre* que haya una relación de inclusión entre los correspondientes lenguajes residuales y no es posible que dos autómatas diferentes que cumplen con la definición en lo referente a los estados cumplan también con la definición de la función δ.

Si $L(\mathcal{A}, q) = u^{-1}L$ se dice que u es una palabra característica del estado q. El tamaño del RFSA canónico de un lenguaje regular tiene como cota superior al tamaño del DFA mínimo equivalente y como cota inferior el tamaño de los NFA minimales equivalentes [DLT02]. En estas condiciones, puede ocurrir que el RFSA canónico sea exponencialmente más pequeño que el DFA mínimo equivalente, o también puede ocurrir que ambos sean del mismo tamaño y que exista un NFA minimal exponencialmente más pequeño. También puede ocurrir que exista un NFA con el mismo número de estados que el RFSA canónico, pero con menos transiciones. Por otra parte, en algunos casos la palabra característica más corta asociada con un estado puede tener una longitud exponencial con respecto al número de estados del RFSA canónico; esto puede ocurrir en caso que la muestra característica se construya con base al DFA mínimo y que el RFSA canónico tenga un tamaño logarítmico con respecto al DFA mínimo. En general, estas afirmaciones muestran que encontrar un algoritmo que converja al RFSA canónico del lenguaje objetivo, no asegura que el resultado de la inferencia sea mínimo; aun así, existe la posibilidad de mejorar los resultados obtenidos con inferencia de DFAs para aquellos lenguajes regulares que se caractericen por ser representados con autómatas que tienen gran cantidad de lenguajes residuales compuestos. Qué tan comunes son en la práctica este tipo de lenguajes es una pregunta para la que aun no se conoce respuesta y de ello dependerá el interés práctico de las aplicaciones de la teoría de los RFSAs.

El siguiente ejemplo, tomado de [DLT02], muestra gráficamente varios autómatas que reconocen un lenguaje regular L; entre ellos un NFA, un DFA y el RFSA canónico. Sea $\Sigma = \{0, 1\}$ y sea $L = \Sigma^*0\Sigma$. Los tres autómatas de la Figura 1.2 reconocen L. Los estados finales (resp. no finales) se han dibujado con líneas gruesas (resp. delgadas).



$\mathcal{A}_1$ $\mathcal{A}_2$ $\mathcal{A}_3$

Figura 1.2: $\mathcal{A}_1$ es un autómata que reconoce $L = \Sigma^*0\Sigma$ pero que no es ni DFA ni RFSA. $\mathcal{A}_2$ es un DFA que reconoce L y también es un RFSA. $\mathcal{A}_3$ es el RFSA canónico para L.

- El primer autómata, $\mathcal{A}_1$ no es ni DFA ni RFSA. Los lenguajes por la derecha asociados a los estados son: $L(\mathcal{A}_1, 1) = \Sigma^*0\Sigma$, $L(\mathcal{A}_1, 2) = \Sigma$ y $L(\mathcal{A}_1, 3) = \varepsilon$. Se puede ver que $L(\mathcal{A}_1, 3) = \varepsilon$ y $\nexists u \in \Sigma^*$ tal que $L(\mathcal{A}_1, 3) = u^{-1}L$.

- El autómata $\mathcal{A}_2$ reconoce L y es el DFA mínimo de L, por lo tanto es un RFSA. En este caso $L(\mathcal{A}_2, 1) = \Sigma^*0\Sigma$, $L(\mathcal{A}_2, 2) = \Sigma^*0\Sigma + \Sigma$, $L(\mathcal{A}_2, 3) = \Sigma^*0\Sigma + \Sigma + \varepsilon$, $L(\mathcal{A}_2, 4) = \Sigma^*0\Sigma + \varepsilon$. Sin embargo no es canónico, ya que, por ejemplo, $L(\mathcal{A}_2, 3) = L(\mathcal{A}_2, 2) \cup L(\mathcal{A}_2, 4)$

- El autómata $\mathcal{A}_3$ es el RFSA canónico de L, el cual no es un DFA. Los lenguajes residuales asociados con los estados de $\mathcal{A}_3$ son: $L(\mathcal{A}_3, 1) = \varepsilon^{-1}L$, $L(\mathcal{A}_3, 2) = 0^{-1}L$ y $L(\mathcal{A}_3, 3) = 01^{-1}L$. Notar que a pesar de ser un ejemplo bastante pequeño, no es trivial representar los lenguajes residuales del modo que se hizo en los autómatas anteriores $\mathcal{A}_1$ y $\mathcal{A}_2$.

Denis [DLT02] define dos operadores sobre RFSAs que preservan la propiedad de equivalencia: saturación y reducción, el primero permite adicionar arcos y estados iniciales al autómata conservando el lenguaje que reconoce; el operador de reducción permite eliminar estados al autómata conservando también el lenguaje que reconoce.

Definición 20 *Dado $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ el* autómata saturado *de $\mathcal{A}$ es el autómata $\mathcal{A}^s = (Q, \Sigma, \delta^s, I^s, F)$, donde $I^s = \{q \in Q | L(\mathcal{A}, q) \subseteq L(\mathcal{A})\}$ y la función de transición se define como $\forall q \in Q, \forall a \in \Sigma, \delta^s(q, a) = \{q' \in Q | aL(\mathcal{A}, q') \subseteq L(\mathcal{A}, q)\}$. Se puede observar que $L(\mathcal{A}) = L(\mathcal{A}^s)$, ya que el lenguaje por la derecha de cada estado es el mismo tanto en el autómata original como en el saturado, de ello se desprende que si $\mathcal{A}$ es un RFSA, entonces $\mathcal{A}^s$ también lo es.*

El operador de reducción se define con base en la noción de estado eliminable [DLT02].

Definición 21 *Sea $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ un NFA y q un estado de Q. Se denota $R(q) = \{q' \in Q - \{q\} | L(\mathcal{A}, q') \subseteq L(\mathcal{A}, q)\}$. Si $L(\mathcal{A}, q) = \bigcup\{L(\mathcal{A}, q') - \{q'\} \in R(q)\}$ se dice que q es* eliminable *en $\mathcal{A}$.*

Definición 22 *Sea $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ un NFA y q un estado de Q. Si q es eliminable se define el* operador de reducción *$\phi(\mathcal{A}, q) = \mathcal{A}'$ tal que $\mathcal{A}' = (Q', \Sigma, \delta', I', F')$ donde: $Q' = Q - \{q\}$. $I' = I$ si $q \notin I$ ó $I' = (I - \{q\}) \cup R(q)$. $F' = F \cap Q'$. δ' se define para todo $q' \in Q', a \in \Sigma$ como: $\delta'(q', a) = \delta(q', a)$ si $q \notin \delta(q', a)$, en caso contrario $\delta'(q', a) = (\delta(q', a) - \{q\}) \cup R(q)$. Si q no es eliminable, $\phi(\mathcal{A}, q) = \mathcal{A}$. Se puede ver que $L(\mathcal{A}) = L(\mathcal{A}')$ siendo $\mathcal{A}' = \phi(\mathcal{A}, q)$. Los operadores de saturación y reducción son permutables: $\phi(\mathcal{A}, q)^s = \phi(\mathcal{A}^s, q)$.*

Estos operadores se pueden utilizar para construir el RFSA canónico ya que se sabe que si $\mathcal{A}$ es un RFSA saturado, el operador de reducción converge y el autómata resultante es el RFSA canónico del lenguaje $L(\mathcal{A})$ [DLT02]; sin embargo, este método es computacionalmente muy costoso, ya que determinar si un lenguaje residual es primo es una tarea de coste exponencial. Existe otro algoritmo para construir el RFSA canónico a partir de un NFA cualquiera, que teniendo complejidad exponencial, puede ser más simple de implementar y se describe a continuación. Este método de García [Gar06], guarda relación con el algoritmo de Polák para minimización de NFAs [Pol05].

El Algoritmo 1 recibe como entrada un NFA, en primer término lo convierte a su DFA mínimo equivalente que será la base de los pasos siguientes: calcular su reverso y el determinista de su reverso. Con la información de los bloques de estados que se forman en el determinista, se construye la matriz básica siguiendo el mismo procedimiento que usa Polák en [Pol05], pero sin agregar nuevos estados. Posteriormente se compara cada par de columnas, detectando si una está contenida en la otra, si todos los unos que hay en $c1$, también están en $c2$, se dice que la columna $c1$ está contenida en $c2$. El hecho que una columna esté contenida en otra, refleja una relación de inclusión entre los estados correspondientes a dichas columnas. De otra parte, cuando una columna puede reconstruirse como la unión de otras, el estado asociado a ella es compuesto. Con la información de estas comparaciones se construye una estructura arborescente en la que un estado es padre de aquellos otros estados cuyos lenguajes por la derecha contiene. Finalmente se recorre el árbol de inclusiones por niveles y se eliminan los estados compuestos; al hacerlo, se tiene cuidado de redireccionar todos los arcos que llegan a un estado compuesto hacia cada uno de los estados incluidos en él.

Ejemplo 23 *Con el fin de ilustrar el funcionamiento de este método para construir el RFSA Canónico correspondiente a un DFA mínimo, se va a utilizar un autómata tomado del artículo* Minimalizations of NFA Using the Universal Automaton *de Polák [Pol05], allí se propone un método de minimización que guarda cierta similitud con esta estrategia para encontrar el RFSA Canónico, ya que ambos métodos usan la matriz básica asociada a un autómata. La Figura 1.3 muestra el autómata de partida, que es un DFA mínimo. El Cuadro 1.1 muestra el mismo autómata representado en forma de tabla: cada fila del cuadro representa un estado del autómata, si el identificador del estado está antecedido por una flecha que apunta hacia la derecha se entiende que ese es un estado inicial, si está antecedido por una flecha que apunta hacia la izquierda, se trata de un estado final (de aceptación) y si tiene ambas el estado es simultáneamente inicial y*

Algoritmo 1 *ConstruirRFSACanonico*($\mathcal{A}$)

1: Convertir el NFA de entrada $\mathcal{A}$ en su DFA mínimo equivalente $\mathcal{A}'$
2: Calcular el reverso de $\mathcal{A}'$, llamado $R(\mathcal{A}')$
3: Calcular el determinista de $R(\mathcal{A}')$, llamado $D(R(\mathcal{A}'))$
4: M = Matriz básica de $D(R(\mathcal{A}'))$
5: $inclusiones = \{\}$
6: **for** $c1$ en Columnas(M) **do**
7: **for** $c2$ en Columnas(M) **do**
8: **if** $c2$ contiene a $c1$ **then**
9: $inclusiones = inclusiones \cup \{(Estado(c1), Estado(c2))\}$
10: **end if**
11: **end for**
12: **end for**
13: Organizar las relaciones de inclusión en forma arborescente
14: **for** s en el árbol de inclusiones, recorrido por niveles a partir de la raíz **do**
15: **if** $Columna[s] = \bigcup_{i \in Hijos(s)} Columna[i]$ **then**
16: $\mathcal{A}' = eliminarEstado(s, \mathcal{A}')$
17: **end if**
18: **end for**
19: Return $\mathcal{A}'$

final; cada columna corresponde a un símbolo del alfabeto. Líneas horizontales y verticales separan los elemento que identifican las filas y columnas del contenido de la tabla. Si la posición (i, j) de la tabla contiene el valor k, significa que desde el estado i mediante el símbolo j se puede alcanzar el estado k. Esta misma notación será utilizada en adelante para representar autómatas mediante tablas.

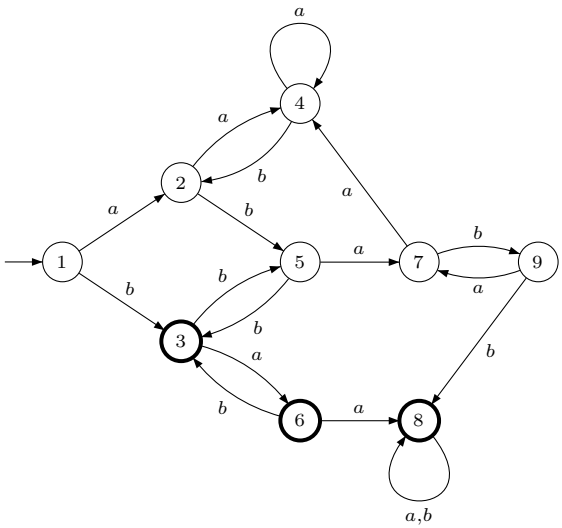


Figura 1.3: Ejemplo de Construcción del RFSA Canónico, Autómata Inicial

El primer paso consiste en calcular el reverso del autómata $\mathcal{A}$, ($R(\mathcal{A})$) que se aprecia en el Cuadro 1.2. A continuación se calcula el determinista del rever-

so $D(R(\mathcal{A}))$, el resultado se presenta en el Cuadro 1.3. Con esta información se construye la matriz básica, ella tiene en las filas los estados de $D(R(\mathcal{A}))$ y en las columnas los estados del DFA mínimo inicial, la posición $M[i,j]$ de la matriz básica puede valer 1 o 0, vale 1 en caso que el estado j del DFA mínimo, forme parte del estado i de $D(R(\mathcal{A}))$ y 0 en otro caso. El Cuadro 1.4 muestra la matriz básica correspondiente al ejemplo.

Cuadro 1.1: Autómata Inicial $\mathcal{A}$ en Representación de Tabla

		a	b
→	1	2	3
	2	4	5
←	3	6	5
	4	4	2
	5	7	3
←	6	8	3
	7	4	9
←	8	8	8
	9	7	8

Cuadro 1.2: $R(\mathcal{A})$ en Representación de Tabla

		a	b
←	1	-	-
	2	1	4
→	3	-	{1,5,6}
	4	{2,4,7}	-
	5	-	{2,3}
→	6	3	-
	7	{5,9}	-
→	8	{6,8}	{8,9}
	9	-	7

Cuadro 1.3: $D(R(\mathcal{A}))$ en Representación de Tabla

		a	b
→	{3,6,8}	{3,6,8}	{1,5,6,8,9}
⇄	{1,5,6,8,9}	{3,6,8}	{2,3,7,8,9}
→	{2,3,7,8,9}	{1,5,6,8,9}	{1,4,5,6,7,8,9}
⇄	{1,4,5,6,7,8,9}	{2,3,4,5,6,7,8,9}	{2,3,7,8,9}
→	{2,3,4,5,6,7,8,9}	{1,2,3,4,5,6,7,8,9}	{1,2,3,4,5,6,7,8,9}
⇄	{1,2,3,4,5,6,7,8,9}	{1,2,3,4,5,6,7,8,9}	{1,2,3,4,5,6,7,8,9}

Comparando las columnas de la matriz básica, se pueden detectar las relaciones de inclusión entre los lenguajes residuales asociados a los diferentes

Cuadro 1.4: Matriz Básica Obtenida a Partir de $D(R(\mathcal{A}))$

		1	2	3	4	5	6	7	8	9
$\rightarrow$	{3,6,8}	0	0	1	0	0	1	0	1	0
$\rightleftarrows$	{1,5,6,8,9}	1	0	0	0	1	1	0	1	1
$\rightarrow$	{2,3,7,8,9}	0	1	1	0	0	0	1	1	1
$\rightleftarrows$	{1,4,5,6,7,8,9}	1	0	0	1	1	1	1	1	1
$\rightarrow$	{2,3,4,5,6,7,8,9}	0	1	1	1	1	1	1	1	1
$\rightleftarrows$	{1,2,3,4,5,6,7,8,9}	1	1	1	1	1	1	1	1	1

estados del autómata inicial $\mathcal{A}$. Se considera que hay inclusión si los unos de una columna están contenidos en los unos de otra, por ejemplo, se puede observar en el Cuadro 1.4 que $2 \prec 3$. Con esta información también se pueden encontrar los estados compuestos, por ejemplo, la columna correspondiente al estado 9, se puede construir uniendo las columnas correspondientes a los estados 5 y 7, por lo tanto 9 es un estado compuesto. Realizando sistemáticamente todas las comparaciones, se obtienen todas las inclusiones y esto produce el árbol que se presenta en la Figura 1.4. También se deduce que los estados compuestos del autómata son: $8, 9, 5, 7$ *por lo tanto estos estados no forman parte del RFSA canónico. Expresando de otra manera las inclusiones obtenidas, se puede afirmar que:* $1, 2, 3, 4, 5, 6, 7, 9 \prec 8$*;* $1, 2, 4, 5, 7 \prec 9$*;* $2, 4 \prec 7$*;* $1, 4, 5 \prec 6$*;* $1, 4 \prec 5$ *y* $2 \prec 3$*, esta información se utilizará en el paso siguiente para eliminar correctamente los estados compuestos.*

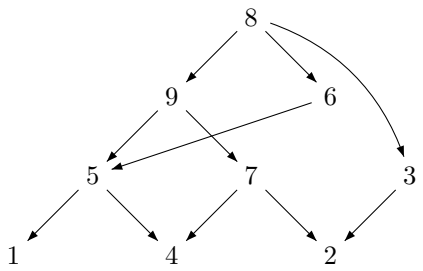


Figura 1.4: Ejemplo de Construcción del RFSA Canónico, Árbol de Inclusiones entre Estados

Una vez se elimina un estado compuesto x, los arcos que llegaban a él deben redireccionarse a cada estado incluido en x. En el ejemplo, el primer estado que se elimina es el 8*, por encontrarse en la raíz del árbol de inclusiones, por lo tanto, los arcos que llegan a* 8*, deben llegar a* $1, 2, 3, 4, 5, 6, 7, 9$*. Una vez realizada la eliminación y redireccionados los arcos, el autómata queda como se ilustra en la Figura 1.5.*

Posteriormente se eliminan siguiendo el mismo procedimiento los estados compuestos: 9*,* 7 *y* 5*, obteniéndose el autómata de la Figura 1.6 que es final y corresponde al RFSA canónico del DFA mínimo de partida.*

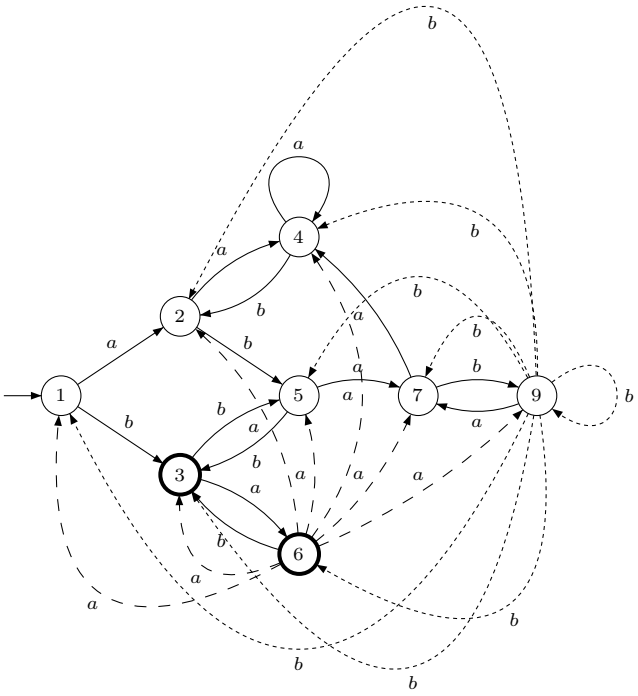


Figura 1.5: Ejemplo de Construcción del RFSA Canónico, Autómata Después de Eliminar el Estado Compuesto 8

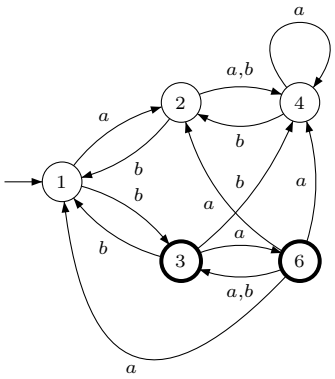


Figura 1.6: Ejemplo de Construcción del RFSA Canónico, Autómata Final

Vale la pena finalizar recopilando algunos resultados referentes a la complejidad de los problemas que se han mencionado en esta sección [DLT02]: decidir si un DFA es saturado pertenece a P; decidir si un NFA es saturado pertenece a $PSPACE - Completo$; construir el saturado de un NFA también pertenece a $PSPACE - Completo$ al igual que decidir si un NFA es RFSA y decidir si el saturado de un DFA es RFSA canónico; finalmente, construir el RFSA canónico equivalente a un DFA es $PSPACE - Completo$.

Estos resultados muestran que al usar RFSAs como modelo de representación para la inferencia no se escapa a la dificultad intrínseca del problema de

inferir NFAs. Inferir RFSAs puede ser beneficioso si se logra obtener hipótesis de tamaño cercano al RFSA canónico, ya que éste en algunos casos puede ser mucho más pequeño que el DFA mínimo correspondiente; aun en esta circunstancia habrá casos en que el RFSA canónico tenga el mismo tamaño del DFA mínimo y allí no se obtendrá mejora.

1.4. Autómata Universal

El autómata universal será de particular importancia en la Sección 3.5.2, por ese motivo, se dedica esta sección a presentar su definición y a describir un método para construirlo.

Definición 24 *Dado un lenguaje L, sea $\mathsf{U} = \{u_1^{-1}L \cap ... \cap u_k^{-1}L \mid k \geq 0, u_1, ..., u_k \in \Sigma^*\}$. El* autómata universal *[ADN70, Car70, Lom01, Pol05] para L se define como $\mathcal{U} = (\mathsf{U}, \Sigma, \delta, I, F)$ con:*

- $I = \{q \in \mathsf{U} \mid q \subseteq L\}$.
- $F = \{q \in \mathsf{U} \mid \varepsilon \in q\}$.
- *La función de transición es tal que $q \in \delta(p, a)$ si y sólo si $q \subseteq a^{-1}p$.*

En otras palabras, el autómata universal tiene como estados todos los lenguajes residuales de un lenguaje L y todas las intersecciones posibles de ellos. El número de derivadas (o lenguajes residuales) de un lenguaje regular es igual al número de estados de su autómata determinista mínimo n, por lo tanto el número de estados de su autómata universal es mayor o igual que n y menor o igual que 2^n.

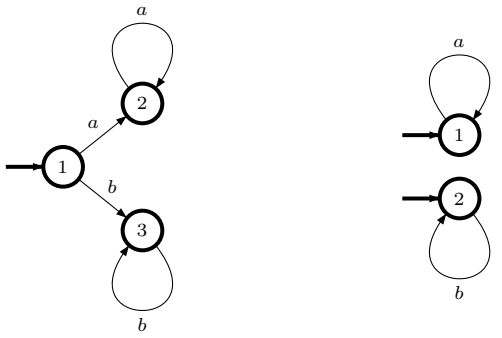
Según Carrez [Car70], el autómata universal $\mathcal{U} = (\mathsf{U}, \Sigma, \delta, I, F)$ para $L \subseteq \Sigma^*$ tiene las siguientes características:

1. $L(\mathcal{U}) = L$.
2. Para un autómata $\mathcal{A} = (Q, \Sigma, \delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ tal que $L(\mathcal{A}) \subseteq L$, la función $\varphi : Q \to \mathsf{U}$ definida como $\varphi(q) = \bigcap_{u \in L(\mathcal{A},q)} u^{-1}L$ es un homomorfismo entre autómatas.

Un método para construir el autómata universal de un lenguaje regular, dado el DFA mínimo que lo reconoce, consiste en partir del DFA mínimo A, calcular el reverso $R(A)$ y el determinista del reverso $D(R(A))$. Los estados de $D(R(A))$ y las intersecciones que entre ellos se puedan construir, forman el conjunto de estados del autómata universal. Para determinar las transiciones, se construye la matriz básica M a partir de A y de $D(R(A))$, en ella cada columna corresponde a un estado de A y cada fila a un estado de $D(R(A))$; $M[i, j] = 1$ si el estado j está contenido en el estado i y si no $M[i, j] = 0$. La matriz básica permite encontrar cuáles estados de A están incluidos en otros: si todos los valores 1 en la columna correspondiente al estado s de A están presentes en la columna correspondiente al estado t, se dice que s está incluido en t. Estas relaciones

permiten asignar las transiciones al autómata universal así: se comienza reflejando las transiciones que existen en el DFA mínimo y luego para cada estado del autómata universal se lanzan nuevas transiciones hacia los estados que están incluidos en él.

Ejemplo 25 *Sea el lenguaje $L = a^* + b^*$, la Figura 1.7 muestra en la parte (a) el DFA mínimo de dicho lenguaje y en la parte (b) un NFA minimal que lo reconoce. Con el propósito de construir el autómata universal de L, se expresa el DFA mínimo A en forma de tabla (ver el Cuadro 1.5), posteriormente se calcula $R(A)$ como se aprecia en el Cuadro 1.6 y el determinista del reverso $D(R(A))$ que se observa en el Cuadro 1.7. Los estados de $D(R(A))$ son también estados del autómata universal, como también lo son todos los estados que surjan de calcular las intersecciones de ellos. Para el ejemplo, los estados del autómata universal son: $\{\{1,2,3\}, \{1,2\}, \{1,3\}, \{1\}\}$ donde los tres primeros son estados del $D(R(A))$ y el último surge al calcular sus intersecciones.*



(a) (b)

Figura 1.7: Autómatas que Representan el Lenguaje para el cual se va a Calcular el Autómata Universal en la parte (a) está el DFA Mínimo y en la parte (b) un NFA Minimal.

Cuadro 1.5: Autómata Inicial A en Representación de Tabla

		a	b
$\leftrightarrows$	1	2	3
$\leftarrow$	2	2	-
$\leftarrow$	3	-	3

Conociendo ya los estados del autómata universal, deben calcularse sus transiciones. Para ello se comienza por construir la matriz básica, que se muestra en la Figura 1.8; comparando entre sí las columnas de la matriz, se detectan relaciones de inclusión entre estados, en el ejemplo, las relaciones que se detectan se reflejan en el gráfico de la Figura 1.8.

Cuadro 1.6: Reverso del Autómata Inicial $R(A)$ en Representación de Tabla

		a	b
$\leftrightarrows$	1	-	-
$\rightarrow$	2	1,2	-
$\rightarrow$	3	-	1,3

Cuadro 1.7: Determinista del Reverso del Autómata Inicial $D(R(A))$ en Representación de Tabla

		a	b
$\leftrightarrows$	{1,2,3}	{1,2}	{1,3}
$\rightarrow$	{1,2}	{1,2}	-
$\rightarrow$	{1,3}	-	{1,3}

Cuadro 1.8: Matriz Básica calculada a partir de $D(R(A))$

		1	2	3
$\leftrightarrows$	{1,2,3}	1	1	1
$\leftrightarrows$	{1,2}	1	1	0
$\leftrightarrows$	{1,3}	1	0	1

Una vez se han establecido las relaciones de inclusión presentes, se itera sobre cada estado del autómata universal ubicando las transiciones que trae el DFA mínimo y llevándolas también a todos los estados que están incluidos en él. La condición de ser estado inicial o final se conoce observando dicho estado en el $D(R(A))$.

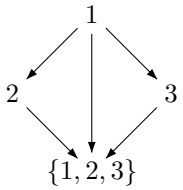


Figura 1.8: Ejemplo de Construcción del Autómata Universal, Árbol de Inclusiones entre Estados

En el ejemplo actual, se observa cómo el estado $\{1, 2\}$ corresponde con el estado 2 del DFA mínimo (esto se nota haciendo la intersección de las columnas 1 y 2 en la matriz básica y observando que el resultado es idéntico con la columna que corresponde al estado 2 del DFA mínimo) análogamente se detecta que $\{1, 3\}$ corresponde con el estado 3 del DFA mínimo y que $\{1\}$ corresponde con el estado 1 del DFA mínimo; para facilitar la escritura del autómata, al estado $\{1, 2, 3\}$ se le llamará 4 en el esquema final que se muestra en la Figura 1.9. Observar también que este estado 4 está incluido en todos los demás es-

tados y por esta razón le llegan las transiciones que llegan a todos los demás. Por último, observar que todos los estados son iniciales y finales, tal como lo indican las flechas a la izquierda de los estados en la Figura 1.8.

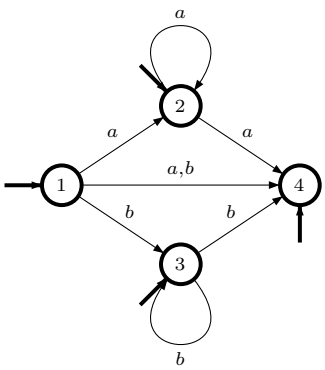


Figura 1.9: Autómata Universal del Lenguaje L

Capítulo 2

Antecedentes

El campo de la inferencia gramatical de lenguajes regulares ha evolucionado mucho en los últimos 40 años, tanto en cuanto a resultados teóricos como en el desarrollo de algoritmos de uso práctico. En este Capítulo se propone una relación comentada de trabajos que se consideran importantes dentro de este campo de estudio. En primer lugar, se hace un recuento de la evolución de la inferencia gramatical de lenguajes regulares, en la que se considera con mayor detalle la estrategia de variar el orden de mezcla de los estados que dio origen a los algoritmos conocidos como EDSM (Evidence Driven States Merging) y el planteamiento del problema de inferencia gramatical de lenguajes regulares como problema de búsqueda. También se presentan algunos antecedentes en cuanto a modelos de aprendizaje, córpora de prueba y estrategias para evaluación de algoritmos de inferencia gramatical.

2.1. Antecedentes en Inferencia Gramatical de Lenguajes Regulares

En las décadas de 1960 y 1970 E. Mark Gold realizó varios aportes importantes en cuanto a la inferencia gramatical de lenguajes regulares: en primer lugar, demostró que encontrar el autómata determinista (DFA) más pequeño, consistente con un conjunto dado de muestras positivas y negativas $D_+ \cup D_-$, es un problema NP-Completo [Gol78]. También propuso el modelo de identificación en el límite [Gol67], que es una valiosa herramienta para demostrar la corrección de algoritmos de inferencia gramatical. Además, Gold propuso un algoritmo de inferencia [Gol78] que dado un conjunto de ejemplos suficientemente representativo, puede construir en tiempo polinomial dicho autómata determinista mínimo. Este algoritmo es particularmente interesante, ya que varios algoritmos posteriores pueden entenderse como modificaciones suyas [GRC04a].

En la misma época, Trakhtenbrot y Barzdin propusieron su algoritmo de inferencia [TB73], que utiliza todas las cadenas de longitud menor o igual a un valor dado para realizar correctamente el proceso de inferencia; además mostraron que si esta información está disponible, el problema de inferencia es tratable. Se sabe que los algoritmos de Gold y Trakhtenbrot funcionan de forma simi-

lar, aunque utilizan representaciones muy diferentes de la información [GCR00].

En la década de 1980 aparecieron nuevos resultados negativos que confirmaron la dificultad teórica del problema de inferencia gramatical de lenguajes regulares. Angluin demostró que el problema de encontrar el autómata determinista más pequeño dado un conjunto de muestras es NP-Completo aún si el autómata objetivo tiene sólo dos estados o si faltan unas pocas muestras en el conjunto de entrenamiento, que debe ser completo hasta una longitud dada [Ang78].

Angluin también demostró que la tarea sigue siendo difícil aún si se supone que el algoritmo puede formular preguntas de pertenencia [Ang87] o de equivalencia [Ang90] a un oráculo; es decir, si dada una cadena puede preguntar si ella pertenece al lenguaje objetivo o no, o si la hipótesis actual es equivalente al autómata objetivo.

Los resultados teóricos negativos causaron un comprensible pesimismo en cuanto a las posibilidades de encontrar algoritmos prácticos de inferencia gramatical, lo cual llevó a la exploración de otros caminos alternativos que continuaron desarrollándose durante la década del 90 y hasta el presente. Por ejemplo, la inferencia mediante autómatas deterministas estocásticos [CO94]; el aprendizaje activo, es decir, aquel que aporta información adicional al proceso de inferencia en la forma de un oráculo o maestro informado que participa en el proceso cuando el algoritmo así lo requiere [PH97, PH00], [Ang87, Ang90, Ang92]; otras formas de involucrar información del entorno en el proceso de inferencia [CFKdlH04]; el uso de técnicas de inteligencia computacional como redes neuronales [OS01], algoritmos genéticos [Dup94, PN02] o programación evolutiva [LR05, BL05].

A comienzos de la década de 1990 apareció el algoritmo RPNI [OG92] y Lang propuso una modificación al algoritmo de Trakhtenbrot y Barzdin [TB73] llamada Traxbar al cual se hace referencia en [Lan92, CK03]. Estos algoritmos, gracias a su capacidad de generalización pueden inferir autómatas deterministas mínimos en tiempo polinomial; en el caso de RPNI la precisión de la aproximación depende de la calidad de la información en el entrenamiento y en caso que el conjunto de entrenamiento sea característico, se demuestra que el algoritmo converge al DFA mínimo del lenguaje objetivo [OG92]. Además de los esfuerzos por generar nuevos algoritmos, algunos investigadores han estudiado el espacio de búsqueda del problema de inferencia gramatical [DMV94, CF03b] y su relación con el espacio de búsqueda de otros problemas NP-Completos [PCS05]. Otra alternativa que se ha explorado es reducir el problema de la inferencia gramatical a otros problemas, por ejemplo: al problema de satisfacción de restricciones [PCS05, OS01] a un problema de cubrimiento de grafos [Cho03, PO99] o al problema algebraico de caracterizar un lenguaje como la preimagen de un hiperplano en un espacio de Hilbert, que es lo que recientemente se ha llamado lenguajes planares [CFW06, CFWS06].

En los años siguientes continuaron apareciendo mejoras a estos algoritmos, por ejemplo: Dupont propuso RPNI2, que es una versión incremental de RPNI [Dup96], Cano, Ruiz y García desarrollaron FCRPNI, que incluye el uso de configuraciones prohibidas como información adicional de entrada [CRG02],

Sebban y Janodet diseñaron una versión probabilística que funcione a partir de muestras con ruido [SJ03].

En el año 1998 Lang propuso el concurso Abbadingo One [LPP98] en el cual compitieron diferentes tipos de algoritmos de inferencia gramatical sobre un conjunto de problemas de inferencia de lenguajes regulares. Ganaron dos algoritmos, uno basado en mezcla de estados y el otro un algoritmo paralelo que hace búsqueda no determinista. El triunfo de un algoritmo basado en mezcla de estados, renovó el interés por la búsqueda de algoritmos exactos. De hecho, el propio Lang, basado en los resultados del concurso propuso su algoritmo Red-Blue descrito en [LPP98, CK03]. El algoritmo EDSM (Evidence Driven State Merging) de Rodney Price, que fue uno de los ganadores, también ha dado origen a posteriores estudios y mejoras [CK03, ACS04].

Una posibilidad que eventualmente ha sido considerada, es la de inferir autómatas no deterministas (NFA) en lugar de deterministas; en el año 1994 Yokomori publicó un trabajo en este tema cuyo método utilizaba consultas a un oráculo y contraejemplos [Yok94, dlH05], también lo hicieron Coste y Fredouille [CF00, CF03a, CFKdlH04] quienes partiendo del estudio de los autómatas finitos no ambiguos obtuvieron un método de inferencia de NFAs.

A partir del año 2000, Denis et. al. [DLT00, DLT02] han enfocado el tema de la inferencia de NFAs desde la parte teórica, al centrarse en un subconjunto de los NFAs, llamado RFSA (Residual Finite State Automata). Para esta subclase se demuestra la existencia de un RFSA canónico único para cada lenguaje regular. Denis propone un algoritmo que converge a él, llamado algoritmo DeLeTe2 [DLT00]; desafortunadamente, pruebas experimentales muestran que sus hipótesis con frecuencia son inconsistentes con las muestras de entrenamiento. Posteriormente, presenta el programa DeLeTe2 [DLT04] (nótese que se reportan los resultados de ejecutar dicho programa, pero no se explica el algoritmo que utiliza), cuyos resultados preliminares son prometedores para un subconjunto de lenguajes regulares en los que los métodos ya conocidos, como RPNI o Red-Blue no se comportan particularmente bien; esto es, cuando los lenguajes objetivo son expresiones regulares o NFAs generados aleatoriamente.

En la actualidad, se conocen pocos grupos trabajando en el tema de la inferencia de NFAs: de un lado se buscan métodos de minimización de NFAs que conduzcan a una forma canónica [Pol05]; otros trabajos han experimentado con algoritmos que converjan a RFSAs [GRC04a, GRC04b]; con algoritmos que infieren UFAs (Unambiguous Finite Automata) [ACS04, CF03b, CF03a], con extensiones a la estrategia RPNI que realizan mezcla no determinista de estados [GRCA05] y también se está trabajando en la inferencia de NFAs por yuxtaposición de subautómatas [dPGR06].

Otra estrategia que se sigue aplicando hasta el presente, consiste en adaptar los métodos de inferencia a tipos particulares de autómatas con miras a su aplicación a una tarea específica, por ejemplo: la inferencia de autómatas con parámetros permite trabajar en tareas con un alfabeto de tamaño grande, como ocurre en la construcción de modelos de hardware y software [BJR06], este trabajo extiende el algoritmo de Angluin que hace inferencia activa. Hacer

inferencia activa de ERAs (Event-Recording Automata) [GJP06] sirve para inferir modelos que necesitan representar el tiempo, como los que se utilizan para modelar sistemas reactivos; la inferencia de este tipo de autómatas se requiere, por ejemplo, en la verificación automática y en la generación de casos de pruebas para sistemas de telecomunicaciones. La inferencia de autómatas de árboles k-testables [KBBdB06] se está utilizando para la extracción de información en documentos estructurados. Coste y Kerbellec [CK05, CK06] han propuesto un algoritmo de inferencia de NFAs para la identificación de proteínas, este algoritmo realiza un alineamiento múltiple parcial en las cadenas de entrenamiento que le permite establecer segmentos de cadena que son similares, cada grupo de segmentos similares se convierte en una unidad gracias al proceso de mezcla y la realización sistemática de todas las mezclas posibles da origen a un autómata no determinista que representa una sucesión compleja de consensos locales.

2.1.1. Variaciones del Orden en la Mezcla de Estados

Todos los algoritmos que se consideran en este trabajo tienen como característica común el hecho de basarse en la mezcla de estados. Este concepto se describirá con más detalle en el Capítulo 3, por el momento basta con decir que dos estados se mezclan cuando se considera que ambos tienen asociado el mismo lenguaje y por lo tanto uno de ellos es redundante. La mezcla disminuye el tamaño del autómata y generaliza el lenguaje que éste reconoce. Se considera que dos estados tienen el mismo lenguaje asociado si no existe ninguna muestra de entrenamiento que contradiga dicha suposición, esto implica que eventualmente se terminen mezclando estados que, ante una mayor cantidad de información disponible no lo habrían sido. Realizar mezclas equivocadas tiene un impacto negativo en el proceso de inferencia, que se traduce en un aumento del tamaño de la hipótesis construida y en una disminución en la tasa de reconocimiento del autómata resultante. Con el fin de evitar hasta donde sea posible la realización de mezclas equivocadas, de la Higuera [dlHOV96] propuso modificar el orden de mezcla lexicográfico para favorecer las parejas de estados que tuvieran menor posibilidad de ser equivocadas; con este fin propuso asociar a cada pareja un puntaje y proceder a mezclar aquella pareja que tenga máximo puntaje; se pueden elegir muchos criterios para asignar el puntaje, en general se espera que dicho valor sea alto si hay abundante evidencia entre las muestras disponibles de que los dos estados tienen el mismo lenguaje asociado y valor bajo en caso contrario. Aunque los resultados iniciales no fueron concluyentes, la idea fue retomada posteriormente por Price [LPP98] quien propuso el algoritmo EDSM (Evidence Driven State Merging) que mostró muy buen desempeño en el concurso Abbadingo, llegando a ser uno de los ganadores.

La estrategia de EDSM [CK03], se resume en el Algoritmo 2. como se puede apreciar en la línea 5, cada vez que se quiere agrupar un par de estados, se evalúan todas las parejas factibles (línea 6), a cada una de ellas se le asocia una puntuación que puede calcularse como se muestra en el Algoritmo 4, es decir, contando el número de parejas en el árbol de mezcla que tienen el valor de salida definido y el mismo para ambos estados. Posteriormente el algoritmo escoge la pareja que tiene el máximo puntaje y realiza esa mezcla; cuando no hay más

candidatos a mezclarse el algoritmo termina.

Algoritmo 2 EDSM($D_+ \cup D_-$)

```
1:  M := APM(D+, D−)
2:  ok := verdadero
3:  while ok do
4:    puntajes := {}
5:    for (x, y) ∈ Q × Q do
6:      if x ≃ y then
7:        puntajes := puntajes ∪ {((x, y), calcularPuntaje(M, x, y))}
8:      end if
9:    end for
10:   if puntajes = {} then
11:     ok := falso
12:   else
13:     (x, y) := obtenerParejaConMaximoPuntaje(puntajes)
14:     M := mezclar(x, y)
15:   end if
16: end while
17: Return M
```

La estrategia EDSM fue posteriormente refinada, porque su principal inconveniente es la necesidad de realizar una gran cantidad de procesamiento para asociar un puntaje a cada posible pareja de estados; una opción fue considerar únicamente las parejas formadas por estados que se encuentren a una distancia W del estado inicial, lo que se conoce como algoritmo W-EDSM [CK03]. Sin embargo, una estrategia mejor para calcular las parejas candidatas llamada *blue-fringe* [LPP98, CK03], dio origen al algoritmo que se conoce también como *redBlue*, el cual se muestra en el Algoritmo 3. Este algoritmo es considerado el estado del arte en cuanto a inferencia de DFAs por mezcla de estados.

El Algoritmo 3 parte del árbol de prefijos de Moore de la muestra. En primer lugar inicializa el conjunto *rojos* con el estado inicial del árbol, este conjunto contiene en cada momento los estados que forman parte de la hipótesis. A partir del conjunto *rojos* se calcula el conjunto *azules*, que contiene los estados hijos de estados *rojos* y que no pertenecen a *rojos*. El algoritmo trabaja hasta que el conjunto *azules* se vuelva vacío. En cada iteración se busca la pareja formada por un elemento de *rojos* t uno de *azules* que tenga el mayor puntaje. El puntaje se calcula, como se aprecia en el Algoritmo 4, contando el número de concordancias de valor de salida en el árbol de mezcla que se genera a partir de la pareja de estados a evaluar y multiplicando este valor por la profundidad del estado rojo. La profundidad de un estado es el número de transiciones que lo separan del estado inicial. Si un elemento azul no tiene posibilidad de agruparse con ningún elemento rojo, se adiciona a la lista de estados promovibles al conjunto *rojos*. Esto se debe a que un estado que no es comparable con ninguno de la hipótesis necesariamente debe pertenecer a ella si se quiere mantener la consistencia con las muestras. En este algoritmo se favorece la acción de promover un estado del conjunto *azules* al conjunto *rojos* (líneas 18-19) a la acción de agrupar la

pareja de estados con el mayor puntaje (líneas 20-24). Al finalizar se retorna la máquina de Moore resultante.

Algoritmo 3 redBlue($D_+ \cup D_-$)

```
1:  M := APM(D+, D-)
2:  rojos := {1}
3:  azules := {x ∈ Q | ∀y ∈ rojos, ∀a ∈ Σ, δ(y, a) = x ∧ x ∉ rojos)}
4:  while azules ≠ {} do
5:    promovibles := {}
6:    for y ∈ azules do
7:      mezclado := falso
8:      for x ∈ red do
9:        if x ≃ y then
10:         puntajes := puntajes ∪ {((x, y), calcularPuntaje(M, x, y) ∗
            profundidad(x))}
11:         mezclado := verdadero
12:       end if
13:     end for
14:     if ¬mezclado then
15:       promovibles := promovibles ∪ {y}
16:     end if
17:   end for
18:   if promovibles ≠ {} then
19:     rojos := rojos ∪ {primero(promovibles)}
20:   else
21:     (x, y) := obtenerParejaConMaximoPuntaje(puntajes)
22:     M := mezclar(M, x, y)
23:     azules := {x ∈ Q | ∀y ∈ rojos, ∀a ∈ Σ, δ(y, a) = x ∧ x ∉ rojos)}
24:   end if
25: end while
26: Return M
```

2.1.2. La Inferencia Gramatical como Problema de Búsqueda

El problema de la inferencia gramatical se puede plantear en forma de problema de búsqueda de la siguiente manera: dado un conjunto de muestras positivas D_+ y un conjunto de muestras negativas D_-, encontrar el autómata determinista más pequeño que sea consistente con dichas muestras. Esto significa que el espacio de búsqueda está formado por todos los autómatas deterministas posibles y que de alguna forma hay que recorrer este inmenso espacio hasta encontrar el autómata más pequeño que sea consistente con las muestras. Trabajos previos han estudiado este espacio de búsqueda [DMV94, CF03b] que puede pensarse como un árbol de autómatas que tiene en la raíz el árbol aceptor de prefijos de las muestras de entrada y cuyos hijos corresponden a cada una de las posibles transformaciones que puede sufrir este árbol. En este enfoque, se pueden construir algoritmos exactos utilizando backtracking que tienen como desventaja su

Algoritmo 4 calcularPuntaje(M, x, y)

```
1:  puntaje := 0
2:  push(pila, (x, y))
3:  while ¬empty(pila) do
4:     (x, y) := pop(pila)
5:     if Φ(x) = Φ(y) ∧ Φ(x) ≠'?' then
6:        puntaje := puntaje + 1
7:     end if
8:     for a ∈ Σ do
9:        if Están definidos δ(x, a) y δ(y, a) then
10:          push(pila, (δ(x, a), δ(y, a)))
11:       end if
12:    end for
13: end while
14: Return puntaje
```

complejidad exponencial, o construir algoritmos de aproximación más veloces, pero que no garantizan llegar a encontrar el autómata más pequeño.

Entre los algoritmos de búsqueda exacta, vale la pena mencionar los algoritmos MMM (Mealy Machine Minimizer) [BO05], BICA [BO05] y Exbar [Lan99, BO05] siendo Exbar el más eficiente de ellos [BO05]. Por su parte, entre los algoritmos que hacen búsqueda aproximada, vale la pena destacar a SAGE [JP98] y a las versiones de EDSM para búsqueda ed-beam y ed-ss [Lan99, BO05].

El algoritmo MMM [BO05, CK03] parte de un autómata vacío que va creciendo a medida que se modifica para obtener consistencia con las muestras positivas y negativas, para ello construye el árbol aceptor de prefijos aumentado de dichas muestras y lo recorre por amplitud; luego realiza una búsqueda por backtracking en la que al considerar cada estado del árbol, el algoritmo decide si es posible identificar dicho nodo con alguno de la hipótesis actual o si la hipótesis debe ser modificada. El algoritmo BICA [OS01, CK03, BO05] sigue la misma estrategia general, pero mejora el mecanismo de búsqueda utilizando una base de restricciones que sirve para, ante un conflicto en el punto actual, detectar cuál de las decisiones anteriores puede estar causando ese conflicto y retroceder directamente a ese punto.

El algoritmo *exbar* es el estado del arte en cuanto a métodos exactos [BO05] de inferencia de DFAs mínimos, por ese motivo será explicado con más de detalle. El algoritmo *exbar* utiliza la técnica de recorrido blue-fringe, que ha sido explicada en el algoritmo *redBlue*, para establecer las parejas de estados que eventualmente se pueden mezclar: considera un conjunto de estados rojos y un conjunto de estados azules, asigna un puntaje a cada elemento del conjunto azul y con ese puntaje determina cuál será el estado azul a procesar en cada iteración, el criterio de selección consiste en contar el número de elementos rojos con los cuales el elemento azul podría ser mezclado y actuar en orden inverso. El Algoritmo 5 muestra la subrutina principal, que recibe un conjunto de muestras pasitivas y negativas y devuelve un DFA. En ella se construye el árbol aceptor

de prefijos de las muestras y posteriormente se itera controlando el tamaño de las hipótesis que se van a generar, empezando en 1 e incrementándolo indefinidamente. Es claro que el algoritmo podría modificarse para que se detenga en cuanto encuentre el primer autómata consistente con las muestres, el cual por la forma de construcción será necesariamente de tamaño mínimo.

Algoritmo 5 exbar($D_+ \cup D_-$)

```
1: maxRojo = 1
2: raiz = APM((D+ ∪ D−))
3: while verdadero do
4:    exhSearch(lista(raiz),maxRojo)
5:    maxRojo = maxRojo + 1
6: end while
```

El Algoritmo 6 recibe como entrada una lista de los estados que componen la hipótesis actual y el tope máximo sobre la longitud de dicha lista; cuando este algoritmo encuentra un DFA que explica las muestras lo escribe y termina. El método utilizado es backtracking aplicado en la forma habitual. Recordar que la subrutina *pickBlueNode* prefiere el nodo azul para el cual existan menos alternativas posibles, es decir, que se prefiere un nodo azul que no sea mezclable con ningún nodo rojo, ya que para el la única posibilidad es ser promovido, en siguiente término se prefiere un nodo azul que sólo sea mezclable con uno de los nodos rojos porque para él sólo hay dos posibilidades: mezclarlo con el único estado rojo compatible o promoverlo a rojo y así sucesivamente incrementando el número de alternativas posibles.

Algoritmo 6 exhSearch(listaRojos,maxRojo)

```
1: if longitud(listaRojos) ≤ maxRojo then
2:    listaAzules = calcularListaAzules()
3:    if length(listaAzules) == 0 then
4:       encontrarSolucion()
5:    else
6:       B = escogerNodoAzul()
7:       for R in listaRojos do
8:          if trataDeMezclar(R,B) != ERROR then
9:             exhSearch(listaRojos,maxRojo)
10:         end if
11:         deshacerMezcla()
12:      end for
13:      exhSearch(extenderLista(listaRojos,B),maxRojo)
14:   end if
15: end if
```

La subrutina *trataDeMezclar(R,B)*, que se puede ver en el Algoritmo 7, se encarga de realizar la mezcla de estados, para ello redirecciona los arcos que llegan y parten de B para que lleguen y partan de R. Adicionalmente, la subrutina *recorrer* (ver Algoritmo 8) propaga la mezcla para mantener el determinismo del

Algoritmo 7 Subrutina *trataDeMezclar(R,B)*

```
1: reasignar los padres del nodo azul como padres del nodo rojo
2: puntajeDeMezcla = 0
3: Return recorrer(R,B,puntajeDeMezcla)
```

Algoritmo 8 Subrutina *recorrer(R,B,puntajeDeMezcla)*

```
1: if B.signo ≠ '?' then
2:    if R.signo ≠ '?' then
3:       if B.signo == R.signo then
4:          puntajeDeMezcla = puntajeDeMezcla + 1
5:       else
6:          Return ERROR
7:       end if
8:    else
9:       R.signo = B.signo
10:   end if
11: end if
12: i = 0
13: while i ≤ longitudAlfabeto do
14:    hijoR = R.hijo[i]
15:    hijoB = B.hijo[i]
16:    if hijoB ≠ NULL then
17:       if hijoR ≠ NULL then
18:          recorrer(hijoR,hijoB,puntajeDeMezcla)
19:       else
20:          R.hijo[i] = B.hijo[i]
21:       end if
22:    end if
23:    i = i + 1
24: end while
25: Return puntajeDeMezcla
```

autómata actual y también ajusta la función de salida del estado R cuando ello sea necesario. La mezcla puede fallar en caso que exista inconsistencia entre los valores de salida de los estados R y B en ese caso la subrutina retorna ERROR. Notar que la mezcla se propaga inmediatamente para garantizar el determinismo del autómata resultante, esta es una de las diferencias más importantes entre Exbar y el algoritmo MMM. La subrutina *trataDeMezclar(R,B)* desempeña el mismo papel que *mezcladet*, que se presenta en el Algoritmo 13 de la Sección 3.2, allí se puede observar un ejemplo de su funcionamiento.

Pasando a los algoritmos de búsqueda aproximada, el algorimo SAGE (Self-Adaptative Greedy Estimate) [JP98, CK03, BO05] es una estrategia de búsqueda en árboles que es no determinista y paralela, ella muestrea el espacio de búsqueda para obtener información sobre regularidades en la distribución de las soluciones, luego la búsqueda se enfoca manteniendo en el siguiente nivel los estados más prometedores de acuerdo a una función objetivo. El algoritmo parte de la raíz del árbol de búsqueda seleccionando en cada nivel un número limitado de alternativas prometedoras, las cuales se procesan paralelamente. El procesamiento consta de dos etapas:

- Construcción: en esta etapa se establece una lista de alternativas de mezcla ubicadas en un mismo nivel del árbol; se asigna un procesador a cada alternativa, el cual se encarga de calcular un puntaje asociado a la misma. El puntaje se calcula en función de un camino aleatorio desde ese punto de la búsqueda hasta llegar a una hoja del árbol.
- Competición: cada procesador elige entre los hijos de su nodo actual, aquel con máximo puntaje y en el momento indicado todos los procesadores comienzan a trabajar en el siguiente nivel del árbol. El algoritmo termina cuando no hay más nodos para procesar.

El algoritmo ed-beam (Evidence Driven Beam search) [Lan99, BO05] combina la idea de beam search del algoritmo SAGE con el heurístico de EDSM, que se usa para dar puntaje a cada una de las parejas candidatas a mezclarse y elegir la más prometedora. Por su parte, el algoritmo ed-ss (Evidence Driven Stochastic Search) [BO05] procura detectar la mezcla con mayor probabilidad de ser equivocada, para eso se utiliza una medida que estima el impacto de cada mezcla realizada sobre todas las demás, teniendo en cuenta: los puntajes de las mezclas que ya no serían posibles, el incremento en los puntajes de las mezclas que se vean favorecidas y el decremento en los puntajes de las mezclas que se perjudican. La idea es que la mezcla que tenga mayor puntaje debe deshacerse primero y esto cambia el procedimiento habitual del backtracking de deshacer la última decisión que se haya tomado.

2.2. Modelos de Aprendizaje

La definición de un modelo de aprendizaje está en la base de cualquier investigación sobre aprendizaje en máquinas y en particular sobre inferencia gramatical. Existen varios modelos de aprendizaje que se han desarrollado a través de los años. En esta sección se presentan brevemente algunos de ellos. En la Sección 1.1 se ha presentado ya el modelo de Gold, que sirve de base para esta investigación.

- El Aprendizaje Activo.
  Se caracteriza por ser interactivo, esto quiere decir que además de la información que recibe de las muestras de entrenamiento, el aprendiz puede interrogar a un interlocutor planteándole una pregunta concreta; normalmente el interlocutor es un programa al que suele llamarse oráculo. Angluin [Ang87, Ang90] propone el modelo *MAT* (Minimally Adecuate Teacher) en el cual el interlocutor sólo es capaz de responder a dos tipos de preguntas concretas: preguntas de pertenencia o de equivalencia, es decir, se puede preguntar si una cadena en particular pertenece al lenguaje objetivo o no, o si la hipótesis actual es equivalente al autómata objetivo.

- Aprendizaje Basado en Métodos Estadísticos.
  Los clasificadores de Bayes son una forma muy consolidada de realizar aprendizaje inductivo. Este modelo se caracteriza por el uso intensivo de probabilidades. Los autómatas estocásticos y los algoritmos que hacen inferencia sobre ellos son un buen ejemplo de aplicación de esta teoría como modelo de aprendizaje [CO94]. La aplicación de Modelos Ocultos de Markov también forman parte de este modelo y se sabe que tienen un buen desempeño en una variedad de situaciones como el reconocimiento de habla entre otras.

- El Modelo PAC (Probably Aproximately Correct).
  En este modelo, que fue propuesto por Valiant [Val84], se parte del supuesto que el aprendiz no va a lograr aprender exactamente el lenguaje objetivo, si no que va a construir una aproximación cuya corrección respeta un nivel predefinido de error con una probabilidad también predefinida. De la Higuera lo define en [dlH05] como un modelo para estimar el grado de dificultad de un problema. En el caso de la inferencia gramatical, estima la dificultad del aprendizaje de una clase de lenguaje; la mayoría de los resultados conocidos son negativos [dlH05]. Por su parte Angluin [Ang92] lo define como un nuevo criterio de corrección para el aprendizaje de conceptos mediante ejemplos, en el que se enfatiza la importancia del aprendizaje en tiempo polinomial. Angluin [Ang92] dice que un algoritmo de aprendizaje A identifica con el modelo PAC un concepto de $\mathcal{C}$ en términos de un espacio de hipótesis $\mathcal{H}$ con respecto a una clase de distribuciones de probabilidad $\mathcal{D}$ si y sólo si para toda distribución D en $\mathcal{D}$ y todo concepto $c \in \mathcal{C}$, para todo número positivo ε, δ, cuando A se ejecuta con entradas ε, δ y accede al oráculo para D y c, él eventualmente termina y produce como salida una hipótesis $h \in \mathcal{H}$ tal que con probabilidad al menos $(1-\delta)$, la distancia entre c y h es menor que ε.

En resumen, se conoce que el modelo de Gold permite establecer en forma sólida la convergencia de un algoritmo de inferencia, aunque requiere información abundante y veraz. El aprendizaje activo funciona en forma similar, salvo que permite la interacción de aprendiz con una fuente externa de información lo cual es positivo porque aumenta la eficacia del modelo en algunos casos, pero requiere la existencia de un oráculo capaz de responder verazmente a las preguntas que formule el aprendiz. Los métodos estadísticos tienen mayor flexibilidad para adaptarse al ruido en la información de entrada, pero los resultados son

siempre aproximaciones.

2.3. Antecedentes en cuanto a Evaluación de Algoritmos de Inferencia Gramatical

El uso de datos sintéticos ha sido la forma más extendida en la comunidad académica para evaluar nuevos algoritmos de inferencia gramatical. Esto se debe por una parte a la necesidad de proveer un ambiente controlado para la efectiva evaluación de los mismos y por otra a que el tamaño de los problemas del mundo real en ocasiones supera las posibilidades de los algoritmos en diseño. Según Lang [LPP98], los autómatas aleatorios tienen una complejidad de Kolmogorov alta y se pueden generar fácilmente para el tamaño que se desee, cualidades estas que los hacen útiles como autómatas objetivo para la construcción de córpora de entrenamiento y prueba.

A medida que los algoritmos de inferencia gramatical han ido mejorando su desempeño, han ido surgiendo también córpora más complejos para realizar las pruebas. Sin embargo, debe señalarse que hacer públicas estas bases de datos no ha sido una costumbre general y por lo tanto es mucho más frecuente que cada grupo construya y utilice sus propios datos de entrenamiento y prueba. Haciendo un seguimiento a los córpora que han estado disponibles de manera pública y que por lo tanto han permitido hacer comparaciones objetivas entre algoritmos de inferencia gramatical de lenguajes regulares, se conocen los siguientes:

- Los lenguajes de Tomita [Tom82]. Estos son lenguajes sobre el alfabeto $\Sigma = \{a, b\}$ que pueden representarse con cuatro estados o menos. Son siete lenguajes:
  - L1: a^*
  - L2: $(ab)^*$
  - L3: Cualquier secuencia que no tenga un número impar de a's consecutivas después de un número impar de b's consecutivas
  - L4: Cualquier secuencia que no tenga más de dos a's consecutivas.
  - L5: Cualquier secuencia con un número par de a's y un número par de b's.
  - L6: Cualquier secuencia cuyo número de a's difiera de su número de b's en 0 módulo 3.
  - L7: $a^*b^*a^*b^*$

  Estos lenguajes fueron extendidos posteriormente por Miclet y Gentile y también por Dupont [Dup94], aumentando 8 lenguajes objetivo más y extendiendo en algunos casos el alfabeto a tres símbolos $\Sigma = \{a, b, c\}$:
  - L8: a^*b
  - L9: $(a^* + c^*)b$
  - L10: $(aa)^*(bbb)^*$

- L11: Cualquier secuencia con un número par de a's y un número impar de b's.
- L12: $a(aa)^*b$
- L13: Cualquier secuencia con un número par de a's.
- L14: $(aa)^*ba^*$
- L15: $bc^*b + ac^*a$

Muestras de entrenamiento para estos lenguajes, utilizadas por Dupont en [Dup94], pueden ser descargadas de la página de Internet GIB, cuya dirección se provee más adelante en esta sección. Estos datos están orientados a la inferencia de DFAs y han sido utilizados también en [dlHOV96, PN02].

- Los datos generados para el concurso Abbadingo One [LPP98, JP98] siguen estando disponibles en el presente[1], así como el oráculo de prueba. En la misma página se pueden descargar implementaciones del algoritmo traxbar y de tres versiones de EDSM, siendo *redBlue* una de ellas. Estos córpora comprenden cuatro problemas sencillos de afinamiento y 16 problemas para la competición propiamente dicha. Los problemas de competición corresponden a cuatro tamaños de lenguaje objetivo: 64, 128, 256 y 512 estados y para cada grupo a cuatro grados de densidad de los datos de entrenamiento. El método para generar los lenguajes objetivo ha sido reutilizado en varios trabajos posteriores [CK02, CK03, BL05] y el formato en que se almacenan las muestras se ha convertido casi en un estándar dentro del área. Estos problemas están orientados a la inferencia de DFAs y tienen una dificultad creciente de modo que algunos de los problemas aun no han sido resueltos.

- Generador de autómatas Gowachin. Utiliza el mismo método de generación de autómatas de Abbadingo One, pero se pueden fijar valores concretos a los parámetros de generación, también se puede introducir ruido en los datos. Los experimentos reportados por [SJ03, ACS04, CFKdlH04, BO05, LR05] se han realizado utilizando este generador de DFAs que sigue funcionando en la actualidad[2].

- Los autómatas usados por Oliveira y Silva [OS01] también están disponibles en Internet[3]. Son 115 máquinas de Mealy generadas aleatoriamente, con entradas y salidas en el alfabeto $\Sigma = \{0, 1\}$, su tamaño varía entre 3 y 19 estados. Hay 575 conjuntos de entrenamiento, cada uno contiene 20 cadenas de longitud 30. Estos experimentos también fueron usados por Lang [Lan99] en los experimentos del algoritmo *exbar*.

- Los datos de prueba del algoritmo *DeLeTe2* [DLT02, DLT04]. Estos datos de entrenamiento y prueba han sido diseñados para probar algoritmos de inferencia de NFAs y están disponibles en Internet[4]. Los lenguajes objetivo se han construido a partir de expresiones regulares y NFAs. Son 120

[1] `abbadingo.cs.unm.edu`

[2] `abbadingo.cs.unm.edu/gowachin`

[3] `enbedded/eecs.berkeley.edu/Alumni/aml`

[4] `http://www.grappa.univ-lille3.fr/~lemay/` seleccionar *Recherche* y luego seleccionar *Annexes*

autómatas objetivo, agrupados en cuatro grupos de 30 lenguajes cada uno. Hay cuatro conjuntos de entrenamiento de 50, 100, 150 y 200 muestras respectivamente y 1000 muestras de prueba para cada conjunto. Estos datos también se han usado en [CF03a] y en la investigación que aquí se reporta.

- Los datos de prueba del algoritmo de inferencia de UFAs [CF03a] se etiquetan a partir de autómatas no ambiguos, pero también a partir de DFAs generados aleatoriamente.

2.3.1. Técnicas de Generación de Autómatas Aleatorios

Dado el interés que para esta investigación tiene el generar datos de entrenamiento y prueba, se presentan con más detalle dos maneras de realizar esta tarea que han sido exitosamente aplicadas en varias experimentaciones anteriormente.

- Generación de autómatas para Abbadingo One [LPP98]. El procedimiento utilizado para generar los autómatas objetivo de n estados consta de los siguientes pasos: primero se construye un digrafo de grado 2 con $\frac{5}{4}n$ nodos, luego se escoge aleatoriamente un estado inicial y se extrae el subgrafo alcanzable desde él, finalmente se etiquetan aleatoriamente los estados del grafo como estados de aceptación o rechazo. Este procedimiento produce grafos con una distribución de probabilidad centrada en n y una distribución de profundidades de los estados centrada cerca de $2log_2n - 2$. La variación en el tamaño no tiene mayor incidencia, pero la variación en la profundidad complica la construcción del conjunto de muestras de entrenamiento, así que una vez se genera el grafo, este es desechado si su profundidad no es exactamente $2log_2n - 2$.

- Generación de DFAs aleatorios [CP05]. Este método se caracteriza por producir DFAs aleatorios uniformemente distribuidos entre todos los DFA con el mismo número de estados. El método se apoya en la existencia de una biyección entre el conjunto de los árboles m-arios con $m \geq 2$, el conjunto de los subconjuntos prefijos de Σ^* con $|\Sigma| \geq 2$ y el conjunto $\mathscr{R}_{(m,n)}$ de tuplas generalizadas. Para que se comprenda mejor la estrategia se presentan primero algunas definiciones necesarias.

Definición 26 *El* grado de entrada (resp. de salida) *de un vértice es el número de arcos que son incidentes hacia (resp. desde) dicho vértice. Se denota $d^-(v)$ (resp $d^+(v)$) al grado de entrada (resp. salida) del vértice v.*

Definición 27 *Un* árbol m-ario *es un grafo acíclico dirigido $\mathscr{T} = \langle V, E\rangle$ donde $V = \{v_1, v_2, \ldots, v_t\}$ es el conjunto de vértices del árbol y $E \subseteq V \times V$ es el conjunto de arcos del grafo. El grado de entrada de cada vértice de un árbol m-ario es exactamente 1, excepto para un estado llamado raíz y denotado v_1 que tiene grado de entrada cero. El grado de salida de los vértices de un árbol m-ario varía entre cero y m.*

Definición 28 *Un* árbol m-ario extendido de orden n *es un árbol m-ario cuyos vértices están particionados de manera que $V = N \uplus L$ con $|N| = n$,*

donde $v \in N \Rightarrow d^+(v) = m$ *y* $v \in L \Rightarrow d^+(v) = 0$. *El conjunto* N *son los nodos internos y el conjunto* L *son las hojas del árbol.*

Según muestra Champarnaud[CP05], existe una biyección entre los árboles m-arios de n vértices y los árboles m-arios extendidos de orden n. Basta con asociar a cada vértice v del árbol m-ario $m - d^+(v)$ hojas para obtener un árbol m-ario extendido.

Definición 29 *Un conjunto de palabras* X *de* Σ^* *es* prefijo *si contiene todas las palabras* $u \in \Sigma^*$ *tal que existe* $w \in \Sigma$ *de modo que* $uw \in X$.

Definición 30 *El conjunto* $\mathscr{R}_{(m,n)}$ *de tuplas generalizadas de elementos del rango* $[1, n] \cap \mathbb{N}$ *se define como:*

$$\mathscr{R}_{(m,n)} = \left\{ (k_1, \ldots, k_s) \in [1, n]^s \mid \forall i \in [2, s], k_i \geq \lceil \frac{i}{m-1} \rceil, k_i \geq k_{i-1} \right\}$$

Para generar los autómatas se parte de tuplas generalizadas, que mediante una biyección dan origen a árboles m-arios extendidos y los árboles m-arios extendidos, permiten generar estructuras de transiciones y posteriormente autómatas deterministas. La uniformidad en la distribución se garantiza ya que se muestra una forma de obtener tuplas generalizadas aleatorias uniformemente distribuidas. Dado que a cada tupla corresponde un sólo árbol m-ario extendido, el árbol también está uniformemente distribuido en su dominio y de cada árbol se elige aleatoriamente uno de los autómatas que puede producir, con lo cual el autómata también resulta estar uniformemente distribuido en su propio dominio. Como detalle matemático vale la pena comentar que el número de tuplas generalizadas y árboles extendidos crece siguiendo una extensión de los Números de Catalán. Las experimentaciones en [CF03b, DLT04] han utilizado este método para la generación de DFAs. En este proyecto se utilizó este método de generación para desarrollar uno de los córpora de prueba.

2.3.2. Criterios de Evaluación para Algoritmos de Inferencia Gramatical

Los criterios de evaluación pueden variar de acuerdo a los objetivos de la evaluación; sin embargo, en general el criterio de evaluación más común es la tasa de acierto sobre muestras de prueba, algunas veces se le llama exactitud (en inglés accuracy). Otro criterio muy importante es el tamaño de las hipótesis producidas por el algoritmo.

En los experimentos estilo Abbadingo One [LPP98], se evalúa el criterio de exactitud, ya que el oráculo sólo acepta la entrada cuando ella corresponde a una exactitud mayor o igual al 99 %; sin embargo en este caso se involucra otra variable que es la densidad de la muestra de entrenamiento, a menor densidad es más difícil lograr una exactitud elevada.

En los experimentos del algoritmo DeLeTe2 [DLT04] se mide la exactitud. Adicionalmente se usa un criterio para comparar dos algoritmos, por ejemplo, A y B. Para cada bloque del corpus, se cuenta cuantas veces la exactitud de A fue mayor que la de B, cuantas veces B fue mejor que A y cuantas veces empataron (se maneja un pequeña holgura para considerar que ha habido empate). Se reportan los tres valores obtenidos.

Capítulo 3

La Mezcla de Estados

La mezcla de estados es una característica común a una variedad de algoritmos de inferencia gramatical como: *RPNI*, *traxbar*, *redBlue*, *exbar*, *DeLeTe2* y todos los algoritmos que se proponen en esta investigación. La mezcla permite disminuir el tamaño de la hipótesis de inferencia y generalizar el lenguaje que acepta; si se hace correctamente, de manera que mantenga la consistencia con las muestras de entrenamiento, es un mecanismo que puede llevar a la convergencia en el límite. En este capítulo se presentan las relaciones de inclusión entre estados en cuanto a su relación con la mezcla de estados y sus posibles aplicaciones en el diseño de algoritmos de inferencia gramatical. También se explica el mecanismo básico de mezcla de estados y algunas variaciones que se han evaluado durante del desarrollo de este trabajo; estas variaciones serán mencionadas más adelante cuando se describan los algoritmos que las han utilizado. Igualmente se presentan las reflexiones que se han realizado sobre la convergencia en el límite de la mezcla de estados y el principal resultado teórico obtenido, que consiste en demostrar que el orden de las mezclas de estados no afecta la convergencia en el límite, tanto si se trabaja con modelos deterministas como no deterministas.

3.1. Relaciones de Inclusión

La relación de inclusión entre estados se refiere exactamente a la inclusión que puede existir entre el lenguaje por la derecha de un estado y el lenguaje por la derecha de otro. Esta relación tiene que ver con la mezcla de estados, ya que dados dos estados p y q, $(p \prec q \wedge q \prec p) \Leftrightarrow p \simeq q$, dicho en palabras: p está incluido en q y q está incluido en p si y sólo si p y q son no obviamente distinguibles y por lo tanto pueden mezclarse. Notar que lo anterior significa que la relación de inclusión es menos estricta que la relación de no-distinguibilidad, es decir, siempre que hay no-distinguibilidad hay inclusión (en ambos sentidos), pero no siempre que hay inclusión hay no-distinguibilidad, por ejemplo: si $p \prec q$ pero $q \not\prec p$. Esto lleva a considerar que las relaciones de inclusión entre estados pueden aportar información útil en el proceso de inferencia gramatical. En esta sección se destacan dos maneras concretas de aprovechar esa información que son la duplicación de transiciones y la definición de valores de salida, pero antes de entrar en materia, se define el árbol de mezcla, que es una construcción

importante para expresar el procedimiento de la mezcla de estados.

3.1.1. El Árbol de Mezcla

Sea $M = (Q, \Sigma, \{0, 1, ?\}, \delta, \Phi, q_0)$ una máquina de Moore determinista. El árbol de mezcla asociado a $p, q \in Q$ es una lista de parejas (x, y) formada por los estados alcanzables desde p y q, cuando se avanza simultáneamente a partir de ellos con el mismo sufijo en orden lexicográfico. El Algoritmo 9 explica con detalle el procedimiento de construcción. El árbol de mezcla se usa en este trabajo en la detección de relaciones de inclusión y en la implementación de la mezcla determinista.

Algoritmo 9 *arbolMezcla*(M, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
Require: $p, q \in Q$
1: $lista1 := \{\}$
2: $lista2 := \{(p, q)\}$
3: **while** $lista2 \neq \{\}$ **do**
4: $(x, y) := primero(lista2)$
5: $lista2 := lista2 - \{(x, y)\}$
6: $lista1 := lista1 \cup \{(x, y)\}$
7: **for** $a \in \Sigma$ **do**
8: **if** $\delta(x, a) \wedge \delta(y, a)$ están definidos **then**
9: $lista2 := lista2 \cup \{(\delta(x, a), \delta(y, a))\}$
10: **end if**
11: **end for**
12: **end while**
13: Return $lista1$

La Figura 3.1 muestra una máquina de Moore a partir de la cual, considerando la pareja de estados $(1, 2)$ se obtiene el árbol de mezcla que presenta la Figura 3.2. En el árbol cada nodo corresponde a una pareja de estados de la máquina, se avanza en el árbol moviéndose en la máquina hacia el sucesor de ambos estados con un mismo símbolo. Si alguno de los estados no tiene sucesor, no se avanza en esa dirección. El símbolo que permite el avance se muestra asociado a cada arco del árbol. Recordar que, en el esquema del árbol $+x$ significa $\Phi(x) = 1$, $-x$ significa $\Phi(x) = 0$. y si no hay ningún símbolo significa $\Phi(x) =?$. Aunque gráficamente siempre se va a presentar el árbol de mezcla en forma arborescente, la información se representa en una lista, para el ejemplo tal lista es: $\{(+1, +2), (+1, 4), (+2, +3), (+2, 6), (+1, 5), (4, 8), (+1, 7), (6, 10), (+1, 9), (8, +12), (+2, 11), (4, -13)\}$

3.1.2. Duplicación de Transiciones

Cuando se habla de inclusión entre estados de un autómata, se hace referencia a la inclusión entre los lenguajes por la derecha asociados a dichos estados (como se ha mencionado en la Sección 1.3). La relación de inclusión juega un papel muy importante en la teoría de los RFSAs y también es de utilidad en el

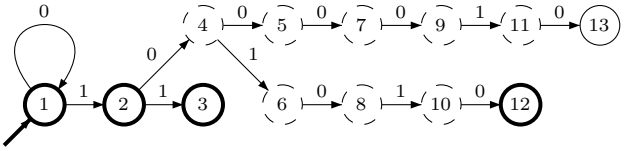


Figura 3.1: Máquina de Moore Inicial Usada para Ilustrar qué es un Árbol de Mezcla

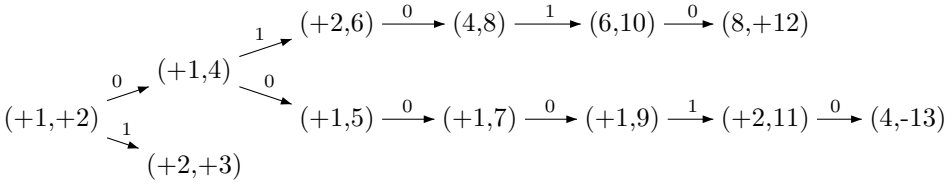


Figura 3.2: Árbol de mezcla para los estados 1 y 2

diseño de algoritmos que infieren RFSAs, como se verá en las Secciones 4.2 y 4.3 donde esta relación se aplica en algoritmos concretos. El hecho que dos estados p y q estén relacionados mediante inclusión ($p \prec q$) en un autómata A, significa que $L(A,p) \subseteq L(A,q)$ lo que se ilustra en la Figura 3.3, donde se representa el lenguaje por la derecha de p como un área triangular con un relleno a rayas a partir de dicho estado, el lenguaje por la derecha de q es un área más grande de fondo blanco, en la cual se distingue como subconjunto el mismo lenguaje por la derecha de p. Conocer que dos estados guardan entre sí una relación de inclusión permite ajustar el autómata para que en caso de eliminar el estado incluyente, el estado incluido reconozca correctamente la porción de lenguaje por la derecha que comparten.

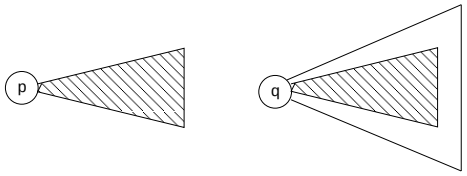


Figura 3.3: Ilustración de los lenguajes residuales asociados a los estados p y q, dado que $p \prec q$. Se representa el lenguaje por la derecha de p como un área triangular con relleno a rayas a partir de dicho estado, el lenguaje por la derecha de q es un área más grande de fondo blanco, en la cual se distingue como subconjunto el mismo lenguaje por la derecha de p

Cuando la información de las muestras es suficiente, se puede establecer la relación de inclusión entre dos estados p y q revisando su árbol de mezcla. Si existe en el árbol una o más parejas cuyos valores de salida sean de la forma $(-x,+y)$, se puede concluir que $p \prec q$, ya que existe al menos una cadena que pertenece al lenguaje residual de y que no pertenece al lenguaje residual de x.

Si por el contrario, se encuentra una o más parejas con valores de salida de la forma $(+x, -y)$, se concluye la relación contraria $p \succ q$. En caso que se encuentren ambos tipos de parejas en el mismo árbol los estados no guardan entre sí relación de inclusión (otros autores se refieren a esta condición como estados no-comparables o también como estados obviamente distinguibles) y si no hay ninguna de las dos combinaciones de valores, la relación entre los estados no puede establecerse con la información disponible.

Ejemplo 31 *Por ejemplo, considerando el árbol aceptor de prefijos de la Figura 3.4 se puede buscar establecer la relación de inclusión entre los estados* 3 *y* 9*, para ello se construye el árbol de mezcla, que se aprecia en la Figura 3.5(a), allí se observa la pareja de estados* $(+6, -12)$*, por lo tanto la relación es* $3 \succ 9$*. En cambio, al buscar la relación entre los estados* 1 *y* 2 *se concluye que los estados no son comparables, dado que en el árbol de mezcla (ver Figura 3.5(b)) existen parejas con las dos combinaciones de signo, por ejemplo:* $(+1, -2)$ *y* $(-2, +4)$*. Por último, si se busca la relación entre los estados* 2 *y* 3*, se construye el árbol de mezcla de la Figura 3.5(c) del cual no es posible deducir ninguna relación de inclusión. Notar que si no hay evidencia que apoye una relación de inclusión, pero hay acuerdo de signos, por ejemplo* $(+x, +y)$ *ó* $(-x, -y)$ *esta información sugiere más bien una posible relación de equivalencia entre los estados raíz del árbol de mezcla.*

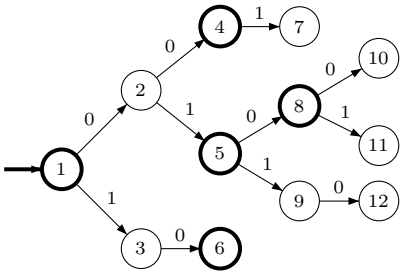


Figura 3.4: Máquina de Moore Inicial Usada para Detectar Relaciones de Inclusión

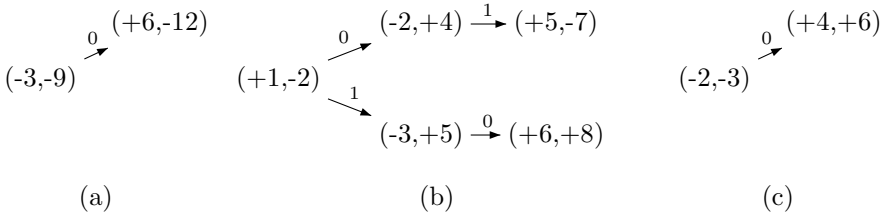


Figura 3.5: Árboles de Mezcla para Detectar Relaciones de Inclusión

Una manera de actuar sobre el autómata, cuando se conoce la relación de inclusión entre dos estados, consiste en redireccionar los arcos que llegan al estado incluyente, para que lleguen también al estado incluido. Esto garantiza que

en caso que el estado incluyente desaparezca, el estado incluido aceptará adecuadamente la porción de lenguaje residual que comparten; si igual cosa ocurre en todos los estados incluidos por un estado compuesto, él podrá ser eliminado sin afectar el lenguaje aceptado por el autómata. La Figura 3.6 muestra gráficamente en qué consiste la redirección de los arcos: en la parte izquierda de la Figura (ANTES), los prefijos del estado q se representan con una región triangular que llega al estado y que tiene un relleno en zigzag y los prefijos del estado p como una región triangular que llega al estado y tiene un relleno en círculos; una vez se realiza la redirección de los arcos, es decir, en la parte derecha de la Figura (DESPUES), en el esquema del estado p se aprecian sus propios prefijos y también los prefijos de q que ahora también llegan a p. En ese mismo diagrama del estado p se aprecia claramente que en esta situación el estado p es capaz de reconocer las mismas cadenas que el estado q en el ámbito del lenguaje residual que comparten. El Algoritmo 10 realiza este procedimiento. Debe destacarse que la introducción de estos arcos, que en principio son redundantes, puede convertir el autómata en no determinista.

Algoritmo 10 *incluido*(M, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
Require: $p, q \in Q$, $p \prec q$
1: **for** $r : r \in Q, a \in \Sigma, \delta(r, a) = q$ **do**
2: $\quad \delta = \delta \cup \{\delta(r, a) = p\}$
3: **end for**
4: Return M

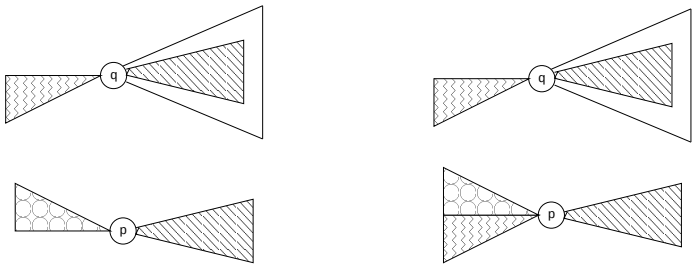


Figura 3.6: Ilustración de los Lenguajes Residuales Asociados a los Estados $p \prec q$ Despues de Redireccionar las Transiciones. Los prefijos del estado q se representan con una región triangular que llega al estado y que tiene un relleno en zigzag, en el esquema del estado p se aprecian sus propios prefijos, como una región con relleno en círculos y también los prefijos de q que ahora también llegan a p. En el diagrama se aprecia claramente que en esta situación el estado p es capaz de reconocer las mismas cadenas que el estado q en el ámbito del lenguaje residual que comparten.

3.1.3. Definición de Valores de Salida

Considerar las relaciones de inclusión entre estados tiene otro aspecto que puede aprovecharse en el diseño de algoritmos de inferencia gramatical: una vez se establece una relación de inclusión entre dos estados y se concluye que ellos no son equivalentes (es decir, que no existe inclusión en ambos sentidos) esta información puede servir para dar valor de salida a estados que hasta el momento tienen valor indefinido. Para ello se toma en cuenta una pareja de estados cuya relación de inclusión se conoce (o se supone) y su correspondiente árbol de mezcla. El razonamiento es el siguiente: si $p \prec q$, y las muestras de entrenamiento son consistentes, ninguna pareja de estados descendientes de (p, q) en el árbol de mezcla podrán tener relaciones del estilo $(+r, -s)$, ya que ello llevaría a una inconsistencia. Eso significa que si $p \prec q$ y en el árbol de mezcla se tiene la pareja (r, s) tal que r es descendiente de p, s es descendiente de q, $\Phi(r) = 1$ y $\Phi(s) =?$ se puede afirmar que $\Phi(s) = 1$ porque en caso que su valor fuera 0 causaría una inconsistencia; análogamente, si $\Phi(r) =?$ y $\Phi(s) = 0$, se puede afirmar que $\Phi(r) = 0$ porque otra opción introduce inconsistencia. Si la relación conocida es $p \succ q$ y en el árbol de mezcla existe una pareja (r, s) con $\Phi(r) = 0$ y $\Phi(s) =?$ se puede afirmar que $\Phi(s) = 0$ y si $\Phi(r) =?$ y $\Phi(s) = 1$, entonces $\Phi(r) = 1$, por las mismas razones ya expuestas. Los nuevos valores de salida que se van adicionando al autómata a medida que se detectan relaciones de inclusión y se dan las condiciones aquí descritas, aumentan la información disponible como si se ampliaran las muestras de entrenamiento a medida que avanza la inferencia [GRCA05]. El Algoritmo 11 presenta más exactamente la manera de proceder dados dos estados p y q tales que $p \prec q$.

Algoritmo 11 *definirValoresSalida*(M, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, \Phi, q_0)$
Require: $p, q \in Q$, $p \prec q$
1: $M' := M$ // $M' = (Q, \Sigma, \{0, 1, ?\}, \delta, \Phi', q_0)$
2: **while** p y q no son no-comparables **do**
3: $arbol := arbolMezcla(M', p, q)$
4: **for** $(u, v) \in arbol$ **do**
5: **if** $\Phi(u) = 1 \wedge \Phi(v) =?$ **then**
6: $\Phi(v) := 1$
7: **end if**
8: **if** $\Phi(u) =? \wedge \Phi(v) = 0$ **then**
9: $\Phi(u) := 0$
10: **end if**
11: **end for**
12: **end while**
13: **if** p y q son no-comparables **then**
14: Return M
15: **else**
16: Return M'
17: **end if**

Ejemplo 32 *La Figura 3.7 ilustra un autómata que puede encontrarse en me-*

dio de un proceso de inferencia, notar que existen bastantes estados cuyo valor de salida se ignora. La Figura 3.8 muestra el árbol de mezcla correspondiente a la pareja de estados $(+1, +2)$ *del autómata de la Figura 3.7 con los respectivos valores de salida de los estados. Suponiendo que la relación entre* 1 *y* 2 *es* $2 \prec 1$*, la pareja* $(8, +12)$ *permite asignar* $\Phi(8) = 1$ *y a través de la pareja* $(4, +8)$ *se asigna* $\Phi(4) = 1$*. El nuevo estado del autómata se observa en la Figura 3.9.*

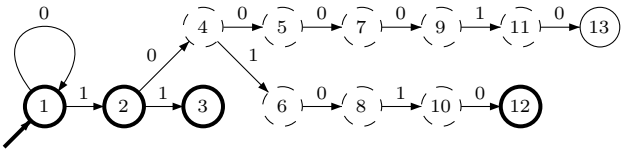


Figura 3.7: Autómata Ejemplo donde $1 \succ 2$

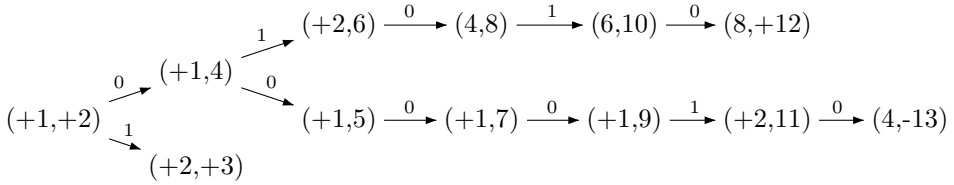


Figura 3.8: Árbol de mezcla a partir de los estados 1 y 2

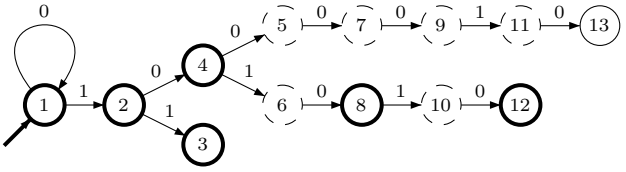


Figura 3.9: Autómata Después de Haber Definido Algunos Valores de Salida

En resumen, si se diseña un algoritmo enfocado a la detección y eliminación de estados compuestos, las relaciones de inclusión permiten transformar la máquina manteniendo el lenguaje que reconoce; si el algoritmo se enfoca a la mezcla de estados, el aumentar la información sobre los valores de salida de los estados permite tomar decisiones más informadas.

3.2. El Mecanismo Básico de Mezcla de Estados

La mezcla de estados, en el sentido que aquí se aplica, es una operación que se realiza sobre un modelo que puede ser determinista o no y que se centra en dos estados específicos que se fusionan en uno sólo. El Algoritmo 12 realiza la

Algoritmo 12 *mezcla*(M, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
Require: $p, q \in Q$, $p \ll q$ en orden lexicográfico
1: **if** $\Phi(p) \neq ? \wedge \Phi(q) \neq ? \wedge \Phi(p) \neq \Phi(q)$ **then**
2: Return M
3: **end if**
4: **if** $\Phi(p) = ?$ **then**
5: $\Phi(p) := \Phi(q)$
6: **end if**
7: **for** $s \in \Sigma$ **do**
8: **for** $r : \delta(r, s) = q$ **do**
9: $\delta := \delta \cup \{\delta(r, s) = p\}$
10: $\delta := \delta \setminus \{\delta(r, s) = q\}$
11: **end for**
12: **end for**
13: **for** $s \in \Sigma$ **do**
14: **for** $x : \delta(q, s) = x$ **do**
15: $\delta := \delta \cup \{\delta(p, s) = x\}$
16: $\delta := \delta \setminus \{\delta(q, s) = x\}$
17: **end for**
18: **end for**
19: $Q := Q \setminus \{q\}$
20: Return M

mezcla de dos estados p y q en una Máquina de Moore determinista M. Es condición necesaria para poder realizar la mezcla que los dos estados tengan valores de salida compatibles, esto es que tengan el mismo valor o que al menos uno de los dos tenga valor indefinido (ver línea 1 del Algoritmo 12). Si el estado que permanece tenía valor de salida indefinido antes de la mezcla, toma el valor de salida del estado eliminado (líneas 3 y 4). Como efecto de la mezcla el autómata se modifica para que el estado p reciba todos los arcos que llegaban al estado q (líneas 7-12) y emita todos los arcos que salían de él (líneas 13-18). Finalmente, el estado q desaparece. La Figura 3.10 muestra un autómata intermedio en un proceso de inferencia, observar los estados 1 y 4 con sus respectivos predecesores y sucesores, la Figura 3.11 muestra las modificaciones que causa la mezcla de dichos estados.

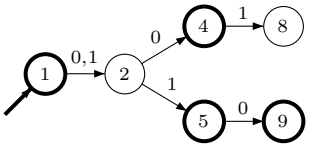


Figura 3.10: Autómata Antes de la Mezcla de los Estados 1 y 4

La modificación que sufren las transiciones al realizar la mezcla de dos estados puede causar no determinismo, lo cual es una característica indeseable en algoritmos que infieren máquinas deterministas, esto suele resolverse propa-

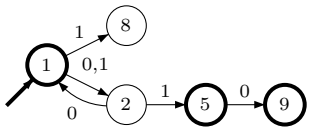


Figura 3.11: Autómata después de haber mezclado los estados 1 y 4 sin propagación

gando la mezcla. El Algoritmo 13 realiza la mezcla con propagación, para ello utiliza una cola en la que va adicionando las parejas de estados que, por efecto de la mezcla ya realizada, y para evitar el no determinismo, deben ser mezcladas también. La propagación debe extenderse hasta que desaparezca el no determinismo, pero fallará en el caso que trate de mezclar dos estados con valores de salida inconsistentes, en ese caso la subrutina retorna la máquina que entró sin efectuarle ningún cambio (líneas 6 y 7).

Algoritmo 13 $mezcladet(M, p, q)$

Require: $M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
Require: $p, q \in Q$, $p \ll q$ en orden lexicográfico
1: $M' := M$
2: $lista := \{(p, q)\}$
3: **while** $lista \neq \emptyset$ **do**
4: $(r, s) := first(lista)$
5: $M_1 := mezcla(M', r, s)$
6: **if** $M_1 = M'$ **then**
7: Return M
8: **else**
9: $M' := M_1$
10: **for** $a \in \Sigma$ **do**
11: **if** $\delta(p, a)$ y $\delta(q, a)$ están definidos **then**
12: $lista := append(lista, (\delta(p, a), \delta(q, a)))$
13: **end if**
14: **end for**
15: **end if**
16: **end while**
17: Return M'

En la Figura 3.11, el no determinismo se aprecia en el estado 1 del cual parten dos arcos con el símbolo 1, en este caso la propagación consiste en mezclar los estados 2 y 8, cuyo resultado se aprecia en la Figura 3.12.

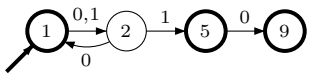


Figura 3.12: Autómata después de haber mezclado los estados 1 y 4 con propagación

La mezcla de estados causa generalización en el lenguaje aceptado, porque el redireccionamiento de arcos genera bucles y nuevos caminos que permiten acep-

tar otras cadenas además de las muestras de entrada. Por ejemplo, observar que la cadena 0011 es rechazada antes de la mezcla (ver Figura 3.10), en cambio después de la mezcla es aceptada (ver Figuras 3.11 y 3.12). Desde el punto de vista de la inferencia gramatical, no se puede afirmar que esta generalización acerque el modelo actual a una representación del lenguaje objetivo. Ello depende de estrategias externas a la mezcla misma como pueden ser evaluar la consistencia del nuevo modelo con las muestras de entrenamiento.

3.3. La Mezcla No-Determinista de Estados

Al estudiar la mezcla de estados en la inferencia de máquinas no deterministas, surge la necesidad de adaptar el modo de proceder, ya que en este caso lo que hasta ahora se ha llamado árbol de mezcla se convierte en una estructura más compleja debido a la existencia de varios sucesores de un estado con un mismo símbolo. Para que la mezcla sea completa debe considerar todas las opciones del no determinismo. El contexto en el que se propone realizar esta mezcla es el conocido como blue-fringe, en el cual se parte del árbol de prefijos de Moore de las muestras de entrada y se definen dos conjuntos de estados: el conjunto rojo formado por los estados que ya han sido procesados y forman parte de la hipótesis actual y el conjunto azul formado por los estados hijos de los estados rojos que a su vez no son rojos, es decir los estados que están en la frontera entre la porción del modelo que ya se ha procesado y el resto. Se mezcla siempre un estado rojo con un estado azul, esto limita el número de parejas candidatas para la mezcla, ya que a falta de este criterio se debería intentar mezclar todas las parejas de estados posibles, tal como ocurre en el algoritmo *EDSM*.

Mezclar un estado del conjunto rojo con un estado del conjunto azul garantiza que aunque el estado rojo puede verse como la raíz de un subgrafo que puede ser no determinista, el nodo azul es siempre la raíz de un subárbol que es una porción determinista de la máquina actual, por ese motivo la mezcla de una pareja rojo, azul es siempre finita, ya que no hay posibilidad de encontrar ciclos en el árbol de mezcla; de hecho, se verá más adelante que el método tiene coste polinomial. El subárbol con raíz en el estado azul establece la forma del árbol de mezcla, en tanto que los sufijos que lo forman proporcionan una manera ordenanda de recorrerlo; a medida que se avanza por cada rama, se avanza simultáneamente por el subgrafo con raíz en el estado rojo, abarcando todos los caminos que permita el no determinismo. El Algoritmo 14 muestra esta manera de proceder.

Ejemplo 33 *La máquina de Moore de la Figura 3.13 presenta un hipotético estado intermedio en un proceso de inferencia que requiere mezcla no determinista, se van a mezclar los estados* 2 *y* 3*. El estado* 2 *pertenece al conjunto rojo, es decir a la porción de la máquina que puede ser no determinista, en tanto que el estado* 3 *pertenece a la porción arborescente y determinista. El primer paso consiste en extraer los sufijos del estado* 3*, en este caso hay un único sufijo que es la cadena* 010*. La mezcla que se va a realizar, se representa claramente mediante el trellis del Cuadro 3.1, en el cual cada columna está asociada a un*

Algoritmo 14 *mezclanodet*(M, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
Require: $p \in$ conjunto rojo y $q \in$ conjunto azul

```
1:  M' := M
2:  for cadena ∈ Suf(q) do
3:      rojos := {p}
4:      azul := q
5:      for simbolo ∈ cadena do
6:          for r ∈ rojos do
7:              M_1 := mezcla(M', r, q)
8:              if M_1 = M' then
9:                  Return M
10:             else
11:                 M' := M_1
12:             end if
13:         end for
14:         rojos' := {}
15:         for r ∈ rojos do
16:             if δ(r, simbolo) esta definido then
17:                 rojos' := rojos' ∪ δ(r, simbolo))
18:             end if
19:         end for
20:         rojos := rojos'
21:         q := δ(q, simbolo)
22:     end for
23: end for
24: Return M'
```

símbolo del sufijo y hay dos filas principales asociadas con los estados que se van a mezclar. La fila superior, correspondiente al estado 2*, ubica bajo cada símbolo el conjunto de estados alcanzables desde los estados de la columna anterior con dicho símbolo. Por ejemplo, la primera columna contiene los estados* 2 *y* 4 *porque ellos son alcanzables desde* 2 *con el símbolo* 0 *y el estado* 6 *que es alcanzable desde* 3 *con dicho símbolo. Esta columna indica que los estados* $(2, 4, 6)$ *deben mezclarse para convertirse en uno sólo. La segunda columna, correspondiente al símbolo* 1*, indica la mezcla de los estados* $(5, 7)$*. Una vez realizada la mezcla, la máquina de Moore tiene el aspecto que se muestra en la Figura 3.14.*

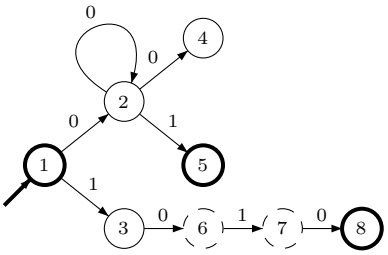


Figura 3.13: Autómata donde se va a realizar la mezcla no determinista de los estados 2 y 3

Cuadro 3.1: Trellis que ilustra la mezcla no determinista de los estados 2 y 3 siguiendo el prefijo 010

	0	1	0
2	2	5	
	4		
3	6	7	8

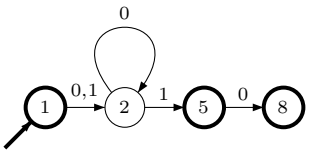


Figura 3.14: Autómata después de realizar la mezcla no determinista de los estados 2 y 3

Teniendo en cuenta que la mezcla no determinista necesita recorrer el subárbol que pende del estado azul, lo que en peor caso puede costar $O(n)$, donde n es el tamaño de la muestra de entrada y que internamente realiza una mezcla determinista de coste $O(n)$, el coste de la mezcla no determinista tiene como cota superior $O(n^2)$.

3.4. Importancia del Orden de Mezcla

Dado un árbol aceptor de prefijos correspondiente a una muestra dada, existen muchas maneras de proceder a la mezcla de estados, desde tiempo atrás

se conoce que existen ordenamientos que permiten obtener más rápidamente la convergencia que otros. Hay que recordar que en la práctica los algoritmos de mezcla funcionan sobre una muestra incompleta del lenguaje objetivo cuyas propiedades no se conocen, no se sabe si es una muestra característica o si es una muestra universal, por ejemplo; esto implica que cuando se agrupan dos estados y se verifica que el nuevo modelo sea consistente con las muestras, la consistencia se puede obtener ó bien porque efectivamente se ha detectado un estado redundante y se ha eliminado ó bien porque las muestras negativas disponibles fueron insuficientes para detectar la inconsistencia. En consecuencia, entre menos información de entrenamiento se tenga hay una mayor posibilidad de que el algoritmo mezcle estados que no debería. Realizar una mezcla equivocada se puede reflejar en un tamaño más grande de lo necesario en la hipótesis final y también en una menor tasa de reconocimiento de dicha hipótesis. Ambas cosas debidas a que dicha mezcla aleja la hipótesis del verdadero autómata objetivo. Una mezcla equivocada que se realiza al principio del proceso de inferencia es mucho más perjudicial que una que se realiza al final, porque una vez que se lleva a cabo una mezcla, todas las decisiones posteriores están determinadas por ella. Con estas ideas en mente, un criterio importante al elegir un ordenamiento de los estados a mezclar será aquel que ayude a minimizar las mezclas incorrectas, siempre limitados por la información que proporcionen las muestras disponibles.

Tal vez el primer criterio de ordenamiento que se utilizó fue el orden lexicográfico, que aplica el algoritmo RPNI [OG92]. Se sabe que la mezcla determinista sólo se hace efectiva si no encuentra ninguna inconsistencia al visitar todo el árbol de mezcla. Si el árbol de mezcla es grande hay mayor información a contrastar antes de dar por válida la mezcla, en cambio si el árbol es muy pequeño la mezcla se puede aceptar con muy poca información al respecto. Por eso, empezar en orden lexicográfico puede ser una buena idea, ya que los primeros estados en ser considerados son aquellos de los que penden los árboles de mezcla más grandes y a medida que la inferencia avanza se van considerando parejas con árboles de mezcla cada vez más pequeños. En principio, este método podría evitar mezclas equivocadas en los primeros niveles, donde son más perjudiciales.

Observando el comportamiento de la inferencia en orden lexicográfico, se notó que el hecho que el árbol de mezcla sea grande no implica necesariamente que contenga más información, ya que la información se encuentra en los valores de salida de los estados y puede ocurrir que dichos valores sean indefinidos, haciendo que aunque el árbol de mezcla conste de muchos nodos no sea posible detectar inconsistencias. Esto llevó a considerar un orden en las mezclas que dependa directamente de la información contenida en las muestras, dando prioridad a la mezcla mejor soportada por ellas. El esquema general lo propuso De la Higuera [dlHOV96] y consiste en establecer un conjunto de parejas candidatas a mezclarse, asociar a cada una de ellas un puntaje calculado con respecto al árbol de mezcla y a las muestras de entrenamiento y finalmente mezclar la pareja que obtenga el mayor puntaje. Se pueden proponer diversos criterios para asignar el puntaje, inicialmente se contaba el número de estados con valor de salida diferente de indefinido en el árbol de mezcla. Sin embargo, un criterio más apropiado fue el propuesto por Lang y Price [LPP98] de contar las coincidencias entre los valores de salida de las parejas que forman el árbol de mezcla; es decir, cada pareja de la forma $(+x, +y)$ ó $(-x, -y)$ incrementa en uno el puntaje. Este

valor base, se complementa con otro término que privilegia la pareja de estados que se encuentre a menor profundidad en el modelo (es decir, más cerca del estado inicial).

Aunque en la siguiente sección se muestra que cualquier orden que se elija para realizar las mezclas de estados converge en el límite, desde el punto de vista práctico elegir un orden adecuado para realizar las mezclas es una línea muy importante de trabajo, porque aporta velocidad en el proceso de convergencia.

3.5. Convergencia de la Mezcla de Estados

3.5.1. Convergencia de la Mezcla de Estados en Máquinas Deterministas

En este apartado se va a demostrar que la inferencia gramatical de lenguajes regulares mediante mezcla de estados es una estrategia que converge en el límite, independientemente del orden en el cual se realicen las mezclas. La experimentación muestra que el tiempo de convergencia se puede reducir significativamente al elegir un orden de mezcla concreto, el orden lexicográfico o la estrategia blue-fringe son alternativas que convergen rápidamente. Sin embargo, cualquier otro orden que se elija también termina por converger. El argumento de la demostración es el siguiente: dada una enumeración cualquiera de Σ^* y una muestra característica, por construcción de la muestra, un algoritmo de mezcla de estados que proceda en el orden de la enumeración de Σ^* converge en el límite al DFA mínimo del lenguaje objetivo.

El primer paso consiste en, dada una muestra positiva cualquiera, convertirla en una muestra característica para un algoritmo que realice mezclas de estados como el Algoritmo 16. Dada una enumeración de Σ^*, sea D_+ una muestra estructuralmente completa (ver Definición 18) y sea n el estado con más alto valor en la enumeración de $APM(D_+)$. Si se amplia D_+ de modo que aparezcan todos los prefijos de palabras del lenguaje con valor menor que n en la enumeración y para los prefijos que no son palabras del lenguaje se adiciona la palabra del lenguaje con menor longitud que lo contiene y para cada par de estados de $APM(D_+)$ que correspondan a estados diferentes en el autómata objetivo, se añaden las muestras positivas o negativas que sean necesarias para distinguirlos (D'_+, D'_-). El conjunto $(D_+ \cup D'_+, D'_-)$ constituye una muestra característica.

El Algoritmo 15 ilustra más detalladamente el procedimiento descrito en el párrafo anterior. La subrutina $mezcladet(M, i, j)$ es muy similar a la que se ha presentado previamente en el Algoritmo 13, salvo que en este caso se debe verificar antes de hacer la mezcla si ese par de estados ya ha sido procesado, esto con el fin de evitar ciclos infinitos. La subrutina $consistente(M, D)$ verifica si la máquina M es consistente con el conjunto de muestras D, si la muestra es positiva y la máquina la acepta, o si es negativa y la máquina la rechaza, retorna $verdadero$; en otro caso retorna $falso$.

El Algoritmo 15 procede en la siguiente forma: en las líneas 5-14 se extiende

el conjunto de muestras positivas garantizando que todos los prefijos del lenguaje objetivo L anteriores en la enumeración al valor máximo inicial n, están presentes. En las líneas 15-17 se extiende el árbol de prefijos de Moore con estas nuevas muestras. En la línea 18 se calcula la distinguibilidad del árbol de prefijos de Moore. Lang[Lan92] la define como: el entero d más pequeño tal que para todo par de estados no equivalentes en la máquina existe un sufijo no más largo que d que envía exactamente uno de los dos estados a un estado de aceptación. En la línea 19 se calcula la profundidad del árbol de prefijos, que es el máximo de las distancias mínimas entre el estado inicial y cada uno de los estados del modelo. Los valores de distinguibilidad y profundidad se utilizan en la línea 25 para acotar el tamaño máximo de las cadenas que deben considerarse para evitar la mezcla de estados diferentes (líneas 20-39), tal como lo hizo Lang en [Lan92]. Los ciclos de las líneas 20 y 21 recorren los estados en el orden de la enumeración E y generan las parejas del mismo modo que lo hará el algoritmo de mezcla, esto permite obviar algunos estados que cuando llegue el momento de considerarlos ya habrán desaparecido a causa de un agrupamiento anterior. En caso que los estados sean mezclables, la mezcla se realiza y se actualiza la máquina M (línea 35); en caso contrario, se adiciona a los conjuntos D'_+ y D'_- la o las muestras necesarias para evitar que los estados i y j puedan agruparse (líneas 25-33).

Se aprecia que la construcción de la muestra característica depende de la enumeración elegida y que además tiene un coste exponencial: de una parte, el número de cadenas que se debe adicionar al árbol de prefijos de Moore inicial puede ser exponencial si el valor n máximo es muy grande; además, el número de prefijos del lenguaje hasta dicho tamaño máximo puede ser exponencial dependiendo del lenguaje, al igual que el número de cadenas a considerar en el ciclo de la línea 25. Sin embargo, vale la pena destacar que este hecho no es impedimento para conseguir la convergencia, en este punto lo que se quiere mostrar es que independientemente de la enumeración elegida, es posible construir una muestra tal que si se pone en la entrada del Algoritmo 16, conducirá a la obtención del DFA mínimo del lenguaje objetivo.

Una vez se tiene una muestra característica se puede ejecutar el algoritmo 16 que realiza las mezclas en el orden de la enumeración de Σ^* hasta que no sea posible realizar ninguna mezcla. Las muestras positivas garantizan la presencia de todos los estados del autómata objetivo en $APM(D_+ \cup D'_+)$ y las muestras negativas garantizan que no se habrán realizado mezclas que eliminen alguno de ellos. El mecanismo de mezcla converge porque el proceso elimina cualquier estado redundante y la forma como se construyó la muestra garantiza que lo único que no es redundante es aquello que representa el DFA mínimo del lenguaje objetivo.

Teorema 34 *El procedimiento de mezcla de estados converge en el límite al DFA mínimo del lenguaje objetivo sin importar el orden en que se realicen las mezclas.*

Demostración. El Algoritmo 15 muestra la forma de construir una muestra característica a partir de cualquier enumeración de Σ^*. Por partir de una

Algoritmo 15 *muestraCaracteristica*(E, D_+, A)

Require: E es una enumeración de Σ^*
Require: D_+ es una muestra estructuralmente completa
Require: A es el DFA mínimo que reconoce el lenguaje objetivo L
1: $M = APM(D_+)$ $//M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
2: $n := \text{máx}_{x \in D_+} E(x)$
3: $D'_+ := \{\}$
4: $D'_- := \{\}$
5: **for** $i := 1$ to n **do**
6: $m := E^{-1}(i)$
7: **if** $m \in Pref(L)$ **then**
8: **if** $\Phi(\delta(q_0, m)) = 1$ **then**
9: $D'_+ := D'_+ \cup \{m\}$
10: **else**
11: $D_+ := D_+ \cup \{mw \mid w \in \Sigma^* \wedge mw \in L \wedge |mw| = \text{mín}_{mk \in L} |mk|\}$
12: **end if**
13: **end if**
14: **end for**
15: **for** $x \in D'_+$ **do**
16: Incluir x en M
17: **end for**
18: $d :=$ el grado de distinguibilidad de M
19: $p := \text{máx}_{q \in Q}$ profundidad(q)
20: **for** $i := 1$ to $|Q|$, en el orden de la enumeración E **do**
21: **for** $j := 1$ to $i - 1$ **do**
22: **if** El estado j no ha desaparecido en un agrupamiento anterior **then**
23: $M' := mezcladet(M, i, j)$
24: **if** (i, j) son estados diferentes en el DFA mínimo A **then**
25: **for** $m \in \Sigma^*$ tal que $|m| \leq p + q + 1$ **do**
26: **if** $\neg consistente(M', m\}$ **then**
27: **if** $\Phi(\delta(q_0, m)) = 1$ **then**
28: $D'_+ := D'_+ \cup \{m\}$
29: **else**
30: $D'_- := D'_- \cup \{m\}$
31: **end if**
32: **end if**
33: **end for**
34: **else**
35: $M := M'$
36: **end if**
37: **end if**
38: **end for**
39: **end for**
40: Return $(D_+ \cup D'_+, D'_-)$

muestra estructuralmente completa y contener todos los prefijos del lenguaje en el rango de la enumeración E que se utiliza en la muestra, se garantiza que en el árbol de prefijos de Moore que se construya a partir de esa muestra característica estarán presentes todos los estados del DFA mínimo; también se garantiza que en la muestra existen cadenas positivas y negativas que impiden la mezcla de estados diferentes en el DFA mínimo y garantizan que todos los estados idénticos en el DFA mínimo serán mezclados. En total, se puede ver que en cuanto una muestra cualquiera contenga una muestra característica, el Algoritmo 16 que mezcla estados en el orden de la enumeración E dada, obtendrá a partir de ella el DFA mínimo del lenguaje objetivo. ■

Algoritmo 16 *mezclaDfas*(E, D_+, D_-)

Require: E es una enumeración de Σ^*
Require: D_+ y D_- son una muestra característica
1: $M = APM(D_+)$ //$M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
2: $lista := \{q : q \in Q\}$ // Ordenada ascendentemente por identificador
3: **for** $j \in lista$ **do**
4: $mezclado := falso$
5: **for** $i \in lista$ and $i \ll_E j$ and $\neg mezclado$ **do**
6: $M' := mezcladet(M, i, j)$
7: **if** $consistente(M', D_-)$ **then**
8: $M := M'$
9: $lista := lista - \{q : q \in lista$ and $q \notin Q\}$
10: $mezclado = verdadero$
11: **end if**
12: **end for**
13: **end for**
14: Return M

Ejemplo 35 *A continuación se ilustra el proceso con un ejemplo. Sea* $\Sigma = \{a, b\}$*, sea la máquina de Moore de la Figura 3.15 el DFA mínimo de un lenguaje objetivo, sea* $D_+ = \{ab, abba\}$ *la muestra de entrada estructuralmente completa y sea* E *la enumeración de* Σ^* *que se presenta en el Cuadro 3.2.*

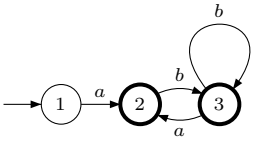


Figura 3.15: Ejemplo Convergencia de la Mezcla en DFAs, DFA Mínimo del Lenguaje Objetivo

El árbol de prefijos de Moore correspondiente a la muestra de entrada se ilustra en la Figura 3.16. Notar que el identificador del estado que se alcanza a través del prefijo x *es el valor* $E(x)$*.*

El primer paso en la construcción de la muestra característica consiste en completar el árbol de prefijos de Moore con todos los prefijos de las cadenas de

Cuadro 3.2: Ejemplo Convergencia de la Mezcla en DFAs, Enumeración E de Σ^*

p	$E(p)$	p	$E(p)$	p	$E(p)$	p	$E(p)$
ε	10	aab	2	aaab	17	baab	25
a	9	aba	1	aaba	18	baba	26
b	8	abb	11	aabb	19	babb	27
aa	7	baa	12	abaa	20	bbaa	28
ab	6	bab	13	abab	21	bbab	29
ba	5	bba	14	abba	22	bbba	30
bb	4	bbb	15	abbb	23	bbbb	31
aaa	3	aaaa	16	baaa	24		

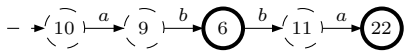


Figura 3.16: Ejemplo Convergencia de la Mezcla en DFAs, Árbol de Prefijos de Moore de la Muestra de Entrada

Σ^* *que pertenecen al lenguaje y que tienen un valor en la enumeración menor que el máximo actual. En el ejemplo el máximo actual es el valor* $E(abba) = 22$ *y los prefijos que pertenecen al lenguaje cuyo valor en la enumeración es menor a* 22 *son:* $E(a) = 9$*,* $E(aba) = 10$*,* $E(abb) = 11$ *y* $E(abab) = 21$*, el árbol de prefijos de Moore aumentado con estas cadenas se muestra en la Figura 3.17.*

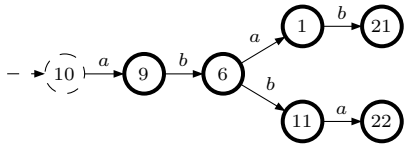


Figura 3.17: Ejemplo Convergencia de la Mezcla en DFAs, Árbol de Prefijos de Moore Extendido

El siguiente paso en la construcción de la muestra característica consiste en adicionar todas las muestras, positivas y negativas necesarias para distinguir los estados del DFA mínimo en el árbol de prefijos de Moore extendido. Para esto se considera cada pareja de estados y se observa qué cadena(s) hace(n) falta en la muestra para impedir que dichos estados se agrupen, notar que si los estados realmente son el mismo no se encontrará ninguna cadena que permita distinguirlos. Por ejemplo, al considerar la primera pareja: $(1, 6)$ *se puede observar en la Figura 3.15 que si se adiciona la muestra negativa abaa esto impediría la mezcla de dichos estados. Actuando de forma análoga para el resto de parejas en el mismo orden en que serán consideradas durante el proceso de mezcla, se obtienen los conjuntos de muestras* $D'_+ = \{\}$ *y* $D'_- = \{\varepsilon, abaa\}$*. Así, se consigue el conjunto característico que es* $(D_+ \cup D'_+, D'_-)$*. Para el ejemplo, esto es: ((ab,abba),(ε,abaa)). El árbol de prefijos de Moore correspondiente a las muestras positivas, que es el punto de partida para el algoritmo de mezcla, es el que*

ya se ha mostrado en la Figura 3.17.

Teniendo ya una muestra característica, se empiezan a realizar comparaciones de estados en el orden de la enumeración, en este caso la primera pareja a compararse es $(1, 6)$ *que no se puede mezclar, la mezcla* $(1, 9)$ *se efectúa y la máquina resultante está en la Figura 3.18.*

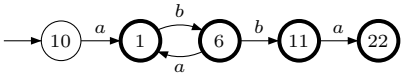


Figura 3.18: Ejemplo Convergencia de la Mezcla en DFAs, Árbol de Prefijos de Moore después de mezclar 1 y 9

Se continúa probando mezclar la pareja $(1, 10)$ *que no produce una máquina consistente con las muestras negativas en* $D'\_$ *igual que las parejas* $(6, 10)$ *y* $(1, 11)$*, luego se intenta mezclar la pareja* $(6, 11)$ *que produce la máquina correspondiente al DFA mínimo de lenguaje objetivo, que se muestra en la Figura 3.19.*

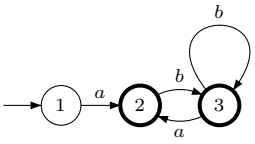


Figura 3.19: Ejemplo Convergencia de la Mezcla en DFAs, Resultado del Algoritmo de Mezcla

Ejemplo 36 *Con el fin de ilustrar cómo varía el proceso al cambiar de enumeración, en este ejemplo se trabaja sobre el mismo lenguaje objetivo y la misma muestra inicial que en el Ejemplo 35, pero utilizando una enumeración en orden lexicográfico, tal como se muestra en el Cuadro 3.3.*

Cuadro 3.3: Ejemplo Convergencia de la Mezcla en DFAs, Otra Enumeración E de Σ^*

p	$E(p)$	p	$E(p)$	p	$E(p)$	p	$E(p)$
ε	1	aab	9	aaab	17	baab	25
a	2	aba	10	aaba	18	baba	26
b	3	abb	11	aabb	19	babb	27
aa	4	baa	12	abaa	20	bbaa	28
ab	5	bab	13	abab	21	bbab	29
ba	6	bba	14	abba	22	bbba	30
bb	7	bbb	15	abbb	23	bbbb	31
aaa	8	aaaa	16	baaa	24		

El árbol de prefijos de Moore correspondiente a la muestra de entrada se ha presentado previamente en la Figura 3.16.

El primer paso en la construcción de la muestra característica consiste en completar el árbol de prefijos de Moore con todos los prefijos de las cadenas de Σ^* *que pertenecen al lenguaje y que tienen un valor en la enumeración menor que el máximo actual. En el ejemplo el máximo actual es el valor* $E(abba) = 22$ *y los prefijos que pertenecen al lenguaje cuyo valor en la enumeración es menor a* 22 *son:* $E(a) = 2$, $E(aba) = 10$, $E(abb) = 11$ *y* $E(abab) = 21$, *el árbol de prefijos de Moore aumentado con estas cadenas y etiquetado de acuerdo a la nueva enumeración se muestra en la Figura 3.20.*

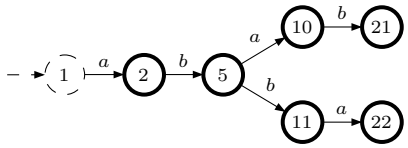


Figura 3.20: Ejemplo Convergencia de la Mezcla en DFAs, Árbol de Prefijos de Moore Extendido

A continuación se deben adicionar las muestras positivas y negativas que sean necesarias para distinguir los estados del DFA mínimo en el árbol de prefijos de Moore extendido. Se comienza a considerar las parejas en el orden en que lo hará el algoritmo de mezcla: los estados $(1, 2)$ *corresponden a estados diferentes del DFA mínimo, por lo tanto se busca una cadena que impida agruparlos, por ejemplo, la cadena* ε *que no pertenece al lenguaje objetivo; la misma cadena sirve para impedir la mezcla de los estados* $(1, 5)$. *Para evitar mezclar los estados* $(2, 5)$ *se debe adicionar la cadena negativa* aa, *y para evitar mezclar los estados* $(1, 10)$ *también basta con incluir la cadena vacía* ε *como muestra negativa. En el caso de los estados* $(2, 10)$ *ellos se pueden mezclar. Siguiendo en la misma forma se consideran las parejas de estados* $(1, 11)$ *y* $(2, 11)$ *que no son mezclables pero que se pueden diferenciar con las mismas cadenas negativas ya consideradas:* ε *y* aa *respectivamente. Finalmente se considera la pareja* $(5, 11)$ *que se puede mezclar. En total, se obtienen los conjuntos* $D'_+ = \{\}$ *y* $D'_- = \{\varepsilon, aa\}$. *Como no hay nuevas muestras positivas para adicionar, el árbol de prefijos de Moore con el que comienza a funcionar el algoritmo, es el mismo que se ha presentado en la Figura 3.20.*

El algoritmo empieza a realizar las comparaciones del caso: las parejas de estados $(1, 2)$, $(1, 5)$, $(2, 5)$ *y* $(1, 10)$ *no se pueden mezclar porque de hacerlo se acepta alguna cadena negativa. La mezcla de* $(2, 10)$ *se realiza y modifica la máquina tal como se muestra en la Figura 3.21. A continuación se consideran las parejas* $(1, 11)$, $(2, 11)$ *que no conducen a mezclas permanentes y la pareja* $(5, 11)$ *que modifica la máquina produciendo el DFA mínimo del lenguaje objetivo, el cual se ha mostrado previamente en la Figura 3.19.*

Los Ejemplos 35 y 36 ilustran el funcionamiento del método para construir una muestra característica que garantiza la convergencia del Algoritmo de mezcla de estados 16. Ellos parten de dos enumeraciones diferentes, esto con el fin de dar mayor claridad sobre la manera de proceder del método y de ilustrar el

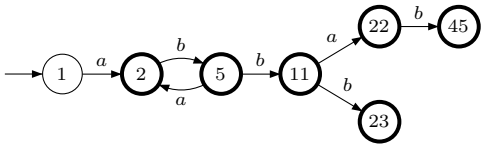


Figura 3.21: Ejemplo Convergencia de la Mezcla en DFAs, Árbol de Prefijos de Moore Luego de Mezclar los Estados 2 y 10

hecho que diferentes enumeraciones conducen a diferentes muestras características, pero todas las muestras características conducen al mismo autómata, que es lo que se ha argumentado en forma general en la demostración del Teorema 34.

3.5.2. Convergencia de la Mezcla de Estados en Máquinas No Deterministas

Como se verá en esta sección, es posible obtener convergencia en el límite independientemente del orden en que se mezclan los estados de una máquina no determinista [GdPAR07]. El argumento es el siguiente: dada una muestra universal es posible agrupar y reagrupar las cadenas que la componen, de modo que las muestras negativas sean suficientes para garantizar que cualquier partición de los estados de $AM(D_+)$ produce un autómata que reconoce el lenguaje objetivo L.

En primer lugar, se va a definir con detalle un conjunto finito de palabras llamado *muestra universal*; para eso se deben realizar definiciones previas acerca de la irreducibilidad de un autómata y de la relación entre los autómatas que reconocen un lenguaje y su correspondiente autómata universal.

Definición 37 *Un autómata $\mathcal{A}$ es irreducible en un lenguaje regular L si y sólo si $L(\mathcal{A}) \subseteq L$ y para toda partición π de los estados de $\mathcal{A}$, $L(\mathcal{A}/\pi) - L \neq \emptyset$.*

Proposición 38 *[GdP05] Sea $\mathcal{A}$ un autómata que acepta L y sea $\mathcal{U}$ el autómata universal de L. Sea φ el morfismo que transforma $\mathcal{A}$ en $\mathcal{U}$. Si existen k estados de $\mathcal{A}$ $q_1, ..., q_k$ tales que $\varphi(q_1) = \varphi(q_k)$ entonces los estados $q_1, ..., q_k$ son mezclables.*

Lema 39 *Sea $\mathcal{A}$ un autómata irreducible (ver Definición 37) con respecto al lenguaje regular L. Entonces $\mathcal{A}$ es isomórfico a un subautómata de $\mathcal{U}$ (el autómata universal de L).*

Demostración. Como $\mathcal{A}$ es irreducible en L, $L(\mathcal{A}) \subseteq L$ y no tiene estados mezclables. Entonces el morfismo definido como $\varphi(q) = \cap_{u \in L_q} u^{-1}L$ es inyectivo. ■

Corolario 40 *Sean $D \subseteq L$ finito y π una partición de los estados de $AM(D)$ tal que $AM(D)/\pi$ es irreducible en L. Entonces $AM(D)$ es isomórfico a un subautómata de $\mathcal{U}$ (el autómata universal de L).*

Ahora se define lo que es un camino de aceptación y la yuxtaposición de subautómatas, con lo cual se procede a definir la muestra universal D, que tiene la propiedad que cualquier partición π tal que $AM(D)/\pi$ es irreducible en L, define un subautómata de $\mathcal{U}$ (el autómata universal de L) que acepta exactamente L.

Definición 41 *Sea $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ un NFA y sea $x = a_1a_2...a_n \in L(\mathcal{A})$. Un camino de aceptación para x en $\mathcal{A}$ es una secuencia de arcos $\langle (q_1, a_1, q_2), (q_2, a_2, q_3)...(q_n, a_n, q_{n+1})\rangle$ con $q_1 \in I$, $q_{n+1} \in F$.*

Definición 42 *Dado un camino $\mathcal{C} = \langle (q_1, a_1, q_2), (q_2, a_2, q_3), ...(q_n, a_n, q_{n+1})\rangle$, el subautómata de $\mathcal{A}$ inducido por $\mathcal{C}$ es $\mathcal{A}_\mathcal{C} = (Q', \Sigma', \delta', \{q_1\}, \{q_{n+1}\})$ donde Q' es el conjunto de estados distintos de $q_1, q_2, ...q_{n+1}$, Σ' es el conjunto de símbolos distintos en $(a_1a_2...a_n)$.*

Definición 43 *Dado un NFA $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ y la colección de subautómatas $\{\mathcal{A}_i\}_{i=1}^n$, donde $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, I_i, F_i)$ la yuxtaposición de los subautómatas es el subautómata $\mathcal{A}' = (Q', \Sigma', \delta', I', F')$ con $Q' = \bigcup_{i=1}^n Q_i$, $\Sigma' = \bigcup_{i=1}^n \Sigma_i$, $I' = \bigcup_{i=1}^n I_i$, $F' = \bigcup_{i=1}^n F_i$ y para todo $q \in Q'$ y todo $a \in \Sigma'$, $\delta'(q, a) = \bigcup_{i=1}^n \delta_i(q, a)$. Es claro que $\bigcup_{i=1}^n L(\mathcal{A}_i) \subseteq L(\mathcal{A}') \subseteq L(\mathcal{A})$.*

Sea $\mathcal{A}$ un autómata, sea $x \in L(\mathcal{A})$ y sea $\mathcal{C}_x$ el conjunto de los caminos de aceptación para x en $\mathcal{A}$. Dados $\{x_1, x_2...x_n\} \subset L(\mathcal{A})$, se puede asociar a todo conjunto $\{c_1, c_2...c_n\}$, con $c_i \in \mathcal{C}_{x_i}$, un subautómata $\mathcal{A}_{\{c_i...c_n\}}$ obtenido por yuxtaposición de los subautómatas $\mathcal{A}_{c_i}$, cada uno asociado a uno de los caminos.

Dado un lenguaje regular L, una *muestra universal* para L es cualquier subconjunto finito $D \subseteq L$ tal que cada subautómata de $\mathcal{U}$ asociado con cada conjunto de caminos (uno por cada palabra de D), reconoce L. Más formalmente:

Definición 44 *Sea $D = \{x_1, ..., x_n\} \subseteq L$. Para todo $i = 1..n$, sea $\{c_1^{(i)}, ..., c_{n_i}^{(i)}\}$ el conjunto de caminos de aceptación para x_i, y $\{\mathcal{A}_1^{(i)}, ..., \mathcal{A}_{n_i}^{(i)}\}$ el conjunto de subautómatas inducidos. D es una muestra universal para L si toda yuxtaposición de los subautómatas $\mathcal{A}_{j_1}^{(1)}, \mathcal{A}_{j_2}^{(2)}..., \mathcal{A}_{j_n}^{(n)}$ para $j_k = 1, ..., n_k$ reconoce L.*

Ejemplo 45 *Con el fin de entender mejor las definiciones previas, se propone el siguiente ejemplo:*

Sea $L = aaa^*$. La Figura 3.22(a) presenta un autómata determinista para L y la Figura 3.22 (b) el autómata universal para L.

El conjunto $\{aaaa\}$ es una muestra estructuralmente completa para $\mathcal{A}$, ya que utiliza todas las transiciones de $\mathcal{A}$. El trellis de todos los caminos que aceptan $\{aaaa\}$ en $\mathcal{U}$ está dibujado en la Figura 3.23 (a). Dado que el alfabeto consta de un sólo símbolo, se describen los caminos usando únicamente los estados visitados, así el conjunto de caminos de aceptación es: $\mathcal{C}_{aaaa} = \{$11123, 11223, 11233, 12123, 12223, 12233, 12323, 12333$\}$.

Todos los subautómatas de $\mathcal{U}$ inducidos por $\mathcal{C}_{aaaa}$, excepto los inducidos por 12123 *y por* 12323*, aceptan L. El subautómata inducido por esos caminos se presenta en la Figura 3.24. Se puede ver que esos autómatas no aceptan, por*

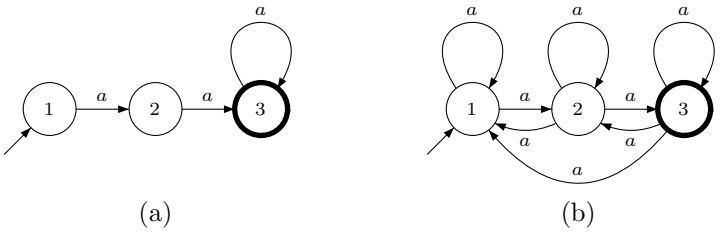


Figura 3.22: (a) Autómata Determinista para el lenguaje $L = aaa^*$. (b) Automata Universal de L.

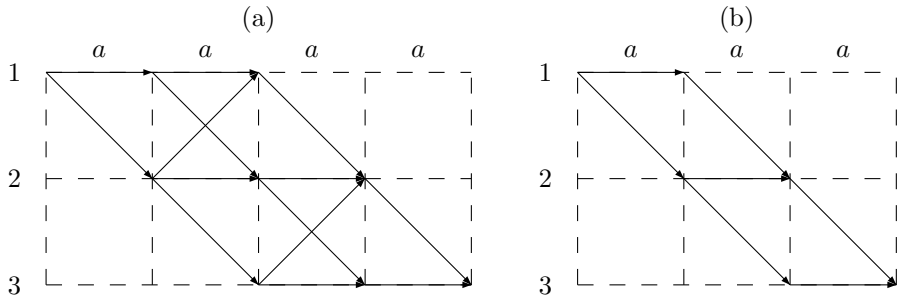


Figura 3.23: Trellis con los Caminos de Aceptación en $\mathcal{U}$: (a) Para la palabra $aaaa$, (b) Para aaa.

Figura 3.24: Subautómata inducido por los caminos 12123 y 12323, que no aceptan L.

ejemplo, la palabra aaa.

Los caminos que aceptan la palabra aaa están dibujados en la Figura 3.23 (b), ellos son $\mathcal{C}_{aaa} = \{1123, 1223, 1233\}$. *Los subautómatas asociados con esos caminos reconocen L y también lo hacen la yuxtaposición de ellos con aquellos dibujados en la Figura 3.24. Por lo tanto, la muestra* $\{aaa, aaaa\}$ *es universal para L. De hecho,* $\{aaa\}$ *es universal para L. El autómata* $AM(aaa)$ *se dibuja en la Figura 3.25. Es fácil darse cuenta que las únicas particiones en sus conjuntos de estados que son irreducibles y consistentes con* $\overline{L}$ *son* $\{\{1,2\},3,4\}$, $\{\{2,3\},1,4\}$ *y* $\{\{3,4\},1,2\}$.

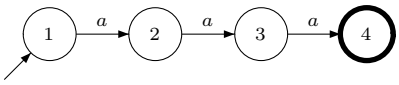


Figura 3.25: Ejemplo Convergencia de la Mezcla en NFAs, $AM(aaa)$.

Teorema 46 *Sea L un lenguaje regular y sea D una muestra universal para L. Sea $\mathcal{A}$ el autómata maximal para D y sea π cualquier partición de los estados de $\mathcal{A}$ tal que $\mathcal{A}/\pi$ es irreducible en L. Entonces $L(\mathcal{A}/\pi) = L$.*

Demostración. Como $\mathcal{A}/\pi$ es irreducible en L, $\mathcal{A}/\pi$ es isomórfico a un subautómata de $\mathcal{U}$. Como D es universal para L, $L(\mathcal{A}/\pi) = L$. ■

La muestra universal es necesaria pero no suficiente para lograr la convergencia de un algoritmo de mezcla de estados. Hace falta acompañarla de una muestra negativa apropiadamente construida para garantizar que todas las particiones que se produzcan reconozcan el lenguaje objetivo.

Lema 47 *Dado un conjunto finito $D_+ \subset L$, existe un conjunto finito $D_- \subset \overline{L}$ tal que para cualquier partición π del conjunto de estados de $AM(D_+)$ el autómata $AM(D_+)/\pi$ es irreducible en $\overline{D_-}$, entonces $AM(D_+)/\pi$ es irreducible en L.*

Demostración. Sean p y q dos estados cualquiera de $AM(D_+)$ tales que $L(mezcla(AM(D_+), p, q)) \not\subset L$. Es suficiente para agregar a D_- cualquier palabra perteneciente a $L(mezcla(AM(D_+), p, q)) - L$. ■

Corolario 48 *Sea D_+ una muestra universal para L, existe un conjunto finito D_- tal que cualquier partición π del conjunto de estados de $AM(D_+)$ irreducible en $\overline{D_-}$ verifique que $L(AM(D_+)/\pi) = L$.*

Teorema 49 *Sea L un lenguaje regular y sea D una muestra universal para L. Sea $\mathcal{A}$ el autómata maximal para D y sea π cualquier partición de los estados de $\mathcal{A}$ tal que $\mathcal{A}/\pi$ es irreducible en L. Entonces $L(\mathcal{A}/\pi) = L$.*

Demostración. Como $\mathcal{A}/\pi$ es irreducible en L, $\mathcal{A}/\pi$ es isomórfico a un subautómata de $\mathcal{U}$. Como D es universal para L, $L(\mathcal{A}/\pi) = L$. ■

Una vez se ha encontrado el conjunto de muestras negativas necesarias para acompañar la muestra universal, el introducir nuevas muestras positivas causa que posiblemente sea necesario introducir también nuevas muestras negativas.

Proposición 50 *Sea D_+ una muestra universal para L y sea D_- un conjunto finito que garantiza que cualquier partición en los estados de $AM(D_+)$ produce un autómata que reconoce L. Si agregamos nuevas palabras de L al conjunto D_+, el nuevo conjunto continúa siendo una muestra universal para L, pero D_- puede no garantizar que cualquier partición del autómata maximal de D_+ todavía acepte L.*

Ejemplo 51 *En el siguiente ejemplo se ilustra un caso en el que introducir nuevas muestras positivas hace que las muestras negativas actuales ya no garanticen que cualquier partición del autómata maximal de D_+ acepte L. Sea $L = a^* + b^*$. El conjunto $D_+ = \{\lambda, a, b, aa, bb, aaa, bbb\}$ es universal para L y $D_- = \{ab, ba, abb, bba, abbb, bbba\}$ es tal que para cualquier partición de $AM(D_+)$ irreducible en $\overline{D_-}$, el autómata cociente con respecto a esa partición acepta L.*

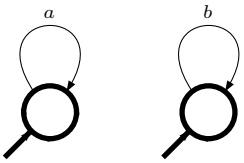


Figura 3.26: Ejemplo Convergencia de la Mezcla en NFAs, Resultado de Aplicar una Partición que Acepta L

Suponiendo que la partición elegida produce el autómata que se presenta en la Figura 3.26, si se adiciona la palabra bbbb a D_+, manteniendo sin cambios D_-, habría una partición irreducible en $\overline{D_-}$ que daría origen al autómata de la Figura 3.27.

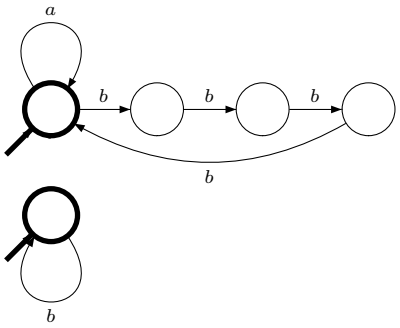


Figura 3.27: Ejemplo Convergencia de la Mezcla en NFAs, Resultado de una Partición Luego de Adicionar la Muestra Positiva *bbbb*

Para evitar esta posibilidad, debe adicionarse una nueva palabra abbbb a D_-. Se puede ver que por cada palabra de la forma b^n que se adicione a D_+, deberá adicionarse por lo menos la palabra ab^n a D_- para mantener la propiedad de la muestra.

Con base en las definiciones y teoremas previos, se propone una manera de proceder con una muestra universal, de manera que se logra la convergencia en el límite. Se da el nombre de algoritmo *OIL* a esta estrategia, que se muestra en el Algoritmo 17.

El Algoritmo 17 recibe como entrada un conjunto de bloques de muestras positivas y negativas para el lenguaje objetivo L (esta es una generalización, en realidad cada bloque podría contener una sóla palabra) y obtiene, en el límite, un autómata que reconoce L. El método comienza construyendo el autómata

Algoritmo 17 $OIL((D_+^{(1)}, D_-^{(1)}), \ldots, (D_+^{(n)}, D_-^{(n)})$

Require: $(D_+^{(1)}, D_-^{(1)}), \ldots, (D_+^{(n)}, D_-^{(n)})$ son bloques no vacíos

1: $M = AM(D_+^{(1)})$ $//M = (Q, \Sigma, \{0, 1, ?\}, \delta, \Phi, I)$
2: $D_- := D_-^{(1)}$
3: $E :=$Una enumeración de los estados de $AM(D_+^{(1)})$ usando los enteros entre 1 y N_1, donde N_1 es el número de estados de $AM(D_+^{(1)})$
4: $M := mezclaNfas(E, M, D_-)$
5: $i := 2$
6: **while** $i \leq n$ **do**
7: $D_- := D_- \cup D_-^{(i)}$
8: $M' := AM(D_+^{(i)})$ $//M' = (Q', \Sigma, \{0, 1, ?\}, \delta', \Phi', I')$
9: $N_i := |Q'|$
10: **if** $\neg consistente(M, D_-^{(i)})$ or $\neg consistente(M, D_+^{(i)})$ **then**
11: **if** $\neg consistente(M, D_-^{(i)})$ **then**
12: $i := 1$
13: $M := AM(D_+^{(1)}))$
14: $E :=$Una enumeración de los estados de $AM(D_+^{(1)})$ usando los enteros entre 1 y N_1, donde N_1 es el número de estados de $AM(D_+^{(1)})$
15: **else**
16: $N_i := |Q'|$
17: Eliminar de M' las muestras consistentes en M
18: $M := M \cup M'$ $//M = (Q \cup Q', \Sigma, \{0, 1, ?\}, \delta \cup \delta', \Phi \cup \Phi', I \cup I')$
19: Extender E asignando identificador aleatorio a los estados de Q' con valores enteros entre $(\sum_{j=1}^{i-1} N_j) + 1$ y $\sum_{j=1}^{i} N_j)$
20: **end if**
21: $M := mezclaNfas(E, M, D_-)$
22: **end if**
23: $i := i + 1$
24: **end while**
25: Return M

maximal del primer bloque de muestras positivas $D_+^{(1)}$, genera un orden aleatorio sobre los estados e invoca la subrutina *mezclaNfas* para mezclar todos los estados posibles en dicho orden hasta obtener un autómata irreducible en $D_-^{(1)}$. La subrutina *mezclaNfas*, que se presenta en el Algoritmo 18, es análoga a la subrutina *mezclaDfas* que se presentó en el Algoritmo 16, salvo que en este caso se inicia con el autómata maximal en lugar del árbol de prefijos de Moore y que el modelo en este caso es no determinista; además se invoca a la subrutina *mezcla* (ver Algoritmo 12) que realiza la mezcla sin propagación de dos estados.

Algoritmo 18 *mezclaNfas*(E, D_+, D_-)

Require: E es una enumeración de Σ^*
Require: D_+ y D_- son una muestra característica
1: $M = AM(D_+)$ $//M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, I)$
2: $lista := \{x : x \in Q\}$ // Ordenanda ascendentemente de acuerdo a la enumeración
3: **for** $j \in lista$ **do**
4: $mezclado := falso$
5: **for** $i \in lista$ and $i \ll j$ and $\neg mezclado$ **do**
6: $M' := mezcla(M, i, j)$ $//M = (Q', \Sigma, \{0, 1, ?\}\delta', \Phi', I')$
7: **if** $M' \neq M$ and $j \in I$ **then**
8: $I' := I' \cup \{i\}$
9: **end if**
10: **if** $consistente(M', D_-)$ **then**
11: $M := M'$
12: $lista := lista \setminus \{x : x \in lista$ and $x \notin Q\}$
13: $mezclado = verdadero$
14: **end if**
15: **end for**
16: **end for**
17: Return M

En cada iteración i del algoritmo se considera un nuevo bloque $(D_+^{(i)}, D_-^{(i)})$. Se evalúa la consistencia de estas muestras con el modelo actual, se acumulan las muestras negativas, se contabilizan los identificadores correspondientes a las muestras positivas y dependiendo del resultado de la evaluación se procede de así:

- Si el modelo actual M es consistente tanto con las muestras positivas como con las negativas se pasa al siguiente bloque.
- Si el modelo actual M no es consistente con las muestras negativas $D_-^{(i)}$, se descarta la máquina actual y se reinicia el proceso considerando el primer bloque de muestras. Sin embargo, al revisitar el primer bloque se hace utilizando las muestras positivas $D_1^{(1)}$ y el acumulado de muestras negativas que había al momento de encontrar la inconsistencia D_-. Se ejecuta el algoritmo de mezcla *mezclaNfas*.
- Si no hay consistencia con las muestras positivas $D_+^{(i)}$, se descartan del autómata maximal M' las palabras consistentes en M, se adiciona esta

nueva información al modelo actual M y se ejecuta el algoritmo de mezcla *mezclaNfas*.

Teorema 52 *El algoritmo* OIL *reconoce en el límite la familia de los lenguajes regulares.*

Demostración. Sea L un lenguaje. Sea $(D_+^{(1)}, D_-^{(1)}), (D_+^{(2)}, D_-^{(2)}) \ldots$, una presentación completa de L en bloques. Existe entonces $n \geq 0$ tal que $D_+ = \bigcup_{i=1}^{n} D_+^{(i)}$ es una muestra universal. Existe también $m \geq n$ tal que $D_- = \bigcup_{i=1}^{m} D_-^{(i)}$ garantiza que la mezcla de estados en el autómata maximal de D_+ que produce un autómata irreducible en $\overline{D_-}$ es un subautómata de $\mathcal{U}$ que acepta L.

En caso que antes de procesar el bloque $(D_+^{(m)}, (D_-^{(m)})$ no se haya alcanzado la convergencia, el autómata actual acepta palabras que no están en L y por lo tanto no es consistente con D_-. El algoritmo entonces procesa todos los bloques hasta el m-esimo pero usando D_- en vez de $D_-^{(i)}$. En este caso el algoritmo va a converger después de procesar el n-esimo bloque.
■

Si n es el número de bloques en que se ha dividido la entrada y $|D_+|$ (resp. $|D_-|$) es la suma de las longitudes de las muestras positivas (resp. negativas), el algoritmo tiene un coste temporal de $O(|D_+|^2|D_-|n^2)$.

Ejemplo 53 *A continuación se ilustra el funcionamiento del algoritmo* OIL *con un ejemplo. Sea el lenguaje objetivo* $L = 0(0+1)^* \cap (0+1)^*0 - (0+1)^*00(0+1)^*$, *es decir, el conjunto de cadenas de enteros binarios que empiezan y terminan por* 0 *y no contienen el segmento* 00 *en su interior. El autómata determinista mínimo que reconoce* L *se presenta en la Figura 3.28(a), mientras que el autómata universal para* L *está en la Figura 3.28(b).*

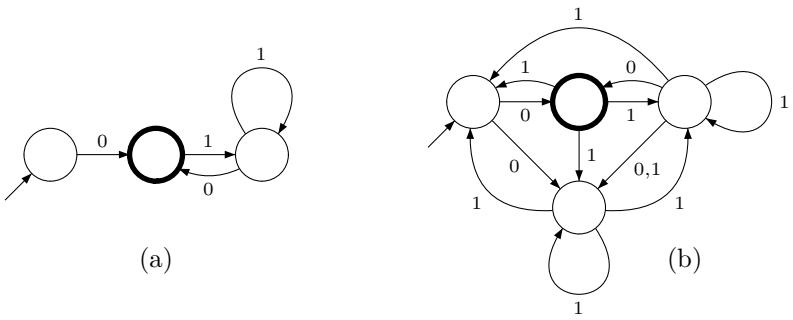


Figura 3.28: Autómata Determinista Mínimo y Autómata Universal para $L = 0(0+1)^* \cap (0+1)^*0 - (0+1)^*00(0+1)^*$

Al ejecutar el algoritmo mezclaNfas *con diferentes enumeraciones de los estados se han obtenido los autómatas de la Figura 3.29, comparándolos con el*

autómata universal de la figura 3.28 parte (b), se aprecia que todas las salidas son subautómatas del autómata universal para el lenguaje objetivo. Los experimentos fueron realizados con 100.000 palabras de longitud máxima 18 agrupadas en bloques de 1000 *palabras cada uno. No se controló la proporción entre muestras positivas y negativas en cada bloque. El experimento que se ilustra se ejecutó cinco veces, en tres ejecuciones la convergencia se alcanzó después de dos bloques, mientras que en los otros tres casos, se necesitaron 18 bloques para alcanzar la convergencia. Se realizaron otras pruebas con mayor cantidad de repeticiones (100 repeticiones sobre 100000 muestras en bloques de 1000) que no se reportan por haber mostrado un comportamiento análogo.*

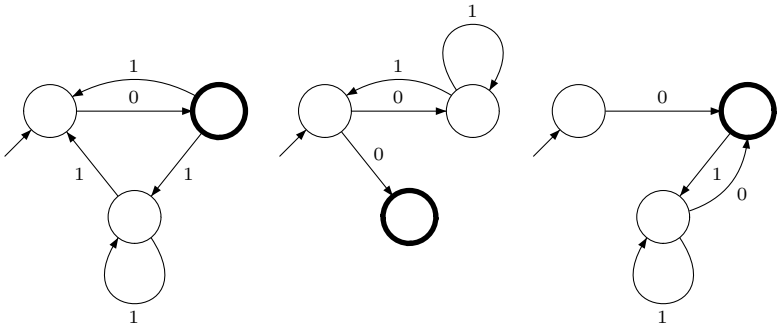


Figura 3.29: Tres Salidas Diferentes del Algoritmo *OIL* con Diferentes órdenes de Mezcla de Estados.

3.6. Sumario

En este capítulo se ha presentado la mezcla de estados como mecanismo de generalización en algoritmos de inferencia gramatical de lenguajes regulares. Al momento de extender este mecanismo a la inferencia de modelos no deterministas se ocurren varias maneras de hacerlo, aquí se ha presentado la forma propia como se enfocó la mezcla no determinista en esta investigación. De igual manera, se ha presentado un breve análisis de la importancia de considerar el orden de mezcla como una variable determinante en el desempeño práctico de cualquier estrategia de inferencia gramatical. Estos conceptos son aplicados intensivamente en el diseño de los algoritmos que se presentan en el capítulo siguiente.

El aporte más importante de este capítulo es la demostración de que la convergencia de un algoritmo de mezcla de estados es independiente del orden en que se realicen las mezclas. Este resultado es muy importante dado que, hasta la fecha, las estrategias EDSM se han considerado heurísticas, ya que no estaba demostrada su convergencia y ahora lo está. Además, el resultado se ha demostrado tanto en el caso determinista como no determinista, lo que puede ser de utilidad en el diseño de nuevos algoritmos de inferencia de NFAs.

El argumento de la demostración se basa, en el caso determinista, en mostrar que dada una enumeración cualquiera de Σ^* y una muestra inicial, es posible

construir a partir de ella una muestra característica tal que un algoritmo de mezcla de estados converja al DFA mínimo del lenguaje objetivo al recibirla como entrada. En el caso no determinista, se muestra que es posible extender una muestra dada para convertirla en una muestra universal y posteriormente complementarla con otras cadenas necesarias para garantizar la convergencia de la inferencia a un subautómata del autómata Universal del lenguaje objetivo.

Capítulo 4

Algoritmos Propuestos

Este capítulo presenta los algoritmos de inferencia gramatical para lenguajes regulares que se han considerado en este trabajo, la Sección 4.1 describe los algoritmos ya existentes que se han estudiado con detalle: RPNI, Gold y Delete2. En las secciones siguientes se presentan los algoritmos nuevos, cuyo diseño y prueba constituyen la mayor parte de esta investigación. Los algoritmos se han agrupado de acuerdo al tipo de mezcla que hacen: determinista o no determinista. Dado que los elementos teóricos que se van a aplicar ya han sido presentados, en este capítulo se centra el interés en la descripción de los algoritmos, en el análisis de su convergencia, su complejidad temporal y en ilustrar su funcionamiento con ejemplos cuando ello sea útil. Se difiere hasta el próximo capítulo la presentación de resultados experimentales y la comparación empírica entre los algoritmos.

Estos algoritmos fueron diseñados con el propósito de combinar elementos adicionales a la mezcla de estados con el fin de mejorar la tasa de reconocimiento y/o disminuir el tamaño de las hipótesis generadas por los algoritmos conocidos. En particular, se han enfocado los esfuerzos a superar los mejores resultados conocidos en la inferencia de lenguajes regulares que son expresiones regulares o NFAs.

Algunos de los algoritmos descritos en este capítulo resultan de combinar el esquema básico de inferencia por mezcla de estados propio de algoritmos muy conocidos como *RPNI* y *redBlue* con el uso de información proveniente de las relaciones de inclusión entre los lenguajes residuales asociados a los estados de un modelo de inferencia. Otros algoritmos han buscado combinar la mezcla de estados con el uso de información procedente de las relaciones de inclusión en nuevas formas.

Las relaciones de inclusión están presentes tanto en los DFAs como en los RFSAs, ya que ambos tipos de autómatas tienen un lenguaje residual asociado a cada uno de los estados que lo componen. En esta investigación se han explorado dos formas concretas de aprovechar la información que pueden aportar las relaciones de inclusión:

- Definir nuevos valores de salida: se utiliza en los algoritmos IRPNI1, IRP-

NI2, RBIR y RRB y conduce a algoritmos que producen autómatas deterministas.

- Adicionar nuevos arcos: se utiliza en los algoritmos NRPNI y MRIA, dando lugar a algoritmos que producen modelos no deterministas.

4.1. Algoritmos Básicos

4.1.1. Algoritmo *RPNI*

El algoritmo *RPNI* (Regular Positive and Negative Inference) [OG92] recibe una muestra positiva y negativa del lenguaje objetivo como entrada y genera como salida, en tiempo polinomial, un autómata determinista consistente con dicha entrada. Este algoritmo converge en el límite al autómata mínimo del lenguaje objetivo. Antes de entrar en materia, vale la pena comentar que simultáneamente con *RPNI* pero en forma independiente, apareció otro algoritmo llamado *traxbar* [Lan92], que describe prácticamente la misma estrategia.

El algoritmo *RPNI* (D_+, D_-) (ver Algoritmo 19) comienza construyendo el árbol de Prefijos de Moore (APM) a partir de la muestra de entrada; luego intenta mezclar cada estado con los estados previos en orden lexicográfico y propaga la mezcla para mantener el determinismo del autómata. Si el autómata resultante acepta alguna muestra negativa, la operación se deshace. La mezcla de estados se realiza en la subrutina *mezcladet* la cual garantiza el determinismo de los autómatas intermedios y por lo tanto del autómata final. La subrutina *mezcladet* puede modificar el valor de la función de salida de los estados que no desaparecen, en caso que alguno de ellos tuviera, previamente a la mezcla, un valor indefinido.

Algoritmo 19 $RPNI(D_+, D_-)$

```
1: M := APM(D+, D-)
2: lista := {u0, u1, ..., ur} //estados de M en orden lexicográfico, u0 = λ//
3: lista' := {u1, ..., ur}
4: q := u1
5: while lista' ≠ ∅ do
6:    for p ∈ lista and p ≪ q (en orden lexicográfico) do
7:       if mezcladet(M, p, q) ≠ M then
8:          M := mezcladet(M, p, q)
9:          exit for
10:      end if
11:   end for
12:   lista := Eliminar de lista los estados que no están en M
13:   lista' := Eliminar de lista' los estados que no están en M
14:   q := first(lista')
15: end while
16: Return M
```

El algoritmo RPNI converge en el límite al DFA mínimo del lenguaje objetivo [OG92] siempre y cuando la muestra de entrada $D_+ \cup D_-$ tenga las siguientes características:

- Que D_+ sea estructuralmente completa (ver Definición 18).
- Que D_- tenga las cadenas necesarias para que si se intenta mezclar dos estados distintos pertenecientes al núcleo del lenguaje objetivo $N(L)$, exista una cadena en D_- que impida la mezcla.

Vale resaltar que el número de muestras requerido para que se cumplan estas características es polinomial con respecto al número de estados del autómata determinista mínimo del lenguaje objetivo. [OG92]

La complejidad del algoritmo *RPNI* se puede expresar en términos del tamaño de la muestra n; analizando el código del Algoritmo 19, se puede ver que *RPNI* es $O(n^3)$ porque, en el peor caso, el árbol de prefijos de Moore tendrá tantos nodos como símbolos haya en la muestra, así que los ciclos de las líneas 5 y 6 del Algoritmo 19 tienen en el peor caso un costo de $O(n^2)$ y el llamado a *mezcladet* tiene un costo de $O(n)$ como se puede apreciar en el Algoritmo 13.

Ejemplo 54 *A continuación se presenta un ejemplo del funcionamiento del algoritmo* RPNI*: Sea* $D = (D_+, D_-)$*, tal que* $D_+ = \{\varepsilon, 00, 11, 010, 10\}$ *y* $D_- = \{0, 1, 001\}$*. El árbol de prefijos de Moore se muestra en la Figura 4.1 (a). La ejecución comienza comparando los estados 1 y 2 que no pueden mezclarse, ya que* $\Phi(1) = 1 \wedge \Phi(2) = 0$*, algo similar ocurre al comparar los estados 1 y 3. La mezcla de los estados 2 y 3 implica mezclar también los estados 4 y 6 y los estados 5 y 7, que sí puede llevarse a cabo. Notar que otro efecto de la mezcla es el cambio de* $\Phi(5)$ *que de ahora en adelante vale 1. Después de realizar la mezcla de los estados 2 y 3, el árbol de prefijos de Moore se a transformado en el autómata que se ilustra en la Figura 4.1 (b). Posteriormente, la mezcla de los estados 1 y 4 modifica nuevamente el autómata, dejándolo en el estado que se aprecia en la Figura 4.1 (c). El estado 5 no se puede mezclar con ninguno de los anteriores y finalmente, al mezclar los estados 1 y 9 se obtiene el autómata final, que se puede apreciar en la Figura 4.1 (d).*

4.1.2. Algoritmos de *Gold* y *DeLeTe2*

El clásico algoritmo de *Gold* converge en el límite al autómata determinista mínimo que reconoce el lenguaje objetivo L [Gol67]. De otro lado, el algoritmo *DeLeTe2* que se describe en [DLT04], se utiliza para inferir lenguajes regulares descritos por autómatas no deterministas. Esta sección describen brevemente los dos algoritmos y la relación que existe entre ellos.

El algoritmo propuesto por Gold [Gol78, TB73] se presenta en el Algoritmo 20. Él consta de tres partes: en la primera (líneas 3-6) se escogen los estados S de la hipótesis, es decir, el subconjunto de estados de M que son distinguibles de todos los otros estados en $Pr(D_+ \cup D_-)$. La segunda parte construye una nueva máquina de Moore M' con los estados de S, y con las transiciones calculadas en

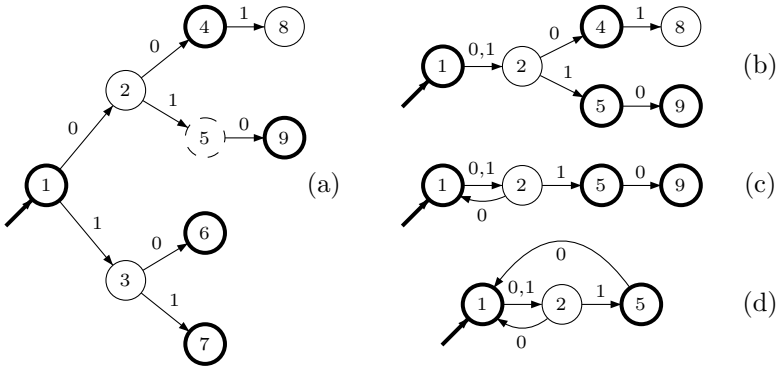


Figura 4.1: Árbol de Prefijos de Moore y Autómatas Intermedios, Producidos al Ejecutar el Algoritmo *RPNI* sobre la Entrada $D_+ = \{\varepsilon, 00, 11, 010, 10\}$ y $D_- = \{0, 1, 001\}$.

las líneas 10-16; las nuevas transiciones reflejan en la hipótesis el efecto de los estados no obviamente diferentes, que se han agrupado y representado por uno de ellos en S. La tercera parte retorna la respuesta de acuerdo con la consistencia de la nueva máquina M'. Si M' es consistente con la muestra de entrada, se retorna como respuesta, si no se retorna el árbol de prefijos de Moore inicial. Una descripción más profunda de este algoritmo se puede encontrar en [GCR00].

El algoritmo *DeLeTe2* (Algoritmo 21) produce como salida un autómata residual (RFSA), más exactamente, el autor afirma que el algoritmo converge al subautómata saturado del DFA mínimo A_{u_p} que según Denis [DLT04] se obtiene al saturar el autómata A y reducir todos los estados mayores que p, siendo p el estado primo más grande de A. El algoritmo busca relaciones de inclusión entre lenguajes residuales y refleja esas situaciones utilizando un operador de saturación. Como podría esperarse, este método se hace más interesante cuando el autómata objetivo contiene muchos lenguajes residuales compuestos, porque en ese caso pueden existir muchas relaciones de inclusión entre estados que hacen disminuir el tamaño de la hipótesis. En caso contrario, si la mayoría de los lenguajes residuales son primos la hipótesis de inferencia tendría un tamaño similar al tamaño del DFA mínimo [DLT04]. El algoritmo *DeLeTe2* infiere comúnmente autómatas que no son consistentes con las muestras de entrada, los autores resolvieron este problema con el programa $DeLeTe2$, desafortunadamente las propiedades teóricas del algoritmo que implementaron en dicho programa no se conocen y las experimentaciones prácticas no dan garantía de que la convergencia al autómata A_{u_p} se mantenga. Cómo mantiene la consistencia el programa se explica en la Sección 4.1.3 donde se presenta con detalle el procedimiento que utiliza el programa, por ahora baste con adelantar que las hipótesis se construyen en forma incremental y que si se adiciona una relación de inclusión que cause inconsistencia con las muestras, el hecho se detecta inmediatamente y se deshace la relación que la causó.

Algoritmo 20 *Gold*(D_+, D_-)

1: $M := APM(D_+, D_-)$ // $M = (Pr(D_+, D_-), \Sigma, \delta_0, q_0, \Phi)$
2: $S := \{\varepsilon\}$
3: **while** $\exists s' \in S\Sigma - S$ tal que $\forall s \in S$, $s \not\simeq s'$ **do**
4: Escoger s' // El primero en orden lexicográfico
5: $S := S \cup \{s'\}$
6: **end while**
7: $Q := S$
8: $q_0 := \{\varepsilon\}$
9: **for** $s \in S$ **do**
10: **for** $a \in \Sigma$ **do**
11: **if** $sa \in S$ **then**
12: $\delta(s, a) := sa$
13: **else**
14: $\delta(s, a) :=$ primer $s' \in S$ tal que $sa \simeq s'$ // En orden lexicográfico
15: **end if**
16: **end for**
17: **end for**
18: $M' := (Q, \Sigma, \delta, \varepsilon, \Phi)$
19: **if** M' es consistente con (D_+, D_-) **then**
20: Return M'
21: **else**
22: Return M
23: **end if**

Algoritmo *NGold1*

Con las siguientes modificaciones al algoritmo de *Gold*, se puede obtener un algoritmo que aunque hereda del *DeLeTe2* original el hecho que no siempre obtiene hipótesis consistentes con las muestras, converge al RFSA saturado del DFA mínimo del lenguaje objetivo.

Se debe reemplazar la línea 8 del Algoritmo 20 por: $I = \{u \in S : u \prec \varepsilon\}$.

El ciclo que comienza en la línea 10 debe modificarse por:

for $a \in \Sigma$ **do**
$\delta(s, a) = \{s' \in S : s' \prec sa\}$
end for

Sea *NGold1* este nuevo algoritmo. Notar que los cambios propuestos no afectan sustancialmente la complejidad temporal con respecto al algoritmo original, ya que revisar si se cumple la relación de inclusión $\prec$ es equivalente a revisar si se cumple la relación de no distinguibilidad $\simeq$ y así, la única diferencia es el mayor número de transiciones que puede llegar a tener el RFSA con respecto al DFA con el mismo número de estados.

Lema 55 *El Algoritmo* NGold1 *converge al RFSA saturado del DFA mínimo que reconoce el lenguaje objetivo.*

Algoritmo 21 *DeLeTe2*(D_+, D_-)

1: Sea $Pref$ el conjunto de prefijos de D_+ en orden lexicográfico
2: $Q := \emptyset$; $I := \emptyset$; $F := \emptyset$; $\delta := \emptyset$; $u := \varepsilon$
3: parar := falso
4: **while** $\neg$ parar **do**
5: **if** $\exists u' | u \simeq u'$ **then**
6: Eliminar $u\Sigma^*$ de $Pref$
7: **else**
8: $Q := Q \cup \{u\}$
9: **if** $u \prec u'$ **then**
10: $I := I \cup \{u\}$
11: **end if**
12: **if** $u \in D_+$ **then**
13: $F := F \cup \{u\}$
14: **end if**
15: $\delta := \delta \cup \{(u', x, u) | u' \in Q, u'x \in Pref, u \prec u'x\} \cup \{(u, x, u') | u' \in Q, ux \in Pref, u' \prec ux\}$
16: **end if**
17: **if** u es la ultima palabra de $Pref$ o $A = (Q, \Sigma, \delta, I, F)$ es consistente con D_+, D_- **then**
18: parar := verdadero
19: **else**
20: $u :=$ siguiente palabra en $Pref$
21: **end if**
22: **end while**
23: Return $A = (Q, \Sigma, \delta, I, F)$

Demostración. *NGold1* funciona en la misma forma que el algoritmo clásico de Gold mientras construye el conjunto S, así que el distingue los estados del autómata objetivo. De otra parte, si $u_1 \prec u_2$ y $u_1 \not\simeq u_2$, existirá una palabra v tal que $u_1 v \in D_- \wedge u_2 v \in D_+$ de modo que también pueden establecerse las relaciones "$\prec$.$^{\text{en}}$tre estados. ■

Algoritmo *NGold2*

El proceso de construir los estados del autómata de salida en *NGold1* continúa hasta que no hay estados en $S\Sigma - S$ que sean obviamente diferentes de todos los estados en S, entonces se emite la hipótesis. Los siguientes cambios en *NGold1* producen un algoritmo que se comporta exactamente como el algoritmo *DeLeTe2* propuesto por Denis en [DLT04]:

Cada vez que se obtiene una nueva versión del conjunto S en el algoritmo *NGold1*:

- Se emite una nueva hipótesis.
- Se revisa si ella es consistente con las muestras de entrada. Si no es consistente y $S\Sigma - S$ no está vacío, se agrega un estado a S. Si es consistente ó $S\Sigma - S$ está vacío, el algoritmo termina.

Este algoritmo es llamado *NGold2*.

Lema 56 *Los algoritmos* NGold2 *y* DeLeTe2 *[DLT04] son equivalentes.*

Demostración. La relación $\simeq$ definida en [DLT04] entre estados es equivalente a la relación conocida en la terminología de Gold como estados *no obviamente diferentes*. El proceso seguido en ambos algoritmos para adicionar nuevos estados al autómata también es equivalente. ■

4.1.3. Programa *DeLeTe2*

El programa *DeLeTe2*, se presenta en [DLT04] como la implementación de una mejora al algoritmo *DeLeTe2* que ha sido explicado en la Sección 4.1.2. Ambos son métodos de inferencia gramatical para lenguajes regulares, cuyas hipótesis de inferencia se expresan mediante RFSAs. El programa *DeLeTe2* pretende mejorar la falta de consistencia de las hipótesis generadas por el algoritmo *DeLete2* con respecto a las muestras de entrenamiento. El programa *DeLeTe2* está disponible de forma libre en Internet[1]; sin embargo, no se conoce ninguna publicación acerca del algoritmo que implementa ni sobre sus propiedades teóricas. Dado lo anterior, se ha procedido a estudiar con detenimiento el código fuente del mencionado programa, con el fin de entender el procedimiento seguido y el resultado de ese estudio es el que a continuación se presenta.

Toda referencia posterior en este documento a *DeLeTe2* se refiere al programa y al algoritmo inferido de dicho programa, no al algoritmo descrito en la Sección 4.1.2 ya que, como se ha indiciado, ese algoritmo tiene problemas de

[1] `http://www.grappa.univ-lille3.fr/~lemay/`, seleccionar *Recherche* y luego *Annexes*

consistencia y además los resultados experimentales exitosos se han obtenido con el programa.

Al comparar el algoritmo de la Sección 4.1.2 con el programa que se describe en esta sección, se observa que aunque teóricamente el programa *DeLeTe2* es una evolución del algoritmo *DeLeTe2*, no se encuentran rastros del segundo en el primero. Las características más importantes del método implementado en el programa *DeLeTe2*, según lo que se puede desprender del código fuente que se conoce son:

- Es un algoritmo de inferencia gramatical de lenguajes regulares que funciona con base en la mezcla de estados.
- El algoritmo parte del árbol de prefijos de Moore de las muestras de entrenamiento e infiere RFSAs. En el proceso aprovecha de diversas formas las propiedades de los lenguajes residuales asociados a los estados; en particular, aprovecha la propiedad transitiva de la relación de inclusión entre lenguajes.
- El algoritmo propone hipótesis de inferencia representadas mediante RFSAs que siempre son consistentes con las muestras de entrenamiento, mejorando así el inconveniente de su predecesor.
- Los resultados experimentales muestran una tasa de desempeño muy alta en lenguajes regulares objetivo que se representan como expresiones regulares o como NFAs; sin embargo, para lenguajes objetivo representados como DFAs generados aleatoriamente, sus resultados son pobres. Esta diferencia obedece a la escasez de estados compuestos en los lenguajes objetivo procedentes de DFAs, que conlleva a que existan pocas relaciones de inclusión entre los lenguajes residuales asociados a los estados.
- Los tamaños de las hipótesis obtenidas por el algoritmo son más grandes de lo esperado, si se tiene en cuenta que el RFSA canónico puede tener un tamaño considerablemente menor que el DFA mínimo del lenguaje objetivo.
- Experimentalmente se ha observado que el tamaño de las hipótesis converge al tamaño del DFA mínimo del lenguaje objetivo. Esto se contradice con la expectativa de producir hipótesis más pequeñas al inferir RFSAs saturados.

La estrategia de funcionamiento de *DeLeTe2* consiste en considerar cada posible par de estados del árbol de prefijos de Moore en orden lexicográfico. Para cada pareja se explora la posible relación entre ellos: *Inclusión*, *No inclusión* o *Desconocida*. Como ya se ha dicho, estas relaciones se refieren en realidad a la relación que exista entre los lenguajes residuales asociados a cada estado. En caso que la relación entre dos estados sea *Desconocida*, se supone por un momento que esa relación realmente es de *Inclusión* y se propaga esa afirmación con todas sus consecuencias; si en algún momento de la propagación se llega a una inconsistencia del tipo ($x \prec y$ y $x \not\prec y$) se deshace la afirmación inicial y todos los efectos de su propagación. Las implicaciones de afirmar una relación de *Inclusión* entre estados pueden ser diversas: puede permitir la definición del

valor de salida a un estado cuyo valor de salida era desconocido hasta el momento; puede causar el descubrimiento de nuevas relaciones de *Inclusión* y *No inclusión* entre parejas de estados cuya relación era desconocida hasta el momento; puede implicar la fusión de estados en caso que se pruebe la inclusión en ambos sentidos para una pareja de estados dada; puede modificar el conjunto de sucesores de un estado, ya que cuando se está calculando una hipótesis y se detecta un estado que pertenece a la hipótesis que no tiene sucesores ó que sus sucesores están fuera de la hipótesis, se consideran como sus sucesores todos los estados de la hipótesis que están incluídos en él.

El algoritmo 22 muestra el *DeLeTe2*. Cada estado almacena sus relaciones de inclusión y no inclusión con respecto a todos los demás estados del autómata en la matriz R; esta información se inicializa marcando las relaciones de no inclusión obvias en el autómata inicial: si $\Phi(x) = 1$ y $\Phi(y) = 0$ entonces $x \not\prec y$ (línea 2), es decir, $R[x][y] = N$. La propagación de la relación desconocida que se supone relación de inclusión (líneas 10-11 y 13-14) es la parte más costosa del algoritmo y se realiza en la subrutina *buscar_inclusion*. Cuando se ha comparado un estado $q1$ con todos los estados $q2$ y no ha sido posible mezclarlo con ningún otro, el estado $q1$ entra a formar parte de la hipótesis actual (línea 21) y se procede a evaluar si ahora la hipótesis es consistente con la muestra positiva (líneas 23-26), si lo es, la hipótesis es la respuesta que se busca y el algoritmo termina, en caso contrario se continúa el procesamiento.

La subrutina *buscar_inclusion* que se muestra en el Algoritmo 23 transforma M cuando realiza agrupamientos de estados, pero sobre todo modifica el registro de relaciones de inclusión R. Esta subrutina utiliza dos pilas para ir acumulando las parejas de estados hacia las cuales se debe propagar la relación de inclusión o de no inclusión; el procesamiento comienza con la pareja de estados que se recibe como parámetro, ella se inserta en la pila de relaciones de inclusión que falta probar, al entrar en el ciclo principal la pareja que está en el tope se desapila y se evalúan las relaciones conocidas entre la pareja de estados y cada uno de los demás estados de la máquina. Los casos que se evalúan depende de lo que se quiera verificar: una relación de inclusión o de no inclusión, los Cuadros 4.1 y 4.2 resumen dichos casos y las subrutinas *evaluarCasosNoInclusion* (ver Algoritmo 24) y *evaluarCasosInclusion* (ver Algoritmo 25) muestran con más detalle cómo se aplican; en particular observar que ellas modifican la matriz R y que apilan nuevos elementos en $pila_n$ y $pila_i$. Al verificar si alguno de estos casos se cumple, se puede detectar una inconsistencia, si es así, se ignora todo lo actuado y se determina que la supuesta relación de inclusión no es posible. Si no se detecta ninguna inconsistencia, la relación se mantiene así como toda la información a que haya dado lugar. Una vez se han hecho todas las comparaciones se pasa a propagar la relación de inclusión, esto se hace hacia los hijos en caso de una relación de inclusión y hacia los padres en una relación de no inclusión, las nuevas parejas de estados para las que se debe verificar si cumplen la relación se adicionan a las pilas correspondientes. Notar que se habla durante el proceso de relaciones de inclusión y no inclusión, esto se debe a que a partir de una relación de inclusión se pueden deducir relaciones de no inclusión (ver Cuadro 4.2) entre otros estados y todas estas relaciones que se descubren deben propagarse hasta sus últimas consecuencias. Las subrutinas *propagarRelacionNoInclusion* (ver Algoritmo 26) y *propagarRelacionInclusion* (ver Algoritmo 27) muestran

en detalle cómo se realiza la propagación de las relaciones, observar que durante la propagación también es posible encontrar inconsistencias, en este caso se procede del mismo modo, deshaciendo todo lo actuado. Si se logra propagar todos los efectos de la nueva relación sin encontrar ninguna inconsistencia, se verifica si la nueva información obtenida permite el agrupamiento de algún par de estados, si es el caso, se realiza la fusión por medio de la subrutina *fusionar* que realiza una mezcla sin propagación.

Algoritmo 22 $DeLeTe2(M, D_+, D_-)$

Require: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$ M es el árbol de Prefijos de Moore de $D_+ \cup D_-$

1: Sea R una matriz de tamaño $|Q| \times |Q|$ donde $R[i][j] = I$ si $i \prec j$, $R[i][j] = N$ si $i \not\prec j$ y está vacía si no se conoce la relación.
2: estados = Q // En recorrido por anchura
3: Inicializar relaciones de no inclusión
4: $S = \{\}$
5: $M' = (\{\}, \Sigma, \{0, 1, ?\}, \{\}, \{\})$ // Autómata vacío
6: **for all** $q1 \in estados$ // En recorrido por anchura **do**
7: **if** $q1$ está activo **then**
8: **for all** $q2 \in estados, q2 \ll q1$ // en recorrido por anchura **do**
9: **if** $q2$ está activo **then**
10: **if** $q1 \neq q2$ **then**
11: **if** $obtener\_relacion(R, q1, q2) = Desconocida$ **then**
12: $buscar\_inclusion(M, R, q1, q2)$
13: **end if**
14: **if** $obtener\_relacion(R, q2, q1) = Desconocida$ **then**
15: $buscar\_inclusion(M, R, q2, q1)$
16: **end if**
17: **end if**
18: **end if**
19: **end for**
20: **if** q_1 está activo **then**
21: $S = S \cup \{q_1\}$
22: **end if**
23: M' = Autómata inducido por S en M
24: **if** $consistenciaPositiva(M', D_+)$ **then**
25: Return M'
26: **end if**
27: **end if**
28: **end for**
29: Return M'

Los Cuadros 4.1 y 4.2 representan situaciones en las que la información disponible permite deducir nuevas relaciones de inclusión al comparar los lenguajes residuales asociados a los estados $q1, q2$ y $q3$. En el Cuadro 4.1 se presentan los casos en los que se pueden detectar nuevas relaciones de *No inclusión* y en el Cuadro 4.2 los casos en los cuales se pueden detectar nuevas relaciones de *in-*

Algoritmo 23 *buscar_inclusion*(M, R, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$
Require: R es una matriz de tamaño $|Q| \times |Q|$ donde $R[i][j] = I$ si $i \prec j$, $R[i][j] = N$ si $i \not\prec j$ y está vacía si no se conoce la relación.
Require: $p, q \in Q$ y se supone $p \prec q$

1: $pila_i := []$ // pila que acumula relaciones de inclusión por validar
2: $pila_n := []$ // pila que acumula relaciones de no inclusión por validar
3: $M' := M$
4: $R' := R$
5: $push(pila_i, (p, q)$
6: **while** $\neg empty(pila_i) \wedge \neg empty(pila_n)$ **do**
7: **while** $\neg empty(pila_n)$ **do**
8: $(e1, e2) := pop(pila_n)$
9: **if** $\neg evaluarCasosNoInclusion(R, e1, e2, pila_n)$ **then**
10: $R := R'$; $M := M'$
11: $R[p][q] = N$
12: Return
13: **end if**
14: **if** $\neg propagarRelacionNoInclusion(M, e1, e2, pila_n)$ **then**
15: $R := R'$; $M := M'$; $R[p][q] = N$
16: Return
17: **end if**
18: **end while**
19: **if** $\neg empty(pila_i)$ **then**
20: $(e1, e2) := pop(pila_i)$
21: **if** $\neg evaluarCasosInclusion(R, e1, e2, pila_i, pila_n)$ **then**
22: $R := R'$; $M := M'$
23: $R[p][q] = N$
24: Return
25: **end if**
26: **if** $\neg propagarRelacionInclusion(M, e1, e2, pila_i, pila_n)$ **then**
27: $R := R'$; $M := M'$; $R[p][q] = N$
28: Return
29: **end if**
30: **end if**
31: **end while**
32: **for** $p, q \in Q \times Q$ **do**
33: **if** $R[p][q] = I \wedge R[q][p] = I$ **then**
34: $fusionar(M, p, q)$
35: **end if**
36: **end for**

Algoritmo 24 *evaluarCasosNoInclusion*$(R, e1, e2, pila_n)$

Require: Se supone $e1 \not\prec e2$

for $e3 \in Q$ **do**
  if $e3 \neq e2 \wedge e3 \neq e1 \wedge e3$ no ha desaparecido en una mezcla anterior **then**
    if $R[e1][e3] = I \wedge R[e3][e2] = I$ **then**
      Return $falso$
    end if
    if $R[e1][e3] = I \wedge R[e3][e2] \neq N$ **then**
      $push(pila_n, (e3, e2))$
      $R[e3][e2] := N$
    end if
    if $R[e3][e2] = I \wedge R[e1][e3] \neq N$ **then**
      $push(pila_n, (e1, e3))$
      $R[e1][e3] := N$
    end if
  end if
end for
Return $verdadero$

Algoritmo 25 *evaluarCasosInclusion*$(R, e1, e2, pila_i, pila_n)$

Require: Se supone $e1 \prec e2$

for $e3 \in Q$ **do**
  if $e3 \neq e2 \wedge e3 \neq e1 \wedge e3$ no ha desaparecido en una mezcla anterior **then**
    if $(R[e2][e3] = I \wedge R[e1][e3] = N) \vee (R[e3][e1] = I \wedge R[e3][e2] = N)$ **then**
      Return $falso$
    end if
    if $R[e2][e3] = I \wedge R[e1][e3] \neq I$ **then**
      $push(pila_i, (e1, e3))$
      $R[e1][e3] := I$
    end if
    if $R[e3][e1] = I \wedge R[e3][e2] \neq I$ **then**
      $push(pila_i, (e3, e2))$
      $R[e3][e2] := I$
    end if
    if $R[e1][e3] = N \wedge R[e2][e3] \neq N$ **then**
      $push(pila_n, (e2, e3))$
      $R[e2][e3] := N$
    end if
    if $R[e3][e2] = N \wedge R[e3][e1] \neq N$ **then**
      $push(pila_n, (e3, e1))$
      $R[e3][e1] := N$
    end if
  end if
end for
Return $verdadero$

Algoritmo 26 *propagarRelacionNoInclusion*$(M, e1, e2, pila_n)$

Require: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$

for $s \in \Sigma$ **do**
  if $\exists p1, p2$ tal que $\delta(p1, s) = e1 \wedge \delta(p2, s) = e2$ **then**
    for $\forall p1$ tal que $\delta(p1, s) = e1$ **do**
      for $\forall p2$ tal que $\delta(p2, s) = e2$ **do**
        if $R[p1][p2] = I$ **then**
          Return $false$
        end if
        if $R[p1][p2] \neq N$ **then**
          $R[p1][p2] := N$
          $push(pila_n, (p1, p2))$
        end if
      end for
    end for
  end if
end for
Return *verdadero*

Algoritmo 27 *propagarRelacionInclusion*$(M, e1, e2, pila_i, pila_n)$

Require: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, q_0, \Phi)$

for $s \in \Sigma$ **do**
  if $\exists p1, p2$ tal que $\delta(e1, s) = p1 \wedge \delta(e2, s) = p2$ **then**
    for $\forall p1$ tal que $\delta(e1, s) = p1$ **do**
      for $\forall p2$ tal que $\delta(e2, s) = p2$ **do**
        if $R[p1][p2] = N$ **then**
          Return $false$
        end if
        if $R[p1][p2] \neq I$ **then**
          $R[p1][p2] := I$
          $push(pila_i, (p1, p2))$
        end if
      end for
    end for
  end if
end for
Return *verdadero*

clusión. Todas las deducciones aplican la propiedad transitiva de la inclusión.

Cuadro 4.1: Casos de Transitividad de las Relaciones de *No Inclusión* Suponiendo que $q1 \not\prec q2$

Si	Entonces
$q1 \prec q3$ y $q3 \prec q2$	$q1 \prec q2$ y Deshacer
$q1 \prec q3$ y $\neg(q3 \not\prec q2)$	$q3 \not\prec q2$
$q3 \prec q2$ y $\neg(q1 \prec q3)$	$q1 \not\prec q3$

Cuadro 4.2: Casos de Transitividad de las Relaciones de *Inclusión* Suponiendo que $q1 \prec q2$

Si	Entonces
$(q2 \prec q3$ y $q1 \not\prec q3)$	$q1 \not\prec q2$ y Deshacer
$(q3 \prec q1$ y $q3 \not\prec q2)$	$q1 \not\prec q2$ y Deshacer
$q2 \prec q3$ y $\neg(q1 \prec q3)$	$q1 \prec q3$
$q3 \prec q1$ y $\neg(q3 \prec q2)$	$q3 \prec q2$
$q1 \not\prec q3$ y $\neg(q2 \not\prec q3)$	$q2 \not\prec q3$
$q3 \not\prec q2$ y $\neg(q3 \not\prec q1)$	$q3 \not\prec q1$

La complejidad temporal del algoritmo *DeLeTe2* se puede calcular en función del tamaño de la muestra de entrada n, ya que en el peor caso, el árbol de prefijos de Moore tendrá tantos estados como símbolos haya en la muestra. Una cota superior de la complejidad en tiempo de la subrutina *buscar_inclusion* es $O(n^2)$, ya que en el peor caso itera sobre tantas parejas de estados como sucesores y predecesores tengan los estados iniciales de la comparación, y para cada pareja se itera sobre todos los estados del autómata. Considerando que los dos ciclos externos al llamado de la subrutina, tienen cada uno complejidad peor caso $O(n)$, una cota superior de la complejidad temporal del algoritmo *DeLeTe2* es $O(n^4)$.

Ejemplo 57 *A continuación se presenta un ejemplo de la ejecución de* DeLeTe2. *El alfabeto es* $\Sigma = \{0, 1\}$, *las muestras de entrada son* $D_+ = \{\varepsilon, 00, 010, 10, 11\}$ *y* $D_- = \{0, 001, 1\}$. *La Figura 4.2 muestra el árbol aceptor de prefijos que se obtiene con dichas muestras, el cual es el punto de partida del algoritmo.*

La inicialización de relaciones de no inclusión (línea 3 del Algoritmo 22) consiste en marcar como no inclusión la relación entre un estado y sí mismo y entre un estado de aceptación y uno de rechazo, es decir que dada la pareja de estados (p, q) *se tiene que si* $\Phi(p) = 1 \wedge \Phi(q) = 0 \Rightarrow p \not\prec q$ *(notar que no se puede intercambiar la posición de los estados en la pareja). Se pueden afirmar estas relaciones de no inclusión ya que no es posible que el lenguaje asociado con el primer estado* (p), *esté incluido en le lenguaje asociado al segundo* (q). *Para el ejemplo, la tabla de relaciones de inclusión quedaría inicializada como*

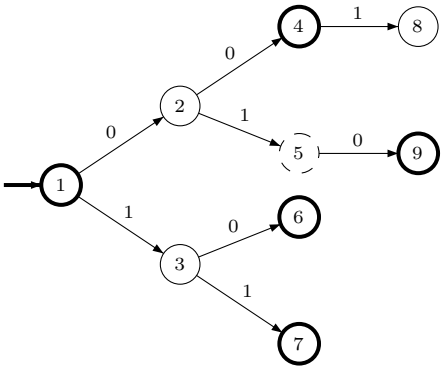


Figura 4.2: Ejemplo *DeLeTe2*, Árbol Aceptor de Prefijos de la Muestra de Entrada $D_+ \cup D_-$

se muestra en el Cuadro 4.3. En esta tabla y las siguientes, la posición $[i, j]$ *representa la relación entre el estado* i *y el estado* j *y los valores posibles para esta relación son: I si es una relación de inclusión, N si la relación es de no inclusión ó espacio en blanco si la relación es desconocida.*

Cuadro 4.3: Tabla de Relaciones de Inclusión Después de su Inicialización

	1	2	3	4	5	6	7	8	9
1	N	N	N					N	
2	N	N							
3	N		N	N					
4		N	N	N				N	
5	N				N				
6		N	N			N		N	
7		N	N				N	N	
8								N	
9		N	N					N	N

A continuación el algoritmo empieza a formar parejas de estados en el orden de un recorrido por anchura del árbol de prefijos de Moore (líneas 6 y 8 del Algoritmo 22).El estado 1 *no tiene estados menores que él en el orden lexicográfico, por lo tanto el ciclo interno no llega a iterar, el estado* 2 *se compara con el estado* 1*, pero la relación entre estos estados ya es conocida, por lo tanto no produce ninguna modificación en el autómata, igual ocurre al considerar los estados* 3 *y* 1*. Al considerar la pareja de estados* 3 *y* 2 *cuya relación es desconocida, se supone que dicha relación es de inclusión y se procede a plasmarla en el autómata (línea 12); ello implica explorar la relación entre* 3 *y cada uno de los demás estados del autómata, así como las relaciones entre* 2 *y todos los demás estados del mismo y descubrir nuevas relaciones a través de la propiedad transitiva de la inclusión (para esto se aplican los casos explicados en los Cuadros 4.1 y 4.2). En el caso particular, se puede ver que si* $3 \prec 2$ *necesariamente*

debe cumplirse que $2 \not\prec 4$. *Al propagar la relación hacia los hijos, se tiene que* $6 \prec 4$ *y* $7 \prec 5$*; de* $6 \prec 4$ *se desprende, gracias a la transitividad, que* $2 \not\prec 6$, $5 \not\prec 2$, $5 \not\prec 3$, $5 \not\prec 8$ *y* $3 \not\prec 6$. *Estas nuevas relaciones se marcan en la tabla de inclusiones. Adicionalmente, al relacionar* 7 *con* 5 *se deduce que* $\Phi(5) = 1$ *ya que de otro modo se causaría una inconsistencia. Continuando la ejecución (líneas 14,15,16) se prueba la relación contraria, es decir* $2 \prec 3$, *la cual implica* $4 \prec 6$, $5 \prec 7$ *y* $7 \not\prec 1$. *Al finalizar la búsqueda de nuevas inclusiones, se barre el autómata buscando estados equivalentes y se detecta que* $2 \backsimeq 3$, $4 \backsimeq 6$ *y* $5 \backsimeq 7$. *Dado que se logró llevar ambas suposiciones hasta sus últimas consecuencias sin causar inconsistencia lo actuado se mantiene, el nuevo estado de la tabla de relaciones de inclusión se muestra en el Cuadro 4.4 y el nuevo estado del autómata, que se modifica a causa de la mezcla de los estados equivalentes, se muestra en la Figura 4.3.*

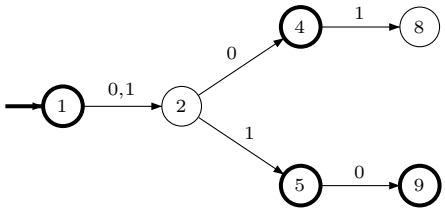


Figura 4.3: Ejemplo *DeLeTe2*, Autómata luego de Mezclar los Estados Equivalentes $2 \backsimeq 3$, $4 \backsimeq 6$ y $5 \backsimeq 7$

Cuadro 4.4: Tabla de Relaciones de Inclusión Después de Mezclar $2 \backsimeq 3$, $4 \backsimeq 6$ y $5 \backsimeq 7$

	1	2	3	4	5	6	7	8	9
1	N	N	N					N	
2	N	N	I	N		N			
3	N	I	N	N		N			
4		N	N	N		I		N	
5	N	N	N		N		I	N	
6		N	N	I		N		N	
7	N	N	N		I		N	N	
8								N	
9		N	N					N	N

Ahora el estado 4 *empieza a compararse con los estados aun existentes que son menores que él en orden lexicográfico:* 1 *y* 2. *En particular las relaciones* $4 \prec 1$ *y* $1 \prec 4$ *son desconocidas y desencadenan la búsqueda de nuevas relaciones de inclusión como* $5 \not\prec 4$, $8 \prec 2$ *en el primer caso y* $2 \prec 8$, $8 \not\prec 1$ *y* $8 \not\prec 4$ *en el segundo. Posteriormente se detecta que* 1 *y* 4 *pueden mezclarse, así como* 2 *y* 8. *El Cuadro 4.5 muestra el nuevo estado de la tabla de relaciones de inclusión y la Figura 4.4 el autómata después de realizadas las mezclas.*

Posteriormente se encuentra que las relaciones entre los estados 1 *con* 5 *y* 2

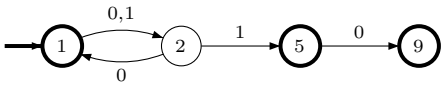


Figura 4.4: Ejemplo *DeLeTe2*, Autómata luego de Mezclar los Estados Equivalentes $1 \backsimeq 4$ y $2 \backsimeq 8$

Cuadro 4.5: Tabla de Relaciones de Inclusión Después de Relacionar 1 con 4

	1	2	3	4	5	6	7	8	9
1	N	N	N	I				N	
2	N	N	I	N		N		I	
3	N	I	N	N		N			
4	I	N	N	N		I		N	
5	N	N	N	N	N		I	N	
6		N	N	I		N		N	
7	N	N	N		I		N	N	
8	N	I		N				N	
9		N	N					N	N

con 5 *son desconocidas, lo cual conlleva el detectar nuevas relaciones de inclusión y no inclusión (ver Cuadro 4.6). Sin embargo, en este caso no se detectan equivalencias entre estados, por lo que el autómata no varía.*

Cuadro 4.6: Tabla de Relaciones de Inclusión Después de Relacionar $2 \prec 5$

	1	2	3	4	5	6	7	8	9
1	N	N	N	I	I			N	I
2	N	N	I	N	I	N		I	I
3	N	I	N	N		N			
4	I	N	N	N		I		N	
5	N	N	N	N	N		I	N	
6		N	N	I		N		N	
7	N	N	N		I		N	N	
8	N	I		N				N	
9	N	N	N					N	N

Cada vez que se alcanza la línea 24 del Algoritmo, se evalúa la consistencia del subautómata inducido por los estados del conjunto S (línea 21 del Algoritmo) en el autómata total. En el momento que dicho subautómata sea consistente con las muestras positivas, se ha llegado a la respuesta. En este ejemplo se obtiene consistencia al terminar de considerar el estado 5*, por lo tanto, la respuesta final es el subautómata inducido por el conjunto de estados* $S = \{1, 2, 5\}$*, que se muestra en la Figura 4.5.*

Nótese cómo la construcción del autómata inducido se comporta de manera especial cuando se encuentran arcos que van a estados fuera del conjunto

S ó cuando no hay arco desde un estado x de S con un símbolo dado. En el primer caso los arcos serán redireccionados a estados que si pertenecen a S, en el segundo caso se crean arcos nuevos. En ambas circunstancias, los nuevos arcos se generan de la misma manera: se recorren los estados de S que tengan un estado hijo que pertenece a S con el símbolo de interés y que estén incluidos en el estado x. Estos hijos serán los destinatarios de los nuevos arcos. En el ejemplo se ilustran ambas condiciones. Observar en la Figura 4.4 cómo el estado 5 *está conectado con el símbolo* 0 *al estado* 9 *que no pertenece a S. En lugar de este arco, la Figura 4.5 muestra como desde el estado* 5*, con el símbolo* 0 *parten sendos arcos hacia los estados* 1 *y* 2*, esto se debe a que tanto* 1 *como* 2 *son hijos, a través del símbolo* 0*, de estados incluidos en* 5 *(*2 *y* 1 *respectivamente), la inclusión se puede verificar observando las posiciones* $[1, 5]$ *y* $[2, 5]$ *del Cuadro 4.6. El estado* 5 *también ilustra lo que ocurre cuando no hay arco con un símbolo dado, en este caso con el símbolo* 1 *(Figura 4.4), se detectan los estados de S que tienen hijos mediante el símbolo* 1 *en S y que están incluidos en* 5*, consultando el Cuadro 4.6 se observa que ellos son los estados* 1 *y* 2*, y que sus hijos con el símbolo* 1 *son respectivamente* 2 *y* 5*, así que los nuevos arcos que parten de* 5 *con el símbolo* 1 *van a* 2 *y a* 5*, como se aprecia en la Figura 4.5.*

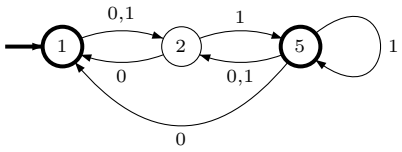


Figura 4.5: Ejemplo *DeLeTe2*, Respuesta Final

Por último, observar cómo la hipótesis obtenida por el algoritmo RPNI *en la Sección 4.1.1 y la aquí obtenida por* DeLeTe2*, ante las mismas muestras de entrada son notablemente diferentes.*

4.2. Algoritmos Propuestos que Hacen Mezcla Determinista

4.2.1. Algoritmo *IRPNI2*

La idea fundamental del algoritmo *IRPNI2* [GRCA05] es aumentar la información disponible para el proceso de mezcla, aprovechando las relaciones de inclusión que se logran establecer cuando los estados no pueden ser mezclados. Como se aprecia en el Algoritmo 28, el ciclo principal y el proceso de mezcla de estados son los mismos que en el algoritmo *RPNI*. La diferencia está cuando los estados p y q no pueden mezclarse, es decir, cuando $\texttt{mezcladet}(M, p, q) = M$ en la línea 7 del Algoritmo 28. Este algoritmo intenta definir el valor de salida mediante la subrutina *definirEstados* que está estrechamente relacionada con la subrutina *definirValoresSalida* que fue explicada con detalle en la Sección 3.1.3.

Las siguiente definiciones son útiles para entender las funciones *intentarIn-*

Algoritmo 28 *IRPNI2*(D_+, D_-)

1: $M := APM(D_+, D_-)$
2: $lista := \{u_0, u_1, \ldots, u_r\}$ //estados de M en orden lexicográfico, $u_0 = \varepsilon$
3: $lista' := \{u_1, \ldots, u_r\}$
4: $q := u_1$
5: **while** $lista' \neq \emptyset$ **do**
6: **for** $p \in lista$ and $p \ll q$ (en orden lexicográfico) **do**
7: **if** $mezcladet(M, p, q) = M$ **then**
8: $M := definirEstados(M, p, q)$
9: **else**
10: $M := mezcladet(M, p, q)$
11: exit for
12: **end if**
13: **end for**
14: $lista :=$ Eliminar de $lista$ los estados que no están en M
15: $lista' :=$ Eliminar de $lista'$ los estados que no están en M
16: $q := first(lista')$
17: **end while**
18: Return M

clusion y *definirEstados* que se muestran en los Algoritmos 29 y 30 respectivamente.

Definición 58 *Los estados p y q son no-comparables (para relaciones de inclusión) en una Máquina de Moore si existen $u, v \in \Sigma^*$ tales que $\Phi(\delta(p,u)) = 1 \wedge \Phi(\delta(q,u)) = 0$ y $\Phi(\delta(p,v)) = 0 \wedge \Phi(\delta(q,v)) = 1$.*

Definición 59 *Dados $p, q \in Q$, se dice que el estado p es potencialmente menor que el estado q si ellos no son no-comparables y no existe $u \in \Sigma^*$ tal que $\Phi(\delta(p,u)) = 1 \wedge \Phi(\delta(q,u)) = 0$.*

Ejemplo 60 *Para explicar el comportamiento de la subrutina* definirEstados *se va a usar la máquina de Moore de la Figura 4.6. Siguiendo el algoritmo se puede ver que $definirEstados(M, 1, 2) = M$, ya que de otro modo la muestra negativa* 1000010 *sería aceptada.*

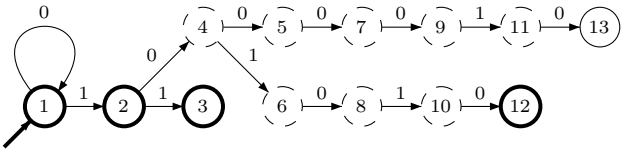


Figura 4.6: Máquina de Moore Inicial Usada para Ilustrar el Funcionamiento de *definirEstados*

En el árbol de la Figura 4.7 se aprecia que no hay parejas de estados de la forma $(+x, -y)$ ni $(-x, +y)$, por lo tanto los estados iniciales del árbol 1 *y* 2 *no*

Algoritmo 29 *intentarInclusion*(M, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}, \delta, \Phi, q_0)$
Require: $p, q \in Q$
1: $M' := M$ // $M' = (Q, \Sigma, \{0, 1, ?\}, \delta, \Phi', q_0)$
2: **while** p y q no son no-comparables **do**
3: $arbol := arbolMezcla(M', p, q)$
4: **for** $(u, v) \in arbol$ **do**
5: **if** $\Phi'(u) = 1 \wedge \Phi'(v) =?$ **then**
6: $\Phi'(v) := 1$
7: **end if**
8: **if** $\Phi'(u) =? \wedge \Phi'(v) = 0$ **then**
9: $\Phi'(u) := 0$
10: **end if**
11: **end for**
12: **end while**
13: **if** p y q son no-comparables **then**
14: Return M
15: **else**
16: Return M'
17: **end if**

Algoritmo 30 *definirEstados*(M, p, q)

if p y q son no-comparables **then**
Return M
else
if p es potencialmente menor que q **then**
M' = intentarInclusion(M,p,q)
end if
if q es potencialmente menor que p **then**
M' = intentarInclusion(M,q,p)
end if
Return M'
end if

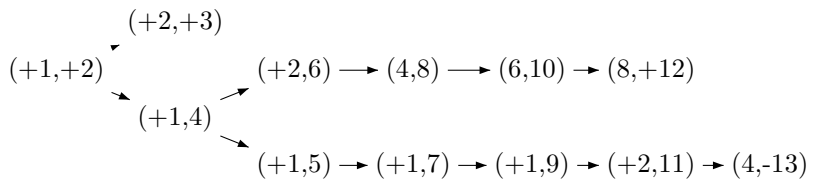


Figura 4.7: Árbol de Mezcla para los Estados 1 y 2

son no-comparables con respecto a la inclusión.

Ejecutar intentar$Inclusion(M, 1, 2)$ retorna M; de otro modo el nodo $(4, -13)$ implicaría $\Phi(4) = 0$ y al mismo tiempo el nodo $(+1, 4)$ implicaría $\Phi(4) = 1$. Por el contrario, al ejecutar intentar$Inclusion(M, 2, 1)$ no se produce ninguna inconsistencia y además cambia el valor de salida de los estados 4 y 8 y entonces la subrutina de$finirEstados(M, 2, 1)$ cambia M haciendo $\Phi(4) = 1$ y $\Phi(8) = 1$.

Lema 61 *El Algoritmo* IRPNI2 *identifica en el límite el autómata determinista (DFA) mínimo del lenguaje objetivo.*

Demostración. En el límite, no es posible adicionar nuevos valores de salida a estados, puesto que llega un punto en que todos los estados tienen definido su valor. En esas condiciones, el algoritmo *IRPNI2* se comporta de forma idéntica que *RPNI* y por lo tanto converge al DFA mínimo por la misma razón que lo hace *RPNI*.
■

Lema 62 *La complejidad temporal del algoritmo* IRPNI2 *es $O(n^3)$.*

Demostración. Las funciones *definirEstados* y *mezcladet* (Algoritmo 13) tienen un coste lineal en función del tamaño de la entrada n. Ellas son invocadas en el peor caso para cada pareja de estados posible, y puede llegar a haber $O(n^2)$ parejas. Por lo tanto, el coste total de algoritmo *IRPNI2* es $O(n^3)$. ■

Ejemplo 63 *A continuación se propone un ejemplo para ilustrar el comportamiento del algoritmo* IRPNI2 *utilizando la muestra $D_+ = \{0,\ 001,\ 000011,\ 0101010\}$ y $D_- = \{01000010\}$, es decir, la misma muestra usada para ilustrar el funcionamiento del algoritmo* IRPNI1*. El árbol de prefijos de Moore inicial se presenta en la Figura 4.8.*

Con esta muestra como entrada, $lista = \{0, 1, \ldots, 18\}$ y $lista' = \{1, 2, \ldots, 18\}$ así que IRPNI2 *mezcla en primer lugar los estados 0 y 1 y se obtiene el autómata de la Figura 4.9.*

Entonces $lista = \{0,\ 3,\ 6,\ 8,\ 9,\ 11,\ 12,\ 13,\ 14,\ 15,\ 16,\ 17,\ 18\}$ y $lista' = \{3,\ 6,\ 8,\ 9,\ 11,\ 12,\ 13,\ 14,\ 15,\ 16,\ 17,\ 18\}$ y se intenta mezclar 0 y 3 (esto elimina 3 de $lista'$) pero esos estados no pueden mezclarse, tal como se puede observar

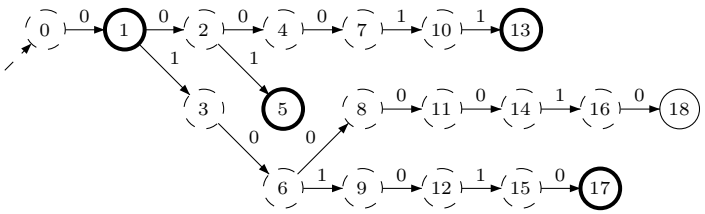


Figura 4.8: Árbol de Prefijos de Moore Inicial

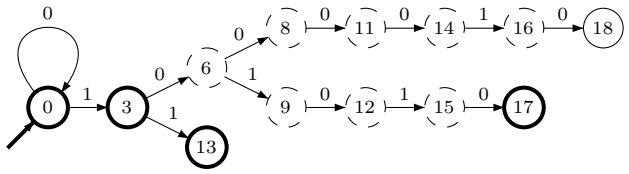


Figura 4.9: Máquina de Moore Después de Mezclar los Estados 0 y 1

en la Figura 4.10 donde las comparaciones $(+0, 6)$ *y* $(6, -18)$ *asignarían valores de salida contradictorios al estado* 6.

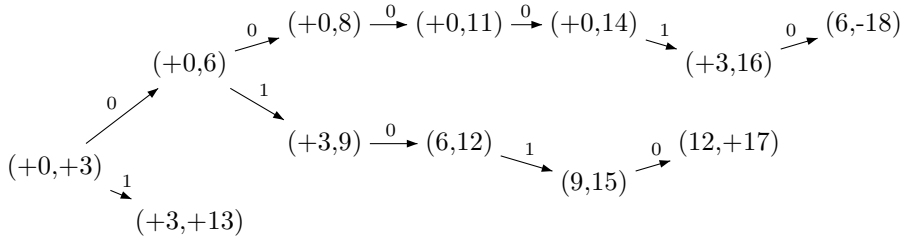


Figura 4.10: Árbol de Mezcla de los Estados 0 y 3

Posteriormente, la subrutina definirEstados *cambia los valores de salida de los estados* 6 *y* 12 *a positivo a causa de la relación de inclusión* $0 \succ 3$*, porque* $(12, +17)$ *permite inferir que* $\Phi(12) = 1$ *y* $(6, +12)$ *permite inferir* $\Phi(6) = 1$ *(ver Figura 4.10). El modelo resultante se aprecia en la Figura 4.11.*

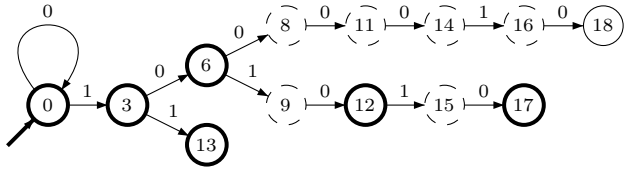


Figura 4.11: Máquina de Moore después de Comparar los Estados 0 y 3

Ahora, lista $= \{0, 3, 6, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18\}$ *y lista*$' = \{6,$

8, 9, 11, 12, 13, 14, 15, 16, 17, 18}, *las siguientes comparaciones son:* $(0, 6)$ *que son no comparables y* $(3, 6)$ *que infiere:* $\Phi(8) = 1$, $\Phi(9) = 1$, $\Phi(11) = 1$, $\Phi(14) = 1$. *El nuevo autómata es el de la Figura 4.12.*

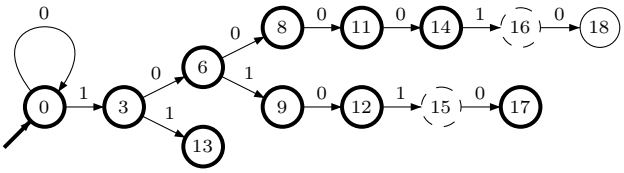


Figura 4.12: Máquina de Moore Después de Comparar los Estados 3 y 6

Ahora, $lista = \{0, 3, 6, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18\}$ *and* $lista' = \{8, 9, 11, 12, 13, 14, 15, 16, 17, 18\}$, *las siguientes comparaciones son:* $(0, 8)$ *que son no comparables y* $(3, 8)$ *que se mezclan. El nuevo autómata está en la Figura 4.13.*

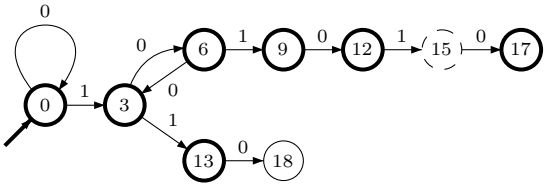


Figura 4.13: Máquina de Moore Después de Mezclar los Estados 3 y 8

En este punto, $lista = \{0, 3, 6, 9, 12, 13, 15, 17, 18\}$ *and* $lista' = \{9, 12, 13, 15, 17, 18\}$, *la siguiente comparación es* $(0, 9)$ *que es una mezcla que produce el autómata de la Figura 4.14. Aunque se realizarán otras comparaciones antes que el conjunto frontera* $lista'$ *se haga vacío, ninguna de ellas mezcla o define nuevos valores de salida, así que esta es la respuesta final.*

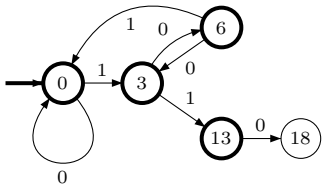


Figura 4.14: Salida Producida por el Algoritmo *IRPNI2*

4.2.2. Algoritmo *IRPNI1*

El algoritmo *IRPNI1* explora una forma de brindar información a la mezcla de estados, aprovechando las relaciones de inclusión que se puedan detectar. En este caso se realiza un primer barrido sobre todas las parejas de estados, generando en cada caso el árbol de mezcla y buscando en él indicios que apunten a una posible relación de inclusión. Una vez se han definido todos los valores

de salida posibles se ejecuta, sobre este modelo enriquecido, el algoritmo *RPNI*. El Algoritmo 31 ilustra el procedimiento, en las líneas 4-8 se generan todas las parejas de estados en orden lexicográfico y se invoca para cada una de ellas la subrutina *definirEstados*, que ha sido definida en la Sección 4.2.1 (ver Algoritmo 30) y se encarga de modificar la máquina adicionando nuevos valores de salida donde ello sea posible. Una explicación más detallada de las condiciones bajo las cuales se pueden adicionar estos nuevos valores se ha hecho en la Sección 3.1.3. Las líneas 9 en adelante reproducen el procesamiento de *RPNI* tal como fue presentado en el Algoritmo 19.

Algoritmo 31 *IRPNI1*(D_+, D_-)

```
1:  M := APM(D+, D-)
2:  lista := {u0, u1, ..., ur} //estados de M en orden lexicográfico, u0 = ε
3:  lista' := {u1, ..., ur}
4:  for q ∈ lista' do
5:     for p ∈ lista ∧ p ≪ q (en orden lexicográfico) do
6:        M := definirEstados(M, p, q)
7:     end for
8:  end for
9:  q := u1
10: while lista' ≠ ∅ do
11:    for p ∈ lista and p ≪ q (en orden lexicográfico) do
12:       if mezcladet(M, p, q) ≠ M then
13:          M := mezcladet(M, p, q)
14:          exit for
15:       end if
16:    end for
17:    lista := Eliminar de lista los estados que no están en M
18:    lista' := Eliminar de lista' los estados que no están en M
19:    q := first(lista')
20: end while
21: Return M
```

Lema 64 *El algoritmo* IRPNI1 *converge en el límite al DFA mínimo del lenguaje objetivo.*

Demostración. En caso de una presentación completa, la definición de valores de salida se hace innecesaria, ya que al estar presentes todas las cadenas del lenguaje, no quedan estados con valor de salida indefinido y el procesamiento realizado sigue la misma estrategia que *RPNI* y por lo tanto converge al DFA mínimo del lenguaje objetivo.

Cuando la presentación no es completa, el efecto de las líneas 4-8 consiste en dar valor de salida a ciertos estados que hasta el momento tienen dicho valor indefinido. Es decir, que en la línea 9 el $APM(D_+, D_-)$ puede haber sido refinado con algunos valores de salida adicionales, de la misma forma que lo habría sido si la muestra de entrenamiento constara de más palabras. En adelante, el procesamiento que se realiza es el del *RPNI*, cuya convergencia en el límite fue demostrada por Oncina y García [OG92] y que también puede demostrarse a

partir de los resultados de la Sección 3.5. Por lo tanto *IRPNI1* converge en el límite al DFA mínimo del lenguaje objetivo definido por el conjunto ampliado de muestras.

Es cierto que al ampliar el conjunto de muestras eventualmente se pueden asignar valores equivocados. Por ejemplo: si la información actual permite afirmar que dos estados guardan una relación de inclusión $x \prec y$, esto puede llevar a asignar valor de salida a un estado $\Phi(z) = 1$. Si posteriormente se conocieran más muestras del lenguaje que modificaran la relación a $x \not\prec y$ ya no podría afirmarse nada sobre el valor de salida de z y esto podría modificar el proceso de inferencia. Sin embargo, notar que en el contexto de la identificación en el límite estas inexactitudes se van corrigiendo a medida que se tiene mayor información disponible y ellas no impiden la convergencia. ■

Lema 65 *El coste temporal del algoritmo* IRPNI1 *está acotado superiormente por* $O(n^3)$.

Demostración. La subrutina *definirEstados* tiene un coste $O(n)$ y se invoca para cada pareja de estados, es decir $O(n^2)$ veces, por lo tanto las líneas 4-8 tienen un coste $O(n^3)$, al igual que las instrucciones de la línea 9 en adelante. Por lo tanto, el coste total del algoritmo *IRPNI1* es $O(n^3)$. ■

Ejemplo 66 *A continuación se presenta un ejemplo de la ejecución del algoritmo* IRPNI1. *Sea* $D_+ = \{0, 001, 000011, 0101010\}$ *y* $D_- = \{01000010\}$. *El árbol de prefijos de Moore inicial se presenta en la Figura 4.15. Cuando la ejecución ha llegado a la línea 9 del Algoritmo 31, después de haber definido todos los valores de salida posibles, la máquina de Moore es la que se aprecia en la Figura 4.16. Al finalizar el procesamiento, la respuesta que se obtiene es la que se muestra en la Figura 4.17.*

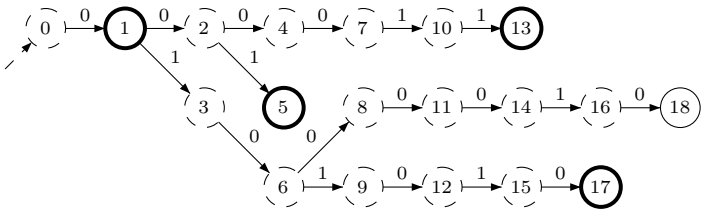


Figura 4.15: Árbol de Prefijos de Moore Inicial

4.2.3. Algoritmo *RBIR*

Se ha dado el nombre general de *RBIR* (Red Blue with Inclusion Relations) a un conjunto de algoritmos que combinan el procesamiento del algoritmo *redBlue* con la inferencia de valores de salida. Es decir, se trabaja siguiendo la estrategia blue-fringe para seleccionar las parejas de estados candidatas a mezclarse, se

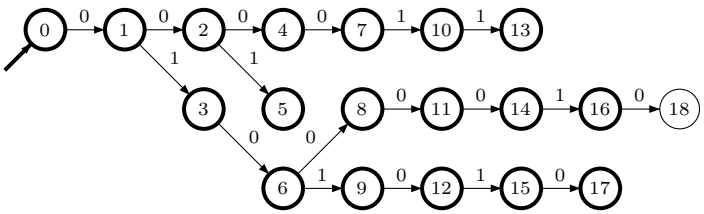


Figura 4.16: Máquina de Moore Después de Haber Definido Valores de Salida

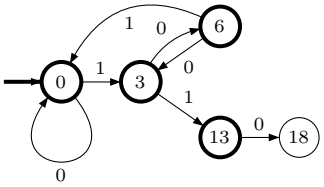


Figura 4.17: Salida Producida por el Algoritmo *IRPNI1*

calcula el puntaje asociado a cada pareja candidata a la manera que lo hace *redBlue* y se exploran diversas formas de hacer inferencia de valores de salida durante el proceso.

La idea de combinar la estrategia *redBlue* con la inferencia de valores de salida, pretende aumentar el desempeño de *redBlue* en la inferencia de lenguajes regulares generados con expresiones regulares o NFAs, ya que en este contexto es superado por otros algoritmos como *IRPNI2* y *DeLeTe2*. La inferencia de valores de salida mostró ser útil para este propósito al aumentar significativamente el desempeño del algoritmo *RPNI*, dando origen al algoritmo *IRPNI2*. Al hacer la combinación de los dos esquemas hay que tener cuidado con lo siguiente: en el momento en que se introduce algún valor de salida nuevo al modelo, esto puede modificar tanto los puntajes ya calculados como las decisiones tomadas en cuanto a las viabilidad de algunas mezclas. A continuación se presenta el algoritmo base que aplica estas ideas.

El Algoritmo 32 parte del árbol de prefijos de Moore de las muestras de entrada, inicializa el conjunto S de estados que pertenecen a la hipótesis y el conjunto frontera T, luego itera mientras que la frontera no sea vacía. En cada iteración se asigna un puntaje a cada pareja de estados (p, q) tal que $p \in S \wedge q \in T \wedge p \simeq q$. En caso que se detecte algún estado $q \in T$ distinguible de todos los estados en S, se realizan todas las definiciones de valores de salida posibles con tal estado y luego se adiciona al conjunto S; en otro caso se realiza la mezcla de estados que haya obtenido el máximo puntaje.

Lema 67 *El algoritmo* RBIR *converge en el límite al DFA mínimo del lenguaje objetivo.*

Demostración. El algoritmo *RBIR* realiza mezclas de estados en un orden establecido por los puntajes que calcula con base en las muestras de entrada. Según la Proposición 34 este procedimiento converge en el límite al DFA mínimo

Algoritmo 32 $RBIR(D_+, D_-)$

1: $M := APM(D_+ \cup D_-)$ // $M = (Q, \Sigma, \{0, 1, ?\}, \delta, \Phi, q_0)$
2: $S := \{q_0\}$
3: $T := S\Sigma - S$
4: **while** $T \neq \{\}$ **do**
5: $listM := \{\}$
6: $listP := \{\}$
7: **for** $q \in T$ **do**
8: $mezclable := falso$
9: **for** $p \in S$ **do**
10: **if** $p \simeq q$ **then**
11: $mezclable := verdadero$
12: $listM := listM \cup \{p, q, puntaje(p, q))\}$
13: **end if**
14: **end for**
15: **if** $\neg mezclable$ **then**
16: $listP := listP \cup \{q\}$
17: **end if**
18: **end for**
19: **if** $listP \neq \{\}$ **then**
20: $seleccionado := first(listP)$
21: **for** $p \in S$ **do**
22: **if** $p \prec seleccionado$ **then**
23: $M := definirEstados(M, p, seleccionado)$
24: **end if**
25: **if** $seleccionado \prec p$ **then**
26: $M := definirEstados(M, seleccionado, p)$
27: **end if**
28: **end for**
29: $S := S \cup \{seleccionado\}$
30: **else**
31: $(p, q, puntaje) := maximoPuntaje(listM)$
32: $M := mezcladet(M, p, q)$
33: **end if**
34: $T := S\Sigma - S$
35: **end while**
36: Return M

del lenguaje objetivo. Adicionalmente a la mezcla, este algoritmo define el valor de salida de algunos estados; como ya se ha comentado en otros algoritmos, este hecho no afecta la convergencia, ya que en el límite no hay posibilidad de definir valores de salida porque todos están ya definidos. Y si a causa de definir prematuramente un valor de salida, se causara alguna inexactitud en el proceso de inferencia, esto será reparado a medida que se obtenga mayor información de entrada. ■

Lema 68 *El coste temporal del algoritmo RBIR es $O(n^4)$ siendo n el tamaño de la muestra de entrada.*

Demostración. El ciclo que comienza en la línea 4 del Algoritmo 32 iterará sobre todos los estados del modelo, por lo tanto tiene un coste $O(n)$; internamente, en las líneas 7 y 9 hay dos ciclos que iteran $O(n)$ veces cada uno; en la línea 10, se evalúa si dos estados son equivalentes, ello implica construir y recorrer el árbol de mezcla correspondiente, lo que en peor caso cuesta $O(n)$. En total el coste es $O(n^4)$. Si se observan los ciclos anidados en las líneas 4 y 21 junto con la invocación a la subrutina *definirEstados* de coste lineal, su coste temporal es $O(n^3)$. Por lo tanto, el coste total del algoritmo *RBIR* es $O(n^4)$.

Notar que en la línea 12, en caso que los estados sean equivalentes se debe calcular el puntaje, lo cual requiere recorrer de nuevo el árbol de mezcla, esto podría llevar a pensar que el coste se eleva en un orden de magnitud. Sin embargo, esto no ocurre ya que el conteo requerido para establecer el puntaje se puede hacer simultáneamente con la evaluación de equivalencia entre estados (línea 10) sin que ello incremente el coste computacional. ■

Se han explorado variaciones a esta estrategia, las más significativas se describen a continuación:

- Elegir aleatoriamente el estado a promover, en lugar de elegir el primero que se ha detectado.
- Realizar la inferencia de valores de salida con todos los estados promovibles y posteriormente elegir uno cualquiera de ellos para promoverlo.
- Establecer un esquema de puntajes para elegir el estado a promover. Tal puntaje debe asignar un valor alto al estado que, según la información disponible, es el más distinto de los demás. El puntaje se calcula contando las parejas de estados de la forma $(+x, -y)$ ó $(-x, +y)$ en los árboles de mezcla que resultan al comparar un estado del conjunto de los promovibles $listP$ con todos los estados del conjunto S. El Algoritmo 33 ilustra las líneas que reemplazan a la línea 20 en el Algoritmo 32 para realizar esta estrategia.
- Asociar un puntaje no sólo a las parejas que tienen posibilidad de mezclarse, sino también a las parejas que tienen posibilidad de definir valores de salida. El puntaje de la pareja (p, q) tal que, $p \prec q$ (resp. $q \prec p$), se mide contando en su árbol de mezcla las parejas de la forma $(+x, +y)$, $(-x, -y)$ y $(-x, +y)$ (resp. $(+x, -y)$). Se realiza la acción que haya obtenido el máximo puntaje. Una vez realizada la acción se recalculan todos los puntajes y se vuelve a seleccionar.

Algoritmo 33 Modificación al Algoritmo 32

1: $listP' := \{\}$
2: **for** $q \in listP$ **do**
3: $\quad acum := 0$
4: $\quad$ **for** $p \in S$ **do**
5: $\quad\quad acum := acum + puntajePromover(M, p, q)$
6: $\quad$ **end for**
7: $\quad listP' := listP' \cup \{(q, acum)\}$
8: **end for**
9: $seleccionado := maximoPuntaje(listP')$

- En la línea de la variación anterior, se puede elegir el estado con mayor puntaje para la mezcla y realizar todas las definiciones de valores de salida que sean posibles relacionadas con él. Si la información adicional hace que la mezcla ya no sea posible, el estado se promueve.
- Se usa un puntaje inverso para elegir el estado a promover, es decir, se escoge el estado que según las muestras sea más diferente de los estados de la hipótesis, para eso se cuentan las parejas $(+x, -y)$ ó $(-x, +y)$ y se promueve el estado que tenga un puntaje mayor.

4.2.4. Algoritmo *RRB*

El algoritmo RRB explora la combinación de los algoritmos *IRPNI2* y *redBlue* con el fin de mejorar las prestaciones del segundo cuando el lenguaje objetivo proviene de una expresión regular o de un NFA. La manera de proceder se presenta en el Algoritmo 34. Se calculan los prefijos de todas las muestras positivas y negativas, se ejecuta sobre dicha muestra el algoritmo *IRPNI2*, posteriormente se reetiquetan muestras y prefijos con el modelo obtenido por *IRPNI2*, lo que aumenta la información de partida para finalmente ejecutar con esta nueva muestra el algoritmo *redBlue*.

Algoritmo 34 $RRB(D_+, D_-)$

1: $S_{D_+} := Pref(D_+)$
2: $S_{D_-} := Pref(D_-)$
3: $M := IRPNI2(D_+, D_-)$
4: $T_{D_+} := etiquetar(S_{D_+}, M)$
5: $T_{D_-} := etiquetar(S_{D_-}, M)$
6: $M := redBlue(T_{D_+}, T_{D_-})$
7: Return M

Lema 69 *El algoritmo* RRB *converge al DFA mínimo del lenguaje objetivo.*

Demostración. Se puede ver que existe una enumeración de Σ^* correspondiente al orden de mezcla determinado por los puntajes que calcula el algoritmo *redBlue*, por lo tanto, es posible construir una muestra característica para esta enumeración y obtener la convergencia al DFA mínimo del lenguaje objetivo, según la Proposición 34.

En cuanto al paso previo de ejecutar el algoritmo *IRPNI2*, se ha mostrado anteriormente que él también converge (ver Proposición 61), sin embargo eso no es relevante en este caso, ya que aquí se utiliza únicamente para etiquetar muestras con la hipótesis que produce. ■

Lema 70 *El coste temporal del algoritmo RRB es* $O(n^3)$ *en el peor caso.*

Demostración. El coste de calcular los prefijos de una muestra es el tamaño de dicha muestra, en este caso n. El coste de ejecutar el algoritmo *IRPNI2* es $O(n^3)$ en el peor caso, el coste de etiquetar una muestra de tamaño n con una máquina de tamaño m es $O(nm)$ y finalmente, el coste del algoritmo *redBlue* es $O(n^3)$. Por lo tanto, el coste total del algoritmo *RRB* es $O(n^3)$. ■

4.3. Algoritmos Propuestos que Hacen Mezcla No Determinista

4.3.1. Algoritmo *NRPNI*

El algoritmo *NRPNI* (Non-deterministic Regular Positive Negative Inference) [AGR06], que se presenta en el Algoritmo 35, parte de un conjunto de muestras positivas y negativas (D_+, D_-) de un lenguaje objetivo L definido sobre un alfabeto Σ. Inicialmente se aplica el algoritmo *RPNI* a la muestra, obteniendo un modelo M. Luego se procura encontrar una hipótesis de tamaño más pequeño que sea consistente con las muestras, para ello se adicionan nuevas transiciones a M de acuerdo a las relaciones de inclusión entre los estados. Las nuevas transiciones pueden convertir M en una máquina no determinista. Si *NRPNI* no encuentra una hipótesis de menor tamaño, retorna como salida el modelo que produjo *RPNI*.

Una copia de la respuesta producida por *RPNI* se almacena en M_1, en tanto que M se convierte en un árbol que conserva sólo las transiciones que existan en los caminos de longitud mínima para alcanzar cada estado desde q_0, la Figura 4.19 muestra los caminos de longitud mínima ($CLM(M_1)$) correspondientes al autómata resultante de aplicar el algoritmo *RPNI* a las muestras de entrada $D_+ = \{\varepsilon, 00, 11, 010, 10\}$ y $D_- = \{0, 1, 001\}$ (Ver Figura 4.18). Notar que para romper los ciclos en el autómata original, el árbol de caminos de longitud mínima puede utilizar estados ficticios, que se reconocen en el diagrama por no tener identificador.

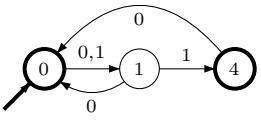


Figura 4.18: Salida del Algoritmo *RPNI*

Algoritmo 35 $NRPNI(D_+, D_-)$

1: $M := RPNI(D_+, D_-)$ // $M = (Q, \Sigma, \delta, q_0, F)$
2: $M_1 := M$ // $M_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$
3: $M := CLM(M)$ // Caminos de longitud mínima desde q_0 hasta cada estado
4: $S := \{q_0\}$; $I = \{q_0\}$; $\delta' := \delta_1$
5: **while** $|S| < |Q_1|$ **do**
6: **for** $(p, q) : p \in S, q \in S\Sigma - S$ // En orden lexicográfico sobre S,$S\Sigma - S$ y $p \ll q$ **do**
7: // Intenta la relación $q \prec p$
8: $\delta' = \delta' \cup \{(r, a, q) \in \delta' : (r, a, p) \in \delta', r \in Q_1, a \in \Sigma\}$
9: **if** $M' = (Q_1, \Sigma, \delta', I, F_1)$ es consistente con D_- **then**
10: **if** $p = \varepsilon$ **then**
11: $I = I \cup \{q\}$
12: **end if**
13: **else**
14: $\delta' := \delta' - \{(r, a, q) \in \delta' : (r, a, p) \in \delta', r \in Q_1, a \in \Sigma\}$
15: // Intenta la relación $p \prec q$
16: $\delta' = \delta' \cup \{(r, a, p) \in \delta' : (r, a, q) \in \delta', r \in Q_1, a \in \Sigma\}$
17: **if** $M' = (Q_1, \Sigma, \delta', I, F_1)$ no es consistente con D_- **then**
18: $\delta' := \delta' - \{(r, a, p) \in \delta' : (r, a, q) \in \delta', r \in Q_1, a \in \Sigma\}$
19: **end if**
20: **end if**
21: **end for**
22: M" =Subautómata inducido por S en $(Q_1, \Sigma, \delta', I, F_1)$
23: **if** M" es consistente con D_+, D_- **then**
24: Return M"
25: **else**
26: $S := S \cup first(S\Sigma - S \cap Q)$ // En orden lexicográfico
27: **end if**
28: **end while**
29: Return M_1

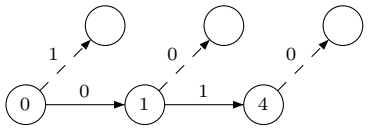


Figura 4.19: Caminos de Longitud Mínima $CLM(M)$ Relacionados con la Máquina de la Figura 4.18.

La máquina M se utiliza para calcular el conjunto de estados de la hipótesis actual S y el conjunto frontera $S\Sigma - S$, mientras M_1 es la base para construir hipótesis potencialmente más pequeñas. Al principio $S = \{q_0\}$. El conjunto de transiciones de la hipótesis actual está formado por las transiciones de M que relacionan estados de S y también por nuevas transiciones generadas a partir de las relaciones de inclusión entre los estados $p \in S$ y $q \in S\Sigma - S$. A partir de la línea 6 se supone la inclusión en el sentido $q \prec p$, es decir, se duplican las transiciones que llegan a p para que también lleguen a q aplicando las ideas explicadas en la Sección 3.1.2 y si ello no conduce a una máquina consistente con las muestras negativas D_-, se eliminan las transiciones adicionadas (línea 14) y se intenta la inclusión contraria (a partir de la línea 15). Notar que si $q \prec \varepsilon$, entonces p se adiciona a los estados iniciales (línea 11). Una vez se han comparado todos los elementos de $S\Sigma - S$ con todos los elementos de S, se obtiene una nueva hipótesis M'' que, si es consistente con la muestra de entrada D_+, D_- es la respuesta; de otro modo se adiciona un nuevo estado al conjunto S (línea 26) y comienza una nueva iteración. Si el tamaño del conjunto S alcanza el mismo tamaño que la hipótesis M obtenida con *RPNI*, se retorna M.

En pruebas empíricas se encontró que aplicar el heurístico de poda de arcos sobre la hipótesis final de *NRPNI* mejora la tasa de desempeño del algoritmo; la poda consiste en reconsiderar todos los arcos que fueron adicionados a causa de relaciones de inclusión, si no son necesarios para garantizar la consistencia con las muestras de entrada son eliminados. El Algoritmo 36 ilustra en detalle el procedimiento que debe realizarse insertando inmediatamente después de la línea 23 del Algoritmo 35 la instrucción $M'' := podar(M'', D_-)$.

Algoritmo 36 *podar*(M, D_-)

Require: $M = (Q, \Sigma, \delta, I, F)$
1: **for** $(r, a, p) \in \delta$ tal que fue adicionado al modelo a causa de una relación de inclusión **do**
2: $\quad \delta := \delta - \{(r, a, p)\}$
3: $\quad$ **if** M no es consistente con D_- **then**
4: $\quad\quad \delta := \delta \cup \{(r, a, p)\}$
5: $\quad$ **end if**
6: **end for**
7: Return M

Lema 71 *Se puede construir una muestra característica que haga que* NRPNI *converja a un modelo que tendrá a lo sumo tantos estados como el DFA mínimo*

del lenguaje objetivo. El procedimiento a seguir es muy similar al utilizado para construir una muestra característica para RPNI.

Demostración. Dado un DFA mínimo para el lenguaje objetivo L, el conjunto de prefijos más cortos $Pc(L)$, formado por las palabras más cortas en orden lexicográfico que conectan el estado inicial con cada estado y $K(L)$ el conjunto núcleo $Pc \cup Pc\Sigma$, donde Σ es el alfabeto de L, la muestra característica debe contener las siguientes palabras:

- Para todo elemento de $K(L)$ debe existir una completación en el lenguaje objetivo, en el caso que la palabra llegue a un estado final, la completación debe ser ε.
- Para todo par $(u, v) \in Pc(L) \times K(L)$ debe existir una palabra w tal que $(uw, vw) \in (D_+ \times D_-) \cup (D_- \times D_+)$.

Para distinguir los estados u, v tal que $u \prec v$ ($L_u \subseteq L_v$) debe tenerse una palabra w tal que $vw \in D_+$ y $uw \in D_-$ y no debe existir w tal que $uw \in D_+$ y $vw \in D_-$. En caso que $u \not\preceq v$, y además $u \not\prec v$ y $u \not\succ v$, se necesitan dos palabras x, y tales que $ux \in L \wedge vx \notin L$ y $uy \notin L \wedge vy \in L$. Así, si dos estados son distinguibles con respecto a equivalencia, también son distinguibles con respecto a la relación de inclusión y en caso que no estén relacionados ni por equivalencia ni por inclusión basta con incluir dos prefijos adicionales para garantizar que la inferencia no los relacione equivocadamente. Una vez construida la muestra característica, la convergencia se da de la misma manera que en el algoritmo *RPNI*. El peor caso ocurre cuando no hay estados compuestos, porque es necesario distinguir todos los estados entre sí. ■

Lema 72 *El algoritmo* NRPNI *converge a un RFSA de tamaño menor o igual al DFA mínimo del lenguaje objetivo y mayor o igual que el tamaño del RFSA canónico.*

Demostración. La primera afirmación viene del hecho que *NRPNI* nunca va a producir como salida un modelo más grande que *RPNI*. La segunda del hecho que pueden haber estados compuestos que, en orden lexicográfico, aparezcan previamente al último estado primo en el autómata de salida. ■

Lema 73 *El coste temporal del algoritmo* NRPNI *es* $O(n^3)$

Demostración. El primer paso de *NRPNI* consiste en invocar a *RPNI*, que tiene un coste en peor caso $O(n^3)$. El procesamiento siguiente tiene un coste $O(n^2)$ debido al coste $O(n)$ del ciclo en la línea 5 y al coste $O(n)$ de la verificación de consistencia en las líneas 9 o 17. Así, el coste temporal del algoritmo completo es, en el peor caso, $O(n^3)$.

El peor caso ocurre cuando no hay estados compuestos en la máquina de Moore inicial y se realizan todas las iteraciones sin obtener ninguna hipótesis consistente con las muestras. ■

Ejemplo 74 *Sea* $D_+ = \{\varepsilon, 00, 10, 11, 010\}$ *y* $D_- = \{0, 1, 001\}$*, el árbol de prefijos de Moore para estas muestras se presenta en la Figura 4.20. A continuación se comenta la ejecución del algoritmo* NRPNI *a partir de estos datos.*

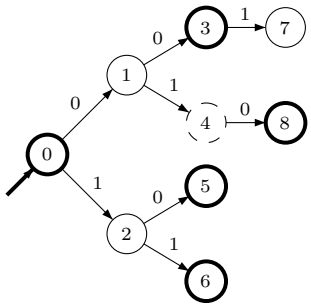


Figura 4.20: Árbol de Prefijos de Moore

Se encuentran las siguientes relaciones entre estados: $(0, 1)$ *y* $(0, 2)$ *son distinguibles, mientras* $1 \simeq 2$*,* $0 \simeq 3$ *y* $1 \simeq 4$*. También se detecta que* $1 \prec 3$ *y* $0 \prec 4$*.*

NRPNI *comienza emitiendo la hipótesis de la Figura 4.21 (a) que no es consistente con las muestras, entonces* $S = \{0, 1\}$*. Posteriormente se mezclan los estados* 1 *y* 2 *y se detectan las relaciones de inclusión:* $0 \simeq 3, 1 \prec 3, 0 \prec 4$ *y* $1 \prec 4$*. Esto da origen a la siguiente hipótesis que se muestra en la Figura 4.21 (b) la cual es consistente con las muestras y por lo tanto es la respuesta del algoritmo que termina.*

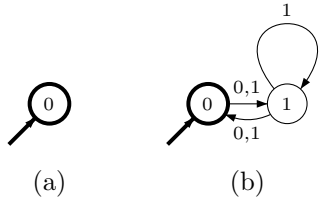


Figura 4.21: Hipótesis Generadas por *NRPNI*.

4.3.2. Algoritmo *MRIA*

El algoritmo *MRIA* (Merging Residuals Inference Algorithm) [AGR06] es un algoritmo de mezcla de estados que, a partir de una muestra completa, produce como salida un autómata de estado finito no determinista consistente con la entrada. A diferencia de otros algoritmos presentados anteriormente, esta estrategia no modifica ningún algoritmo previo. La idea tras el algoritmo es la de realizar un recorrido en orden lexicográfico sobre las parejas de estados suponiendo relaciones de inclusión en ambos sentidos y verificando la consistencia de tales suposiciones mediante las muestras negativas. Las relaciones de

inclusión en ambos sentidos se manejan como relaciones de equivalencia y causan la mezcla de los dos estados, cuando dos estados no se mezclan, se genera una hipótesis a partir del subautómata inducido por los estados ya procesados y si dicha hipótesis es consistente con la muestra de entrada, el algoritmo termina.

El algoritmo 37, que comienza construyendo el árbol de prefijos de Moore de las muestras positivas, consta de dos ciclos anidados: el externo itera sobre cada estado q del árbol en orden lexicográfico, el ciclo interno itera sobre todos los estados p menores que q también en orden lexicográfico. Primero se revisa si p está *incluido* en q ($p \prec q$); para ello se invoca la subrutina $compare(\mathcal{A}, p, q)$ que será descrita más adelante. Si la respuesta es positiva, las transiciones resultantes de dicha relación permanecen y en caso contrario se deshacen. Además, si el estado q no ha sido agrupado previamente con otro, se evalúa $Compare(\mathcal{A}, q, p)$; si también se cumple, los estados p y q son agrupados mediante la subrutina $nmerge(\mathcal{A}, p, q)$, que es la versión para máquina no determinista de la Subrutina 12. Vale resaltar que el agrupamiento no se propaga, así que en este caso no se mantiene el determinismo de la máquina.

Algoritmo 37 $MRIA(D_+, D_-)$

1: $\mathcal{M} := APM(D_+)$
2: $S = \{\}$
3: $lista := \{q_0, q_1, \ldots, q_r\}$ //estados de M en orden lexicográfico, $q_0 = \lambda$
4: **for** `q` $\in$ `lista` **do**
5: $agrupado = falso$
6: **for** `p` $\ll$ `q` (en orden lexicográfico) **do**
7: Compare $(\mathcal{M}, p, q, D_-)$
8: **if** $\neg agrupado$ **then**
9: Compare $(\mathcal{M}, q, p, D_-)$
10: **end if**
11: **if** $\neg agrupado$ y $p \prec q$ y $q \prec p$ **then**
12: $nmerge(M, p, q)$
13: $agrupado = verdadero$
14: **end if**
15: **end for**
16: **if** $agrupado$ **then**
17: $Q = Q - \{q\}$
18: **else**
19: $S = S \cup \{q\}$
20: $\mathcal{H}$:= Autómata inducido por S en M
21: **if** $\mathcal{H}$ es consistente con D_+, D_- **then**
22: Return $\mathcal{H}$
23: **end if**
24: **end if**
25: **end for**
26: $\mathcal{H}$:= Autómata inducido por S en M
27: Return $\mathcal{H}$

Al final de cada ciclo interno, si el estado q no ha sido agrupado con ninguno

de los estados anteriores a él, se emite una nueva hipótesis de inferencia que es el autómata inducido por el conjunto de estados previos a q. Notar que aun si el estado q ha sido agrupado con otro previamente, él sólo es borrado después de haber sido comparado con el resto de sus predecesores. Si no se hiciera de esta manera, algunas transiciones no podrían llegar a generarse.

La relación de inclusión entre estados se implementa con la subrutina *compare*(M, p, q) (ver Algoritmo 38) que envía al estado p todas las transiciones que llegan al estado q; ellas permanecerán en caso que el autómata resultante sea consistente con las muestras negativas. Notar también que si la relación se cumple y q es estado inicial, p se vuelve inicial.

Algoritmo 38 *compare*(M, p, q, D_-)

Require: $M = (Q, \Sigma, \delta, I, F)$, $p, q \in Q$
1: $I' = I$
2: $\delta' = \delta$
3: **for all** $(r, a, q) \in \delta$ **do**
4: $\quad \delta := \delta \cup \{(r, a, p)\}$
5: **end for**
6: **if** $q \in I$ **then**
7: $\quad I := I \cup \{p\}$
8: **end if**
9: **if** M no es consistente con D_- **then**
10: $\quad I = I'$
11: $\quad \delta = \delta'$
12: **end if**

El agrupamiento de estados se realiza en la subrutina *nmerge*(M, p, q) (ver el Algoritmo 39). Si el valor de salida de p es indefinido ($\Phi(p) =?$) y el valor de salida de q no es indefinido ($\Phi(q) \neq?$), p toma el valor de salida de q. Las transiciones que llegan a y parten de q son redireccionadas a p. Si q es estado inicial, p se convierte en estado inicial.

Con el objetivo de evitar la sobregeneralización en el proceso de inferencia, se puede incluir en el algoritmo *MRIA* un heurístico de poda de arcos innecesarios. La idea de hacer esta modificación surge al observar que las tasas de reconocimiento de *MRIA* para las muestras de prueba negativas suelen ser más bajas que las tasas para muestras positivas. El heurístico consiste en eliminar de la hipótesis final aquellos arcos que provienen de relaciones de inclusión (no de agrupamientos) y que no son indispensables para mantener la consistencia con las muestras de entrenamiento. Este heurístico no afecta ni la convergencia ni la complejidad del algoritmo. Para incluir el heurístico basta con insertar la instrucción $\mathcal{H} :=$ *podar*$(\mathcal{H}, D_-)$ después de la línea 26 del Algoritmo 37. La subrutina podar ya ha sido presentada en el Algoritmo 36.

Lema 75 *El algoritmo* MRIA *converge a un autómata no determinista que reconoce el lenguaje objetivo* L.

Algoritmo 39 *nmerge*(M, p, q)

Require: $M = (Q, \Sigma, \{0, 1, ?\}\delta, \Phi, q_0)$
Require: $p, q \in Q$, $p \ll q$ en orden lexicográfico
1: **if** $\Phi(p) \neq ? \wedge \Phi(q) \neq ? \wedge \Phi(p) \neq \Phi(q)$ **then**
2: Return M
3: **end if**
4: **if** $\Phi(p) = ?$ **then**
5: $\Phi(p) := \Phi(q)$
6: **end if**
7: **for** $s \in \Sigma$ **do**
8: **for** $x : \delta(x, s) = q$ **do**
9: $\delta := \delta \cup \{\delta(x, s) = p\}$
10: $\delta := \delta \setminus \{\delta(x, s) = q\}$
11: **end for**
12: **end for**
13: **for** $s \in \Sigma$ **do**
14: **for** $x : \delta(q, s) = x$ **do**
15: $\delta := \delta \cup \{\delta(p, s) = x\}$
16: $\delta := \delta \setminus \{\delta(q, s) = x\}$
17: **end for**
18: **end for**
19: **if** $q \in I$ **then**
20: $I := I \cup \{p\}$
21: **end if**
22: $Q := Q \setminus \{q\}$
23: Return M

Demostración. En el límite, el algoritmo recibe suficientes muestras negativas y positivas para producir como hipótesis un autómata irreducible con respecto al lenguaje objetivo L, porque lo único que impide la mezcla de estados son las muestras negativas y si ellas son suficientes garantizan la irreducibilidad. Aplicando el Lema 39, se tiene que el autómata obtenido es un subautómata del autómata Universal de L y que reconoce L. ■

Lema 76 *El algoritmo* MRIA *tiene un coste de* $O(n^3)$

Demostración. Sea n el tamaño de la muestra de entrada. El algoritmo *MRIA* compara cada estado con todos los que son menores que él en orden lexicográfico, esto tiene un coste $O(n^2)$ en el peor caso. Para cada pareja de estados, se evalúa la consistencia de la hipótesis con la muestra de entrada, lo que tiene un coste lineal en función de n. Por lo tanto, el coste total del algoritmo es $O(n^3)$. ■

Ejemplo 77 *A continuación se muestra un ejemplo del comportamiento del algoritmo* MRIA*. Sea* $D_+ = \{\varepsilon,\ 00,\ 10,\ 11,\ 010\}$ *y* $D_- = \{0,\ 1,\ 01{,}001\}$*, el árbol de prefijos de Moore para ella está en la Figura 4.22.*

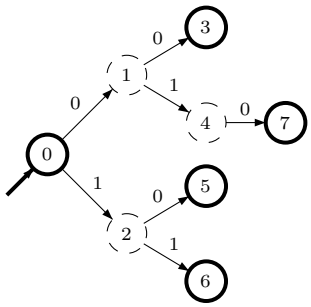


Figura 4.22: Árbol de Prefijos de Moore de D_+

MRIA *compara los estados* 0 *y* 1 *(*$0 \prec 1$*). Para hacerlo, todas las transiciones que llegan a* 1 *son enviadas también a* 0*. Entonces se revisa la consistencia del modelo resultante con las muestras negativas (en la subrutina* compare*, ver Algoritmo 38). La máquina extendida se muestra en la Figura 4.23 (a), se puede ver que esta máquina acepta la muestra negativa* 0*, así que la respuesta a esta comparación es no. A continuación se intenta la comparación contraria (*$1 \prec 0$*), cuyo resultado se observa en la Figura 4.23 (b), que también acepta* 0*. Así que el algoritmo continúa con el cálculo de la primera hipótesis, que no es consistente con las muestras, como se aprecia en la Figura 4.25 (a).*

El estado 2 *debe compararse con los estados* 0 *y* 1*. La única comparación que retorna verdadero es la de los estados* 1 *y* 2*, lo que causa que se adicione la transición* $\delta(0,1) = 1$ *tal como se aprecia en la Figura 4.24 (a). Se calcula una nueva hipótesis que es la de la Figura 4.25 (b).*

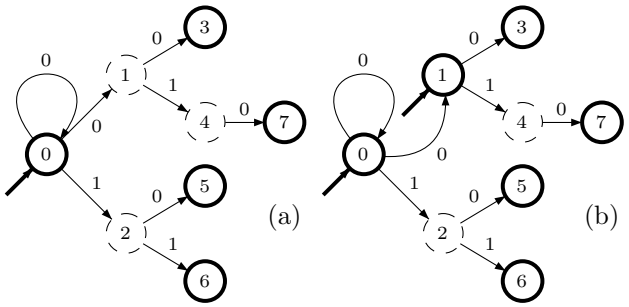


Figura 4.23: Comparaciones Entre los Estados 0 y 1

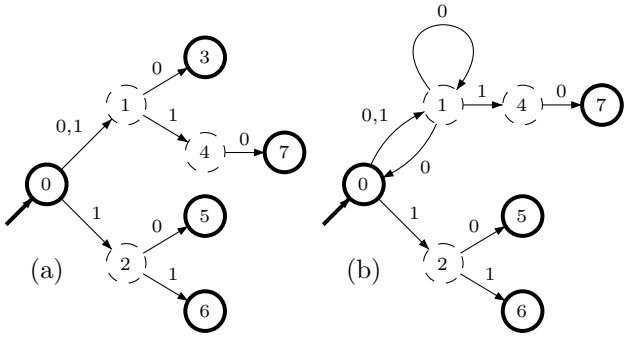


Figura 4.24: Máquinas de Moore Intermedias

Como las dos comparaciones entre 0 *y* 3 *son verdaderas, estos estados se mezclan. Notar que se continúa comparando al estado* 3 *con el resto de los estados previos. Luego se detecta que* $1 \prec 3$ *y el resultado se muestra en la Figura 4.24 (b), posteriormente se encuentra que los estados* 1 *y* 4 *son equivalentes, que* $2 \prec 5$*, que* 0 *y* 5 *son equivalentes y así sucesivamente hasta obtener la última hipótesis que es la de la Figura 4.25 (c).*

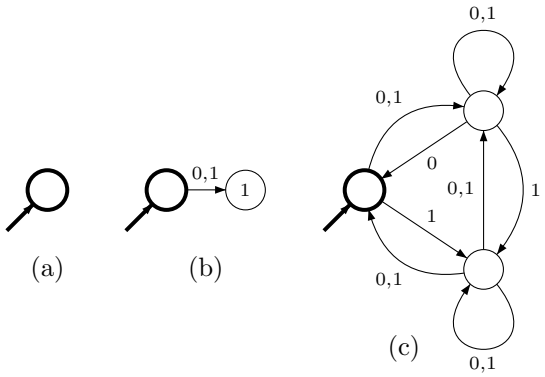


Figura 4.25: Hipótesis Emitidas por *MRIA*

4.4. Sumario

En los algoritmos que se han presentado en este capítulo se han considerado estrategias que conducen a producir modelos deterministas o no deterministas. Al combinar la estrategia de mezcla de estados con la definición de valores de salida, se exploraron dos alternativas que producen hipótesis de inferencia deterministas:

- Realizar primero todas las definiciones de valores de salida posibles a partir de la información provista por las muestras de entrenamiento y luego realizar las mezclas al estilo *RPNI*. Este es el algoritmo *IRPNI1*.
- Ir realizando las mezclas de estados al estilo *RPNI* y en cada comparación de estados que no lleve a una mezcla realizar las definiciones de valores de salida que sean posibles. Este es el algoritmo *IRPNI2*.

Al combinar la estrategia de mezcla de estados al estilo *redBlue* con la definición de valores de salida, se exploraron dos alternativas que también conducen a hipótesis de inferencia deterministas:

- Realizar la definición de valores de salida sólo en caso que se vaya a promover un estado azul al conjunto rojo. Este es el algoritmo *RBIRX*.
- Dar puntaje a las comparaciones que potencialmente pueden llevar a la definición de nuevos valores de salida, siguiendo un procedimiento similar al que se usa para dar puntaje a las posibles mezclas y elegir aquella acción que obtenga el máximo puntaje. Este es el algoritmo *RBIRZ*.

También se ha evaluado una estrategia de inferencia que combina la ejecución de los algoritmos *IRPNI2* y *redBlue*. En ella se empieza ejecutando el algoritmo *IRPNI2* con las muestras de entrenamiento, posteriormente se extraen todos los prefijos sin valor de salida asociado del árbol de prefijos de Moore de la muestra de entrenamiento, se les asocia valor de salida con la hipótesis que produjo *IRPNI2* y se incluyen en la muestra de entrenamiento. Por último, se ejecuta el algoritmo *redBlue* sobre la muestra extendida. Este es el algoritmo *RRB*.

La inclusión de nuevos arcos se propone como un postprocesamiento que puede aplicarse a la hipótesis obtenida por otro método de inferencia con el fin de disminuir el tamaño de la hipótesis de inferencia. El postprocesamiento consiste en detectar relaciones de inclusión y duplicar los arcos que llegan al estado incluyente para que lleguen también al estado incluido. Si esta estrategia se aplica a la salida de *RPNI* es el algoritmo *NRPNI*; si se aplica a la salida de *IRPNI1* es el *NRPNI1* y si se aplica a la salida de *IRPNI2* es *NRPNI2*.

Otra alternativa que se exploró fue adicionar relaciones de inclusión siempre que ellas sean consistentes con la muestra de entrenamiento y mezclar estados cuando se detecte la relación de inclusión en ambos sentidos. Este es el algoritmo *MRIA*.

Tanto los algoritmos *NRPNI** como *MRIA* pueden incluir un heurístico de poda que limita el número de arcos de la hipótesis final al tiempo que garantiza

la consistencia con las muestras de entrenamiento. Para realizar la poda, durante el proceso de inferencia se marcan los arcos que son añadidos a causa de una relación de inclusión. Cuando se obtiene la consistencia de la hipótesis de inferencia, se reconsideran todos los arcos que provienen de relaciones de inclusión y se eliminan siempre y cuando al hacerlo no se pierda la consistencia con las muestras de entrenamiento. Este heurístico disminuye la sobregeneralización del modelo de inferencia.

Dado el antecedente del algoritmo *DeLeTe2* que infiere RFSAs y utiliza información sobre las relaciones de inclusión entre estados, es de esperar que estos algoritmos muestren mejor desempeño que los algoritmos en que se basan ya que todos ellos involucran, de una u otra manera, la información que aportan las relaciones de inclusión entre estados y esta información no es considerada por los algoritmos de referencia *RPNI* y *redBlue*. En cuanto al algoritmo *DeLeTe2*, prácticamente todas las alternativas propuestas (salvo RBIR que tiene su misma cota de complejidad) tienen una complejidad temporal inferior a él en un orden de magnitud.

Capítulo 5

Resultados Experimentales

5.1. Datos de Entrenamiento y Prueba

A continuación se presentan las bases de datos que se han utilizado para realizar las pruebas de los algoritmos de inferencia gramatical. Todas ellas tienen en común algunas características, por ejemplo usar siempre un alfabeto $\Sigma = \{0, 1\}$, y utilizar los formatos de archivos del Corpus de DeLeTe2. Este formato se aplica a la forma de presentar las muestras y también a la forma de almacenar los autómatas. En la Figura 5.1 se observa un pequeño archivo de muestras que ilustra el formato: en la primera línea hay dos valores, el primero de ellos indica el número de muestras que contiene el archivo y el segundo el número de símbolos del alfabeto. A partir de allí cada línea es una muestra etiquetada, el primer valor de una línea es la etiqueta: marca 1 si la muestra es positiva y 0 si es negativa, el segundo valor es la longitud de la muestra y a partir del tercer dato se presenta la muestra propiamente dicha. Por ejemplo, la segunda línea del archivo indica que la muestra vacía es positiva, la siguiente muestra es 0 y es negativa.

```
8 2
1 0
0 1 0
1 2 0 0
0 3 0 0 1
1 3 0 1 0
0 1 1
1 2 1 0
1 2 1 1
```

Figura 5.1: Formato de los Archivos de Muestras

En cuanto al formato para representar los autómatas, se puede ver un ejemplo en la Figura 5.2, gracias a los comentarios el resultado es bastante legible, lo primero que se establece es el alfabeto enumerando los símbolos que lo componen. A continuación se indica el número de estados, posteriormente se refiere a los estados iniciales, el primer valor indica cuántos estados iniciales hay y a

continuación se enumeran; igual ocurre con los estados finales, primero se dice cuántos son y luego cuáles. Por último, se indica cuántas transiciones hay y se muestra una transición en cada línea, el primer valor es el identificador del estado inicial, el segundo valor es el símbolo que etiqueta la transición y el tercero es el identificador del estado de llegada. En este formato los estados siempre están identificados con los enteros entre cero y el número de estados menos uno.

```
# Alfabeto
{0,1}

# Numero de estados
3
# Estados iniciales
1 0
# Estados finales
2 0 2
# Descripcion de las transiciones
5
0 0 1
0 1 1
1 0 0
1 1 2
2 0 0
```

Figura 5.2: Formato de los Archivos de Autómatas

El método de generación de muestras también es común a los diferentes corpus. Consiste en obtener en primer lugar la longitud de la muestra, generando un número aleatorio entre cero y la longitud máxima permitida, luego se generan tantos números aleatorios como sean necesarios en el rango del tamaño del alfabeto, para generar la muestra propiamente dicha.

5.1.1. Corpus Utilizado para Evaluar el Algoritmo DeLeTe2

Dado que uno de los puntos de partida de esta investigación ha sido el algoritmo DeLeTe2, y que los datos usados para ilustrar su comportamiento en [DLT04] están disponibles en internet en la página de Aurélien Lemay [1], esta base de datos fue la primera que se utilizó para evaluar los algoritmos a medida que eran diseñados e implementados. Esta base de datos se describe en [DLT04], a continuación se resumen sus principales características.

La base de datos consta de dos clases de lenguajes objetivo: algunos provienen de expresiones regulares y otros de NFAs. Para la generación de autómatas a partir de expresiones regulares se considera el conjunto de operaciones

[1] `http://www.grappa.univ-lille3.fr/~lemay/`, seleccionar *Recherche* y luego *Annexes*

$Op = \{\emptyset, 0, 1, *, \cdot, +\}$. Se elige una cota superior n_{op} para el número de operadores a utilizar y se define una distribución de probabilidad p sobre Op. El operador raíz se escoge mediante la distribución p, si es un operador 0-ario la expresión termina, si es unario el procedimiento se llama recursivamente con parámetro $n_{op} - 1$ y si es binario, se llama dos veces con parámetros $\lceil n_{op}/2 \rceil$ y $\lfloor (n_{op} - 1)/2 \rfloor$. Según Denis et. al. los valores asignados a estos parámetros para generar los lenguajes objetivo de esta experimentación fueron: $n_{op} = 100$, $p_\varepsilon = 0{,}02$, $p_0 = p_1 = 0{,}05$, $p_* = 0{,}13$, $p_\cdot = 0{,}5$ y $p_+ = 0{,}25$. En cuanto a la generación de NFAs, se eligen aleatoriamente el número de estados n, el tamaño del alfabeto $|\Sigma|$, el número de transiciones que salen de cada estado n_δ y las probabilidades de ser estado inicial y final (p_I y p_F respectivamente) para cada estado. Cada estado tiene exactamente n_δ sucesores. El símbolo y el estado destino de cada transición se escogen aleatoriamente. Si se recorta el autómata algunos estados pueden quedar con menos de n_δ transiciones. Los valores de los parámetros usados para generar los autómatas objetivo fueron:: $n = 10$, $|\Sigma| = 2$, $n_\delta = 2$, $p_I = p_F = 0{,}5$.

Se generan 120 expresiones regulares (er) y 120 NFAs (nfa) que se agrupan en cuatro grupos de 30 lenguajes objetivo cada uno. Ellos sirven para etiquetar conjuntos de muestras aleatorias donde la longitud de cada muestra es menor que 30 y la longitud de los conjuntos puede ser de 50, 100, 150 o 200 muestras para entrenamiento y 1000 para prueba. La notación utilizada consiste en identificar la fuente de los lenguajes (er ó nfa) seguida por el número de muestras de entrenamiento en el correspondiente grupo, así que existen los conjuntos: er_50, er_100, er_150, er_200, nfa_50, nfa_100, nfa_150, nfa_200. Se garantiza que hay entre un 20 % y un 80 % de muestras positivas en cada conjunto. En adelante, cuando se haga referencia a esta base de datos, se le llamará el corpus de DeLeTe2.

La principal ventaja de utilizar esta base de datos es que permite compararse de manera muy precisa con el algoritmo *DeLeTe2* y también con otros algoritmos conocidos como *RPNI* y *redBlue*, ya que Denis reporta dichos resultados en [DLT04]. Otra ventaja importante es que debido a la forma como se generaron los lenguajes objetivo, esta base de datos muestra un terreno en el que los algoritmos tradicionalmente buenos no lo son tanto. Algunas desventajas de esta base de datos son que hay lenguajes objetivo repetidos y que hay muestras repetidas en los conjuntos de entrenamiento. Tampoco es particularmente bueno que las muestras no sean incrementales y que en cada grupo se entrenan diferentes lenguajes objetivo, porque esto impide comparar los resultados obtenidos con diferentes tamaños de conjuntos de entrenamiento.

Evaluación del Corpus de DeLeTe2

Con el fin de establecer una cota inferior de la posibilidad de aprendizaje que permite el corpus de DeLeTe2, se le aplicó el siguiente algoritmo de mayoría: se cuenta el número de muestras positivas y negativas en el corpus de entrenamiento, si hay más positivas que negativas, el algoritmo sólo acepta las muestras positivas y rechaza todas las demás. Si hay más muestras negativas que positivas, el algoritmo acepta el complemento de las muestras negativas y las

rechaza sólo a ellas. El Cuadro 5.1 muestra el resultado de aplicar el algoritmo de mayoría, el *RPNI* y el *DeLeTe2* al corpus de DeLeTe2. Allí se aprecia que en el caso de los NFA la información aportada por las muestras no es suficiente para que los algoritmos desplieguen su verdadero potencial de aprendizaje. Esto se nota al observar que el método de mayoría que no hace ningún esfuerzo de inferencia obtiene mejores resultados que algoritmos como *RPNI* y *DeLeTe2* (ver nfa_50). Estos resultados, obtenidos por Manuel Vázquez de Parga, sugieren que sería útil construir otro corpus más amplio, donde los algorimos puedan medirse con mayor exactitud. Sin embargo, dado que Denis ha reportado sus resultados con este corpus y que en los experimentos er_* la información aportada por las muestras parece suficiente, el corpus será utilizado como primera prueba para los algoritmos en evaluación y en todo caso se aplicarán otros conjuntos de datos con más información.

Los datos del Cuadro 5.1 también dejan al descubierto lo poco efectivo que es el algoritmo *RPNI* en este contexto, notar que en varios bloques de experimentación NFA logra tasas de acierto inferiores a las del algoritmo de mayoría.

Cuadro 5.1: Comparación de Tasas de Reconocimiento Promedio entre *MAYORÍA*, *RPNI* y *DeLeTe2* en el Corpus de DeLeTe2

	MAYORÍA	*RPNI*	*DeLeTe2*
Iden.	Tasa de Rec.	Tasa de Rec.	Tasa de Rec.
er_50	67.25 %	76.36 %	81.68 %
er_100	71.93 %	80.70 %	91.72 %
er_150	66.70 %	84.46 %	92.29 %
er_200	68.75 %	91.36 %	95.86 %
nfa_50	70.63 %	65.00 %	69.80 %
nfa_100	70.19 %	68.31 %	74.82 %
nfa_150	70.17 %	71.20 %	77.14 %
nfa_200	73.79 %	71.74 %	79.42 %

5.1.2. Corpus Utilizado para Evaluar Unambiguous Finite Automata (UFAs)

En [CF03b] Coste y Fredouille utilizan la base de datos descrita en 5.1.1 y la extienden para probar sus propios algoritmos, al hacerlo adicionan lenguajes objetivo que son UFAs y DFAs, en esta investigación se han aprovechado principalmente los DFAs. Estos lenguajes fueron generados utilizando el método de Champarnaud y Parentoën [CP05] para generar DFAs aleatorios con una distribución de probabilidad uniforme sobre todos los DFAs que tienen el mismo número de estados. Estos lenguajes se identifican en forma análoga a los de la sección 5.1.1, como ufa_50, ufa_100, ufa_150, ufa_200, dfa_50, dfa_100, dfa_150, dfa_200. Cada grupo consta de 30 lenguajes a ser aprendidos, cada muestra tiene una longitud menor de 30, los nombres ilustran el número de muestras de

entrenamiento y en todos los grupos hay 1000 muestras de prueba. En adelante, cuando se haga referencia a esta base de datos, se le llamará el corpus de los UFAs.

5.1.3. Primer Corpus Extendido

Esta experimentación aprovecha los lenguajes objetivo del Corpus de DeLeTe2. En este caso, en lugar de dividirlos en 4 grupos de 30 lenguajes, se trabaja con los 120 lenguajes en cada grupo, esto permite estudiar la convergencia del algoritmo a medida que aprende los mismos lenguajes a partir de conjuntos de muestras cada vez más grandes. Las muestras de entrenamiento se han construido de manera incremental, es decir, que cada vez que se aumenta el número de muestras, el conjunto mayor contiene al menor. Los tamaños del conjunto de entrenamiento son: 25, 50, 75, 100, 200, 300, 400 y 500 muestras. Los grupos que componen el corpus son: er_25, er_50, er_75, er_100, er_200, er_300, er_400, er_500, nfa_25, nfa_50, nfa_75, nfa_100, nfa_200, nfa_300, nfa_400, nfa_500. Cada muestra tiene una longitud máxima de 50 símbolos. Las muestras de entrenamiento y prueba son diferentes entre sí y el conjunto de prueba consta de 1000 muestras. Cuando se haga referencia a esta base de datos se le llamará Corpus Extendido1, ya que es una extensión del corpus de DeLeTe2 y habrá otra.

Este corpus tiene como ventaja el hecho de utilizar más lenguajes en cada experimento, en el corpus de DeLeTe2 eran 30 y aquí 120, también es importante que las muestras son incrementales, lo que permite hacer comparaciones y seguimiento entre los diferentes grupos de trabajo. La principal desventaja que tiene es que las muestras se generaron muy largas, de longitud máxima 50 y eso hace que el tiempo de entrenamiento para algoritmos más pesados como el DeLeTe2 se haga demasiado largo. Se considera larga la ejecución de un grupo que tarda más de 15 días de tiempo real.

5.1.4. Segundo Corpus Extendido

Esta base de datos sirve para hacer la comparación definitiva entre los algoritmos diseñados en esta investigación y los algoritmos de referencia. En ella se han tenido en cuenta las desventajas detectadas en las bases de datos anteriores. La base aprovecha de nuevo los autómatas objetivo del corpus de DeLeTe2 y los DFAs del corpus de los UFAs, al eliminar los autómatas repetidos se tiene que hay en la parte de expresiones regulares (er) 102, en nfas 120 y en dfas 119. Se generan 500 muestras de entrenamiento diferentes entre sí, que se reparten en cinco conjuntos incrementales de tamaño 100, 200, 300, 400 y 500. También se generan 1000 muestras de prueba diferentes a las de entrenamiento. La longitud de las muestras varía aleatoriamente entre cero y 18. Cada autómata etiqueta todas las muestras. Los grupos formados son entonces: er_100, er_200, er_300, er_400, er_500, nfa_100, nfa_200, nfa_300, nfa_400, nfa_500, dfa_100, dfa_200, dfa_300, dfa_400, dfa_500. No se controla el porcentaje de muestras positivas y negativas en cada conjunto de entrenamiento. Cuando se haga referencia a esta base de datos, se le llamará corpus Extendido2.

Evaluación del Corpus Extendido2

Con el fin de verificar que la información del corpus es suficiente para permitir aprendizaje, se comparan el algoritmo de mayoría y los algoritmos de referencia; en este caso se encuentra que los algoritmos de referencia logran resultados claramente superiores al de mayoría en experimentos er y nfa, lo que indica que la muestra cuenta con suficiente información para que los algoritmos apliquen sus estrategias de inferencia. Notar que en el grupo de experimentos dfa, el algoritmo *DeLeTe2* difícilmente logra obtener mejores resultados que el método de la mayoría, lo que confirma que el contexto de funcionamiento adecuado de este algoritmo es el de lenguajes regulares provenientes de expresiones regulares y autómatas no deterministas.

El Cuadro 5.2 muestra en detalle los datos obtenidos, se reportan resultados con el algoritmo de mayoría, el *RPNI* y el *DeLeTe2*.

Cuadro 5.2: Comparación de Tasas de Reconocimiento Promedio entre *MAYORÍA*, *RPNI* y *DeLeTe2* en el Corpus Extendido2.

	MAYORÍA	*RPNI*	*DeLeTe2*
Iden.	Tasa de Rec.	Tasa de Rec.	Tasa de Rec.
er_100	66.33 %	83.35 %	91.65 %
er_200	66.35 %	93.91 %	96.96 %
er_300	66.46 %	96,32 %	97.80 %
er_400	66.27 %	97.45 %	98.49 %
er_500	66.16 %	98.11 %	98.75 %
nfa_100	66.94 %	66.50 %	73.95 %
nfa_200	67.00 %	69.27 %	77.79 %
nfa_300	66.97 %	72.90 %	80.86 %
nfa_400	67.04 %	74.59 %	82.66 %
nfa_500	67.03 %	76.75 %	84.29 %
dfa_100	72.13 %	66.29 %	62.94 %
dfa_200	72.13 %	71.01 %	64.88 %
dfa_300	72.13 %	80.61 %	66.37 %
dfa_400	72.13 %	87.39 %	69.07 %
dfa_500	72.14 %	91.67 %	72.41 %

5.1.5. Corpus de DFAs Aleatorios

Los autómatas objetivo para construir este corpus se generaron aleatoriamente mediante el método de Champarnaud y Parëntoen [CP05], el cual se ha explicado con algún detalle en la Sección 2.3.1. La base de datos está dividida en tres partes: e4, e8 y e16 de acuerdo con el tamaño de los autómatas objetivo. Cada parte se divide a su vez en cuatro subpartes correspondientes a los diferentes tamaños de los conjuntos de entrenamiento: 50, 100, 150 y 200; estos valores se eligieron después de explorar varias opciones, porque en ese rango es

posible observar el comportamiento de los algoritmos en un tiempo aceptable. Cada subparte consta de 30 lenguajes a aprender, para cada uno de los cuales se generan cinco conjuntos de muestras de entrenamiento diferentes entre sí. Cada muestra tiene una longitud menor de 20 símbolos. En e16 se redujo a un conjunto de muestras por cada lenguaje ya que el tiempo de entrenamiento se hace muy largo y porque los resultados obtenidos con e4 y e8 muestran que la información adicional que aporta el entrenar cinco conjuntos es ínfima con respecto a la que se consigue entrenando un sólo conjunto. Hay 1000 muestras de prueba para cada lenguaje a entrenar.

En una segunda fase se generan 30 autómatas más, que se convierten en no deterministas reasignando aleatoriamente el símbolo asociado a cada transición. Por lo demás se tienen las mismas características que en e4, e8 y e16. En este caso se trabajó con autómatas iniciales de 10 estados por lo que la experimentación se llama e10.

5.2. Resultados Experimentales

A continuación se presentan los resultados obtenidos al ejecutar los algoritmos descritos en el Capítulo 4 sobre una o varias de las bases de datos descritas en la Sección 5.1. El propósito de estos experimentos es verificar si los algoritmos propuestos mejoran las tasas de desempeño de los algoritmos de referencia y/o disminuyen el tamaño de las hipótesis de inferencia que ellos generan.

Las variables que se miden en todos los experimentos son la tasa de reconocimiento promedio y el tamaño promedio de las hipótesis obtenidas. Cada grupo de experimentos consta de un conjunto de muestras de prueba distintas a las utilizadas en el entrenamiento, la tasa de reconocimiento promedio se calcula mediante promedio simple del número de muestras de prueba que el algoritmo clasifica correctamente. El tamaño promedio de la hipótesis se calcula sumando el número de estados de cada hipótesis obtenida y dividiendo el total entre el número de lenguajes entrenados.

En cuanto al hardware utilizado, se contó con dos posibilidades para realizar los experimentos: algunos se llevaron a cabo en un ordenador personal que tiene una procesador Pentium(R) 4 a 3.00GHz, con 1Gb de memoria RAM y 380Gb de disco duro. Sin embargo, una buena parte de los experimentos se lanzaron en un cluster del Centro de Cálculo de la Universidad Politécnica de Valencia, que es un Cluster IBM 1350, está constituido por un conjunto de 60 servidores biprocesador Intel Xeon y sistema operativo Linux RedHat, está compuesto de 57 nodos destinados a servicios de cálculo y tres nodos que realizan tareas fundamentales para el funcionamiento del cluster. Todos los algoritmos fueron implementados en lenguaje C++, los programas auxiliares se escribieron en python y bash.

Teniendo en cuenta el coste computacional de los algoritmos a ejecutar, que en todos los casos es por lo menos cúbico en función del tamaño de la muestra de entrenamiento, se sabía que las ejecuciones podrían tomar mucho tiempo;

por eso se estableció un tiempo límite para esperar los resultados. Aunque este tiempo límite ha ido variando durante el transcurso del proyecto, siempre ha sido superior a 15 días de tiempo real, aunque no siempre se ha podido garantizar la exclusividad del uso del procesador para esta tarea. En general, este tiempo de procesamiento fue suficiente para completar la experimentación, en lo casos que no fue así se hace la observación explícita al describir los resultados obtenidos.

Cada una de las siguientes subsecciones se refiere a uno o varios algoritmos estrechamente relacionados entre sí e internamente se agrupan los resultados de acuerdo a la base de datos sobre la cual se hayan realizado los experimentos; al finalizar cada subsección se hace un resumen de las conclusiones que se pueden extraer de las pruebas concernientes a cada algoritmo.

5.2.1. Mezcla Determinista, Experimentos con Algoritmos *IRPNI1* e *IRPNI2*

Los algoritmos *IRPNI1* e *IRPNI2*, presentados en las Secciones 4.2.2 y 4.2.1, fueron evaluados con el Corpus de DeLeTe2, el Corpus Extendido1 y el Corpus Extendido2. Estos tres córpora están compuestos de lenguajes provenientes de expresiones regulares y NFAs, que es el contexto en el que se busca realizar un aporte. En este caso se utilizan como algoritmos de referencia *RPNI* y *DeLeTe2*. *RPNI* es el punto de partida para el desarrollo de *IRPNI1* y *IRPNI2* y se considera una cota inferior para el desempeño de los nuevos algoritmos ya que se sabe que su desempeño en estas basas de datos no es el mejor. *DeLeTe2* se usa como referencia porque tiene el mejor desempeño conocido en este contexto y porque también utiliza relaciones de inclusión también se reportan los resultados obtenidos por *redBlue* para permitir una comparación más completa.

Experimentación con el corpus de DeLeTe2

El Cuadro 5.3 muestra los resultados obtenidos al ejecutar los algoritmos *redBlue*, *RPNI*, *IRPNI1*, *IRPNI2* y *DeLeTe2* sobre el Corpus de DeLeTe2. Notar que *RPNI* y *redBlue* generan las hipótesis más pequeñas y también las peores tasas de desempeño, esto no es de extrañar, ya que este corpus no es particularmente propicio para dichos algoritmos, como indica Denis en [DLT04].

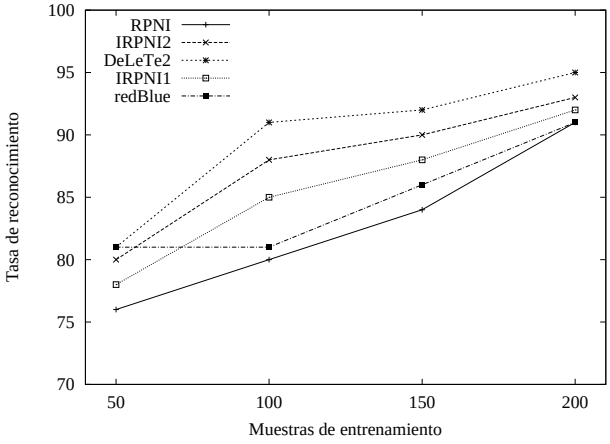
IRPNI1 y *IRPNI2* superan en varios puntos la tasa de reconocimiento de *RPNI*, aunque no llegan a superar las tasas de *DeLeTe2*. La relación es diferente si se observan los tamaños promedio de las hipótesis de inferencia obtenidas, ya que tanto *IRPNI2* como *IRPNI1* generan hipótesis más pequeñas que *DeLeTe2* y la tendencia se acentúa a medida que aumenta el número de muestras de entrenamiento. Comparando *IRPNI2* con *IRPNI1*, se nota que *IRPNI2* genera tasas de reconocimiento más altas e hipótesis de tamaño más pequeño, lo cual le hace mejor opción que *IRPNI1*.

En la Figura 5.3 partes (a) y (b) se aprecian en forma gráfica los resultados de tasas de reconocimiento, en tanto que en la Figura 5.4 partes (a) y (b) se

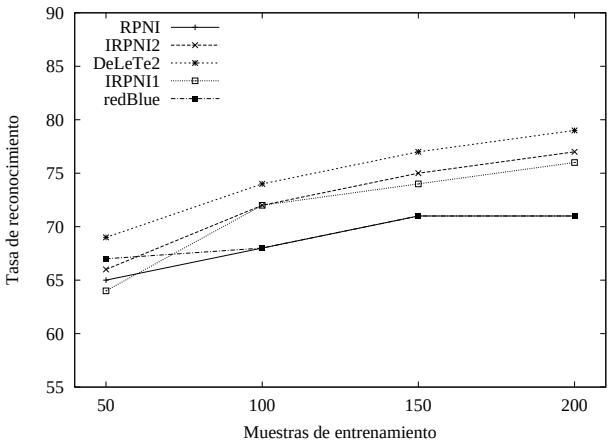
Cuadro 5.3: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *redBlue*, *RPNI*, *IRPNI2*, *IRPNI1* y *DeLeTe2* en el Corpus de DeLeTe2

Iden.	redBlue		RPNI		IRPNI2		IRPNI1		DeLeTe2	
	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.
er_50	81.36 %	8.36	76.36 %	9.63	80.03 %	20.73	78.23 %	27.53	81.68 %	32.43
er_100	81.05 %	13.73	80.70 %	14.13	88.68 %	22.56	85.47 %	36.53	91.72 %	30.73
er_150	86.35 %	14.36	84.46 %	15.43	90.61 %	29.10	88.56 %	42.36	92.29 %	60.96
er_200	91.74 %	11.73	91.36 %	13.3	93.39 %	31.10	92.34 %	40.33	95.86 %	47.73
nfa_50	67.65 %	13.10	65.00 %	14.23	66.47 %	34.90	64.72 %	43.6	69.80 %	71.26
nfa_100	68.38 %	20.43	68.31 %	21.76	72.79 %	56.13	72.11 %	67.26	74.82 %	149.13
nfa_150	71.39 %	25.86	71.20 %	28.16	75.60 %	76.63	74.57 %	90.36	77.14 %	218.26
nfa_200	71.42 %	31.46	71.74 %	33.46	77.23 %	93.26	76.39 %	104.56	79.42 %	271.3

presentan los tamaños promedio de las hipótesis generadas.



(a) Experimentos ER



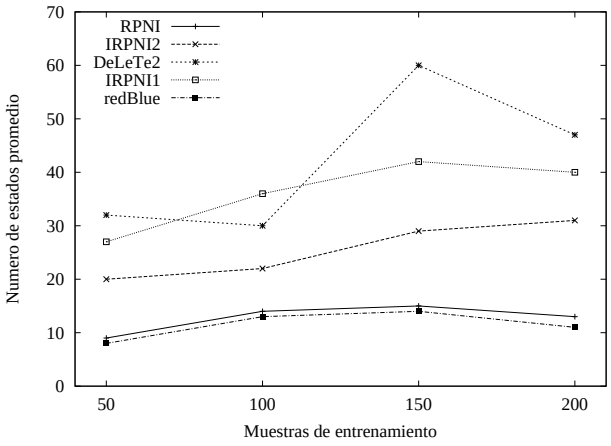
(b) Experimentos NFA

Figura 5.3: Comparación de Tasas de Reconocimiento Promedio entre *redBlue*, *RPNI*, *IRPNI2* *IRPNI1* y *DeLeTe2* en el Corpus de DeLeTe2

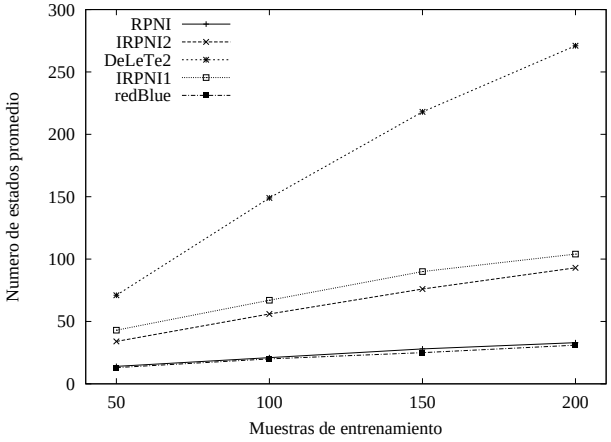
Observar que las tasas de reconocimiento de *DeLeTe2* no superan en más de tres puntos porcentuales a las de *IRPNI2* a través de toda la experimentación; que las hipótesis más grandes son las de *DeLeTe2* prácticamente en todos los bloques de experimentos, incrementándose aun más la diferencia con los demás algoritmos a medida que aumenta el número de muestras de entrenamiento.

Experimentación con el corpus Extendido1

En este caso se comparan los algoritmos *RPNI*, *IRPNI2* y *DeLeTe2* sobre los mismos lenguajes del corpus anterior, pero explorando un rango más amplio en cuanto al número de muestras de entrenamiento. El algoritmo *IRPNI1* no



(a) Experimentos ER



(b) Experimentos NFA

Figura 5.4: Comparación de Tamaño Promedio de Hipótesis entre *redBlue*, *RPNI*, *IRPNI2 IRPNI1* y *DeLeTe2* en el Corpus de DeLeTe2

se reporta ya que sus resultados en este caso fueron inferiores a los de *IRPNI2* en tasa de reconocimiento y superiores en el tamaño de las hipótesis obtenidas. La ejecución de *DeLeTe2* con 500 muestras no se completó dentro del tiempo establecido y por ese motivo no se reporta.

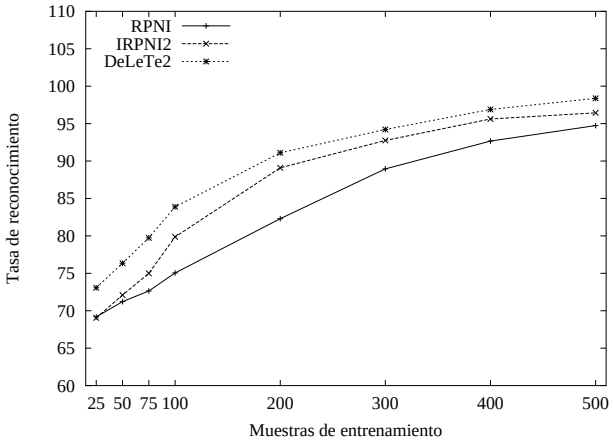
Cuadro 5.4: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *RPNI*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido1

Iden.	RPNI		IRPNI2		DeLeTe2	
	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.
er_25	69.18 %	9.01	69.05 %	20.69	73.07 %	24.08
er_50	71.22 %	13.05	72.1 %	35.59	76.35 %	63.36
er_75	72.64 %	16.35	75.00 %	48.33	79.74 %	104.49
er_100	75.05 %	18.89	79.87 %	52.55	83.88 %	132.07
er_200	82.30 %	24.64	89.10 %	58.63	91.10 %	192.35
er_300	88.95 %	23.06	92.75 %	61.07	94.21 %	234.25
er_400	92.67 %	21.45	95.62 %	51.25	96.89 %	153.95
er_500	94.74 %	19.78	96.44 %	55.12	98.38 %	89.72
nfa_25	63.52 %	10.28	64.68 %	23.16	66.49 %	28.91
nfa_50	64.87 %	15.52	65.41 %	44.30	68.01 %	85.22
nfa_75	65.44 %	20.25	67.20 %	65.08	68.41 %	144.71
nfa_100	66.22 %	24.98	66.71 %	84.61	70.45 %	231.11
nfa_200	68.56 %	40.18	72.27 %	154.38	74.72 %	594.92
nfa_300	69.94 %	54.02	75.24 %	212.96	78.50 %	947.69
nfa_400	72.23 %	64.58	77.09 %	270.79	80.27 %	1335.20
nfa_500	73.10 %	75.25	77.72 %	336.47		

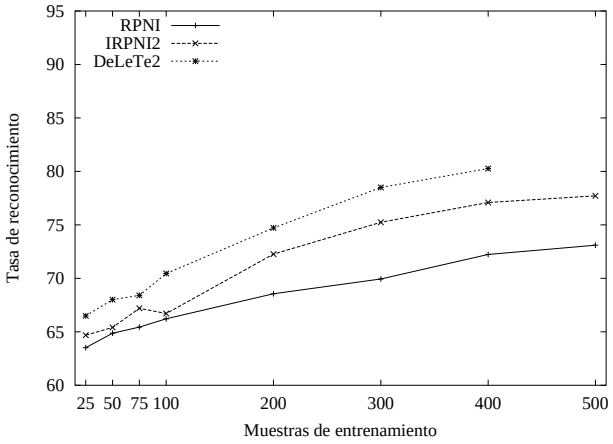
El Cuadro 5.4 presenta las porcentajes de acierto en muestras de prueba y los tamaños promedio de las hipótesis obtenidas. Esta experimentación se sobrelapa con el Corpus de DeLeTe2 en los bloques er_100, er_200, nfa_100 y nfa_200, al cruzar los resultados destaca una baja general en las tasas de acierto del corpus Extendido1 con respecto al corpus de DeLeTe2, eso se explica por la mayor longitud de las muestras de entrenamiento (pasa de 30 a 50 símbolos) y en menor medida por el hecho de entrenar 120 en lugar de 30 lenguajes por bloque. Es interesante resaltar que a medida que el tamaño del conjunto de entrenamiento aumenta, la diferencia entre las tasas de reconocimiento de *IRPNI2* y *DeLeTe2* se mantiene igual o tiende a disminuir. Estos resultados confirman las tendencias observadas anteriormente: El algoritmo *DeLeTe2* mantiene las mejores tasas de reconocimiento seguido de cerca por el algoritmo *IRPNI2*, mientras que los tamaños de las hipótesis obtenidas por el primero aumentan aceleradamente a diferencia de las hipótesis del segundo que crecen a un ritmo mucho más lento.

La Figura 5.5 ilustra las tasas de reconocimiento para los experimentos ER (parte (a)) y NFA (parte (b)). Observar que para tamaños pequeños existen algunas fluctuaciones que posteriormente se suavizan mostrando en el caso ER una tendencia a la convergencia y en el caso NFA cierta estabilidad en la rela-

ción entre los algoritmos. Es claro que los autómatas objetivo no deterministas pueden representar lenguajes más complejos que las expresiones regulares y por lo tanto requerir más muestras de entrenamiento para lograr la convergencia.



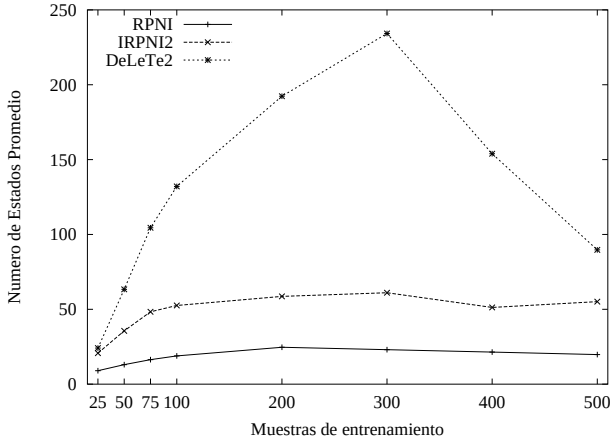
(a) Experimentos ER



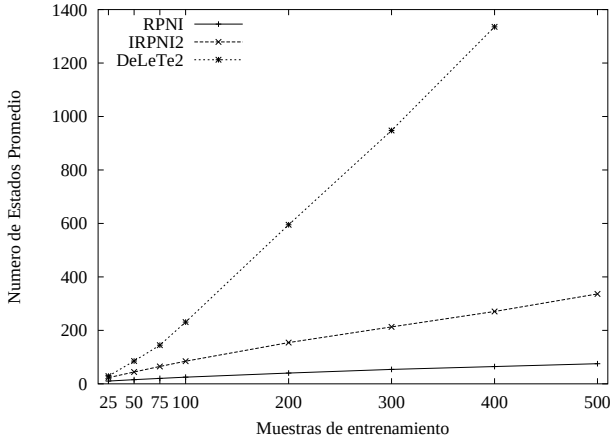
(b) Experimentos NFA

Figura 5.5: Comparación de Tasas de Reconocimiento Promedio entre *RPNI*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido1

En la Figura 5.6 se pueden comparar los tamaños promedio de las hipótesis para los experimentos ER (parte (a)) y NFA (parte (b)). Notar que en ER luego de una etapa en la que crecen los tamaños, empiezan a disminuir, esto es normal en el proceso de convergencia y es coherente con el comportamiento de las tasas de reconocimiento presentado en la Figura 5.5. En NFAs no ocurre tal disminución y destaca especialmente el desmedido tamaño de las hipótesis generadas por *DeLeTe2*.



(a) Experimentos ER



(b) Experimentos NFA

Figura 5.6: Comparación de Tamaño Promedio de Hipótesis entre *RPNI*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido1

Experimentación con el corpus Extendido2

El Cuadro 5.5 presenta los resultados obtenidos con el corpus Extendido2 para los algoritmos: *redBlue*, *RPNI*, *IRPNI2* y *DeLeTe2*.

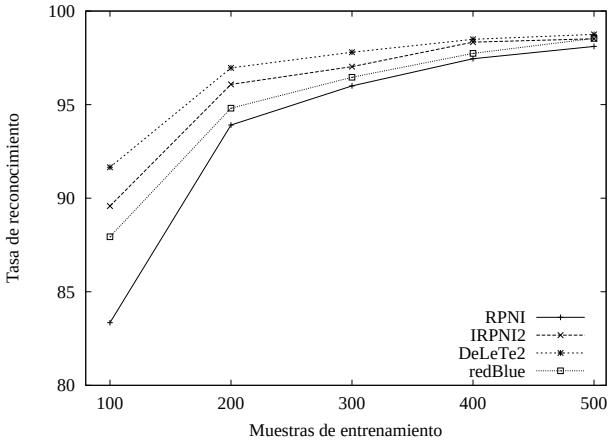
La Figura 5.7 muestra gráficamente las relaciones entre los algoritmos, notar que tanto en los experimentos ER como NFA, el algoritmo con mejores tasas es *DeLeTe2*, en cambio en los experimentos con DFAs es superado por *RPNI*. Observar también que el algoritmo *IRPNI2* se acerca progresivamente a las tasas de *DeLeTe2* en los experimentos ER y DFA, pero en los NFAs mantiene una diferencia constante de aproximadamente tres puntos porcentuales.

En cuanto al tamaño de las hipótesis, esta experimentación muestra que *DeLeTe2* siempre obtiene las hipótesis más grandes, tanto en los casos que ellas llevan a buenas tasas de reconocimiento como en los que sus tasas son malas. Destaca además que este mayor tamaño llega a ubicarse un orden de magnitud por encima de los tamaños de las hipótesis obtenidas por *IRPNI2*, tal como se aprecia en la Figura 5.8.

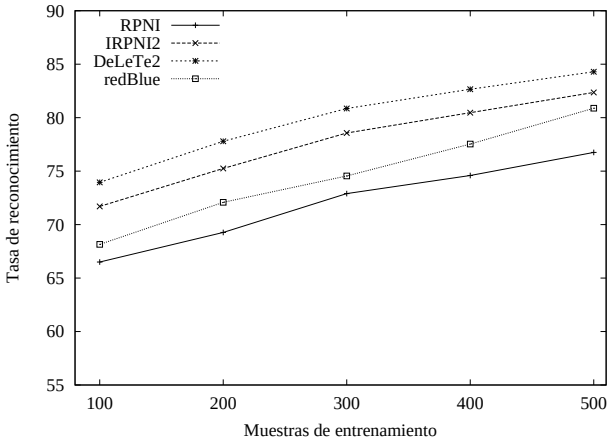
En resumen, *IRPNI2* obtiene tasas de reconocimiento cercanas a las de *DeLeTe2* mientras el tamaño de sus hipótesis es significativamente más pequeño. Este hecho hace que *IRPNI2* pueda ser un competidor válido para *DeLeTe2* cuando el tamaño de las hipótesis generadas sea un criterio sensible.

Cuadro 5.5: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *redBlue*, *RPNI*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2

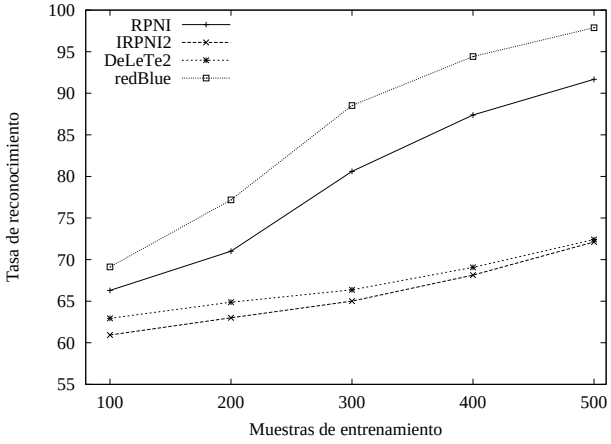
Iden.	*redBlue*		*RPNI*		*IRPNI2*		*DeLeTe2*	
	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom	Tasa de Rec.	Tamaño Prom
er_100	87.94 %	10	83.35 %	12.39	89.58 %	19.00	91.65 %	30.23
er_200	94.81 %	9.97	93.91 %	11.52	96.08 %	16.92	96.96 %	24.48
er_300	96.46 %	11.05	96,32 %	11.17	97.03 %	18.12	97.80 %	31.41
er_400	97.74 %	10.43	97.45 %	10.98	98.34 %	15.37	98.49 %	27.40
er_500	98.54 %	10.47	98.11 %	10.95	98.52 %	16.75	98.75 %	29.85
nfa_100	68.15 %	18.83	66.50 %	20.31	71.70 %	43.49	73.95 %	98.80
nfa_200	72.08 %	28.80	69.27 %	32.35	75.25 %	72.36	77.79 %	220.93
nfa_300	74.55 %	36.45	72.90 %	40.86	78.57 %	94.04	80.86 %	322.13
nfa_400	77.53 %	42.58	74.59 %	49.75	80.47 %	109.93	82.66 %	421.30
nfa_500	80.88 %	47.54	76.75 %	55.91	82.36 %	125.75	84.29 %	512.55
dfa_100	69.12 %	18.59	66.29 %	20.27	60.94 %	56.52	62.94 %	156.89
dfa_200	77.18 %	25.83	71.01 %	31.11	63.00 %	102.32	64.88 %	432.88
dfa_300	88.53 %	25.10	80.61 %	33.33	65.01 %	134.89	66.37 %	706.64
dfa_400	94.42 %	21.36	87.39 %	31.90	68.14 %	156.63	69.07 %	903.32
dfa_500	97.88 %	18.75	91.67 %	29.61	72.12 %	169.92	72.41 %	1027.42



(a) Experimentos ER

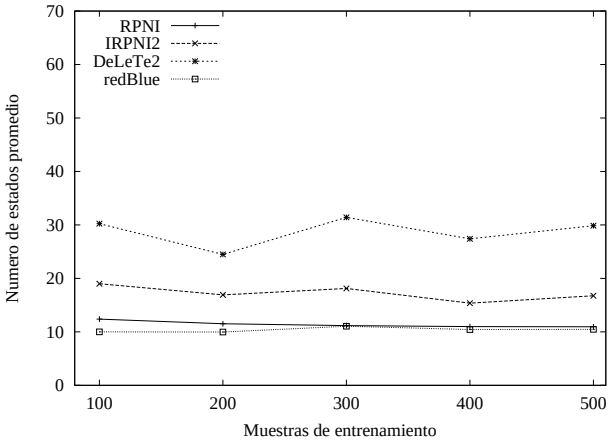


(b) Experimentos NFA

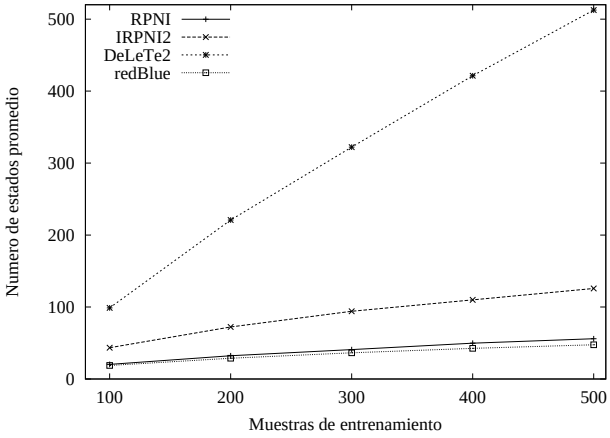


(c) Experimentos DFA

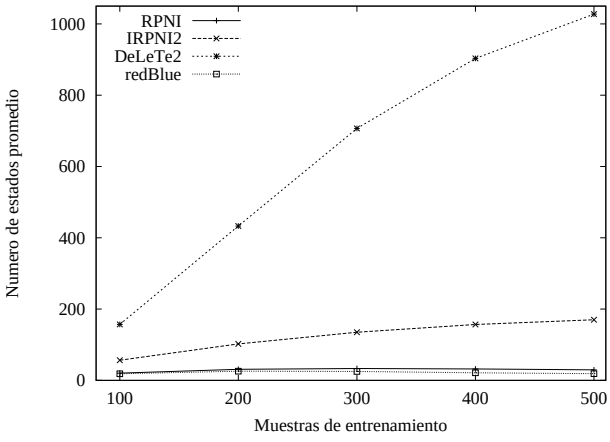
Figura 5.7: Comparación de Tasas de Reconocimiento Promedio entre *redBlue*, *RPNI*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2



(a) Experimentos ER



(b) Experimentos NFA



(c) Experimentos DFA

Figura 5.8: Comparación de Tamaño Promedio de Hipótesis entre *redBlue*, *RPNI*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2

5.2.2. Mezcla Determinista, Experimentos con Algoritmo *RBIR*

A continuación se reportan los resultados obtenidos con dos variaciones del algoritmo *RBIR*, ellas se identifican como *RBIRX* y *RBIRZ* (ver Sección 4.2.3). Los algoritmos de referencia en este caso son *redBlue* por ser la base para los algoritmos propuestos e *IRPNI2* por ser el mejor algoritmo desarrollado hasta el momento en esta investigación y porque utiliza también información obtenida a partir de las relaciones de inclusión entre lenguajes residuales. También se presentan los resultados obtenidos por *DeLeTe2*. Dado que los algoritmos *RBIR** tienen no determinismo, cada experimento se ha realizado 10 veces, al calcular la desviación estándar se observa que en general la desviación es pequeña: inferior a un punto porcentual en un alto porcentaje de los casos y que disminuye progresivamente a medida que aumenta el número de muestras de entrenamiento, como es de esperar en un algoritmo convergente. Debido a que las desviaciones entandar no aportan en este caso información nueva, no se detallan en los cuadros de datos ni se dibujan en los gráficos. En algunos bloques de experimentación DFA, se reporta únicamente una ejecución (no 10) porque los primeros resultados confirmaron que los resultados serían muy pobres y se prefirió dar prioridad a otros experimentos más prometedores, tales bloques están marcados con un asterisco (*) en el Cuadro 5.6, el cual presenta los resultados obtenidos sobre el corpus Extendido2.

La Figura 5.9 ilustra los resultados obtenidos en cuanto a tasa de reconocimiento. Vale la pena notar lo cercanos que son los resultados de ambas versiones *RBIRX* y *RBIRZ* con respecto al algoritmo *IRPNI2* en experimentos ER y NFA. Ambas versiones mejoran los resultados de *redBlue*, lo cual se aprecia particularmente en la experimentación NFA; en los experimentos DFA, los resultados son claramente inferiores con respecto a los del algoritmo *redBlue*. En cuanto a los tamaños de las hipótesis obtenidas, las versiones *RBIR* producen hipótesis más grandes que las de *IRPNI2*, pero sin llegar a los tamaños que produce *DeLeTe2*.

Comparando entre sí las versiones *RBIRX* y *RBIRZ*, la primera tiene menor complejidad temporal que la segunda, sus tasas de reconocimiento presentan diferencias muy pequeñas, de menos de un punto porcentual y el tamaño de las hipótesis tiende a ser inferior en el primero. En esas condiciones puede ser más adecuado *RBIRX* dado que es más rápido de ejecutar y produce hipótesis de menor tamaño.

En resumen, parece que modificar *redBlue* para que tome en consideración la información obtenida a partir de las relaciones de inclusión entre estados conduce al mismo nivel de tasas de reconocimiento que se consigue al modificar el algoritmo *RPNI* para que tenga en cuenta dicha información. Sin embargo, al comparar las versiones que se han desarrollado, ocurre que las hipótesis obtenidas a partir de *IRPNI2* son unos pocos estados más pequeñas que las que se consiguen con las versiones *RBIR*, por lo que *IRPNI2* puede ser una mejor opción.

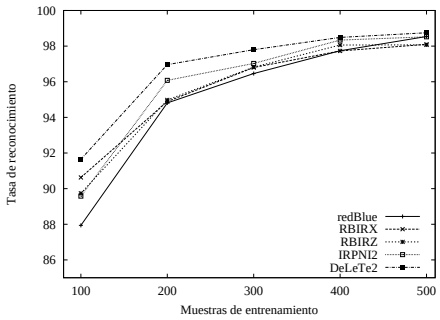
La Figura 5.10 muestra cómo las hipótesis generadas por las versiones de

Cuadro 5.6: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *redBlue*, *IRPNI2*, *RBIRX*, *RBIRZ* y *DeLeTe2* en el Corpus Extendido2

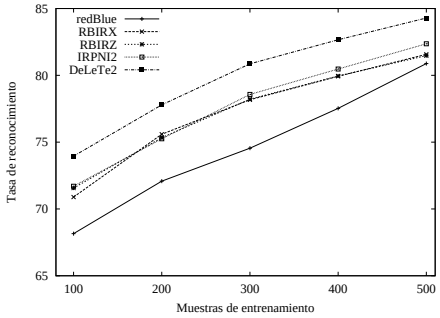
Iden.	redBlue		RBIRX		RBIRZ		IRPNI2		DeLeTe2	
	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.
er_100	87.94 %	10	90.63 %	17.40	89.74 %	19.62	89.58 %	19.00	91.65 %	30.23
er_200	94.81 %	9.97	94.89 %	18.87	94.98 %	19.54	96.08 %	16.92	96.96 %	24.48
er_300	96.46 %	11.05	96.80 %	19.54	96.82 %	19.67	97.03 %	18.12	97.80 %	31.41
er_400	97.74 %	10.43	97.73 %	17.68	98.06 %	17.84	98.34 %	15.37	98.49 %	27.40
er_500	98.54 %	10.47	98.10 %	19.59	98.07 %	20.81	98.52 %	16.75	98.75 %	29.85
nfa_100	68.15 %	18.83	70.89 %	39.60	71.56 %	43.34	71.70 %	43.49	73.95 %	98.80
nfa_200	72.08 %	28.80	75.59 %	66.17	75.34 %	71.90	75.25 %	72.36	77.79 %	220.93
nfa_300	74.55 %	36.45	78.17 %	89.45	78.33 %	89.16	78.57 %	94.04	80.86 %	322.13
nfa_400	77.53 %	42.58	79.93 %	107.01	79.77 %	115.46	80.47 %	109.93	82.66 %	421.30
nfa_500	80.88 %	47.54	81.57 %	122.66	81.47 %	132.35	82.36 %	125.75	84.29 %	512.55
dfa_100	69.12 %	18.59	61.99 %	51.03	55.63 % (*)	56.23	60.94 %	56.52	62.94 %	156.89
dfa_200	77.18 %	25.83	62.42 %	96.28	50.53 % (*)	128.82	63.00 %	102.32	64.88 %	432.88
dfa_300	88.53 %	25.10	63.00 %	134.93	53.74 % (*)	158.17	65.01 %	134.89	66.37 %	706.64
dfa_400	94.42 %	21.36	63.59 %	169.29	57.65 % (*)	192.47	68.14 %	156.63	69.07 %	903.32
dfa_500	97.88 %	18.75	64.19 %	204.60	57.17 % (*)	227.13	72.12 %	169.92	72.41 %	1027.42

RBIR son ligeramente más grandes que las de *IRPNI2* en los tres tipos de experimentos: ER, NFA y DFA. Comparando *RBIRX* con *RBIRZ*, la primera versión genera hipótesis un poco más pequeñas que la segunda.

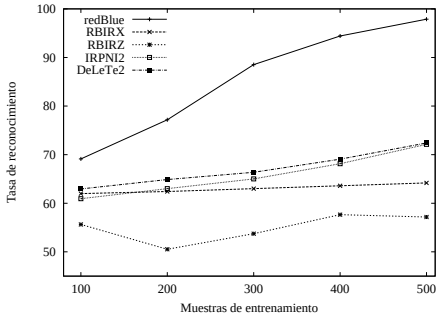
Dado que los tamaños de las hipótesis producidas por *DeLeTe2* obligan a graficar un rango muy grande en el eje Y, las diferencias entre los demás algoritmos no logran apreciarse con claridad en la Figura 5.10, particularmente en las partes (b) y (c), por eso en la Figura 5.11 se omiten los resultados de *DeLeTe2* y se ajusta la escala del gráfico para apreciar mejor el desempeño de los algoritmos restantes.



(a) Experimentos ER

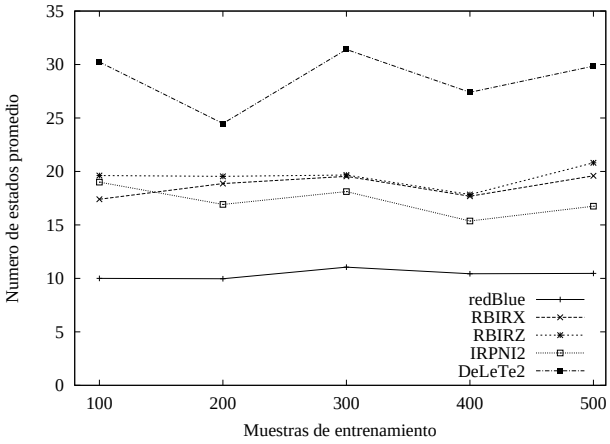


(b) Experimentos NFA

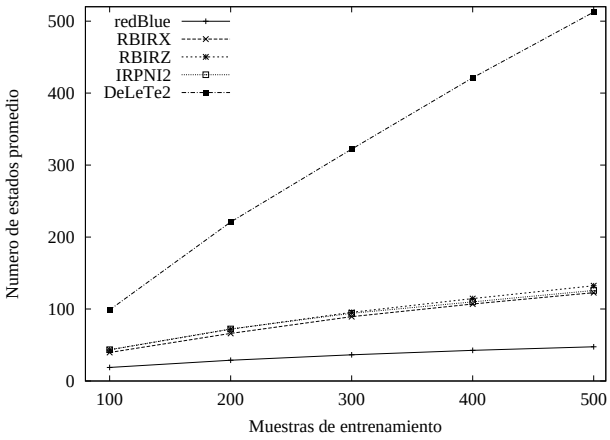


(c) Experimentos DFA

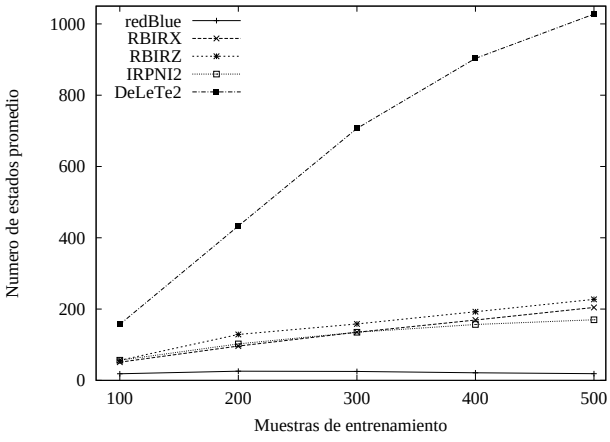
Figura 5.9: Comparación de Tasas de Reconocimiento Promedio entre *redBlue*, *RBIRX*, *RBIRZ*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2



(a) Experimentos ER

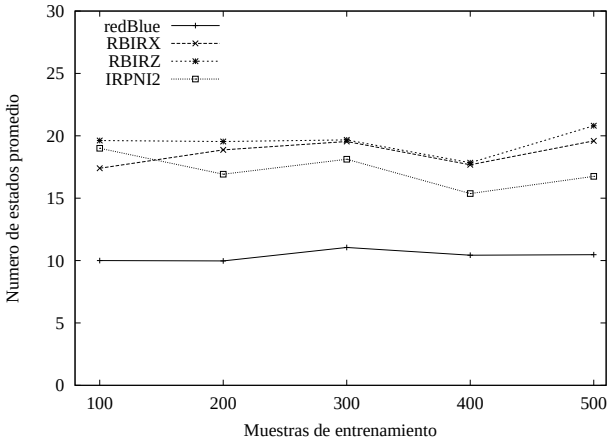


(b) Experimentos NFA

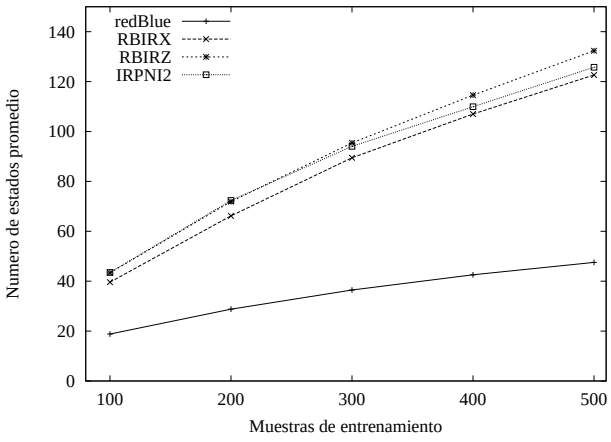


(c) Experimentos DFA

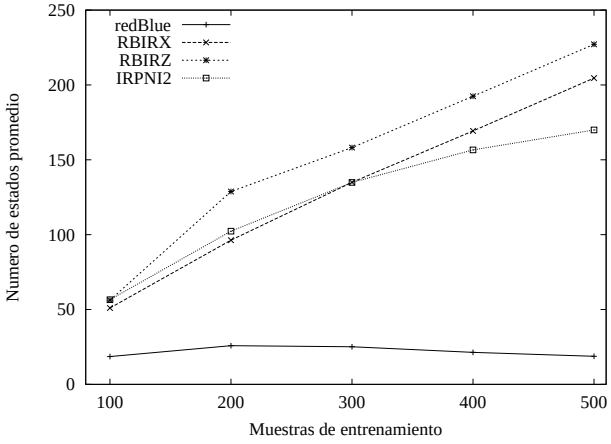
Figura 5.10: Comparación de Tamaño Promedio de Hipótesis entre *redBlue*, *RBIRX*, *RBIRZ*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2



(a) Experimentos ER



(b) Experimentos NFA



(c) Experimentos DFA

Figura 5.11: Comparación de Tamaño Promedio de Hipótesis entre *redBlue*, *RBIRX*, *RBIRZ* e *IRPNI2* en el Corpus Extendido2

5.2.3. Mezcla Determinista, Experimentos con Algoritmo *RRB*

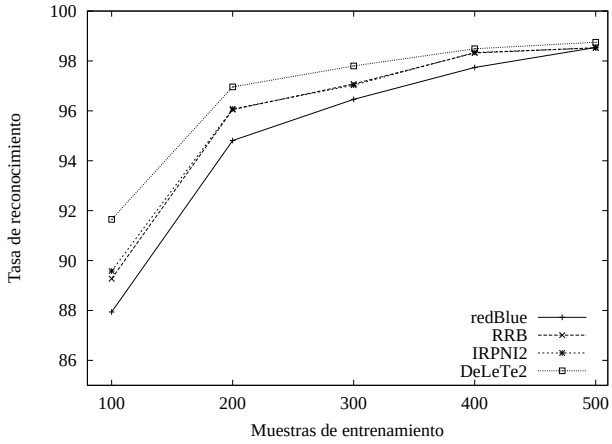
Esta experimentación se llevó a cabo sobre el corpus Extendido2, y compara el algoritmo *RRB*, descrito en la Sección 4.2.4 con los algoritmos de referencia *IRPNI2* y *redBlue*. También se reportan los resultados conseguidos con *DeLeTe2* para facilitar una comparación más completa.

La Figura 5.12 muestra cómo el comportamiento del algoritmo *RRB* supera en unas cuantas décimas de punto porcentual al de *IRPNI2* en las tres experimentaciones. Por su parte, *redBlue* es superado por *IRPNI2* y *RRB* en los experimentos ER y NFA, mientras que es el ganador en los experimentos DFA. Esta condición ya es conocida y simplemente viene a confirmarse a través de estos resultados. En cuanto a los tamaños de las hipótesis (ver Figura 5.13), *RRB* las genera ligeramente más pequeñas que *IRPNI2*. La Figura 5.14 omite los datos de *DeLeTe2* para permitir visualizar en un rango más reducido los resultados de los demás algoritmos.

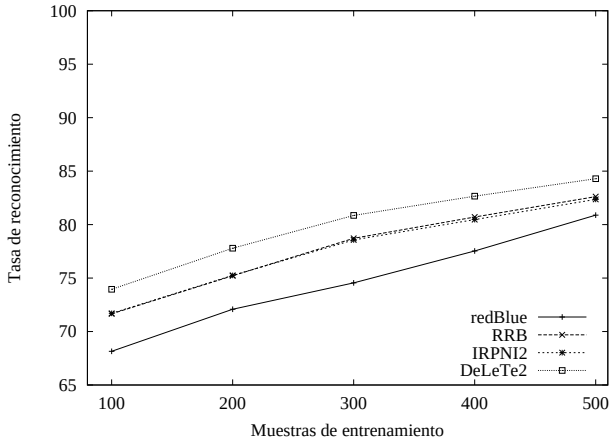
En conclusión, la estrategia *RRB* que utiliza los algoritmos *IRPNI2* y *redBlue* logra una mejora leve en tasa de reconocimiento y también una ligera disminución en el tamaño de las hipótesis obtenidas. Esto unido a que tiene un comportamiento convergente hace que sea una estrategia perfectamente viable para su aplicación. Hay que considerar, sin embargo, que *IRPNI2* puede obtener resultados muy similares realizando menos trabajo, ya que al utilizarlo no es necesario etiquetar todos los prefijos de las muestras ni se debe ejecutar *redBlue*, por lo que en determinados contextos pueden no ser significativas las ventajas obtenidas en tasa y tamaño con respecto al trabajo adicional que se debe realizar.

Cuadro 5.7: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *redBlue*, *RRB*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2

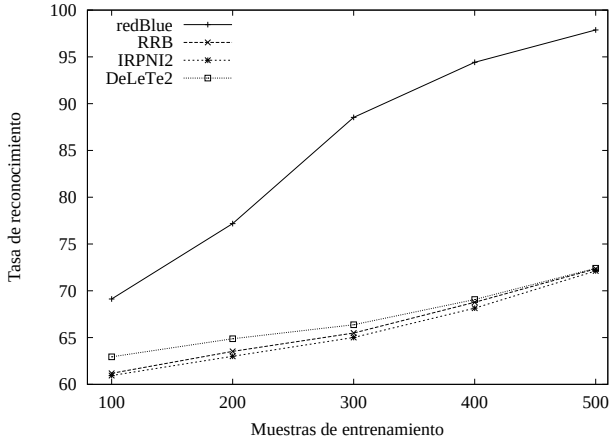
Iden.	*redBlue*		*RRB*		*IRPNI2*		*DeLeTe2*	
	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.
er_100	87.94 %	10	89.27 %	16.86	89.58 %	19.00	91.65 %	30.23
er_200	94.81 %	9.97	96.05 %	16.99	96.08 %	16.92	96.96 %	24.48
er_300	96.46 %	11.05	97.08 %	18.00	97.03 %	18.12	97.80 %	31.41
er_400	97.74 %	10.43	98.32 %	15.07	98.34 %	15.37	98.49 %	27.40
er_500	98.54 %	10.47	98.53 %	16.61	98.52 %	16.75	98.75 %	29.85
nfa_100	68.15 %	18.83	71.66 %	43.40	71.70 %	43.49	73.95 %	98.80
nfa_200	72.08 %	28.8	75.22 %	71.27	75.25 %	72.36	77.79 %	220.93
nfa_300	74.55 %	36.45	78.72 %	93.45	78.57 %	94.04	80.86 %	322.13
nfa_400	77.53 %	42.58	80.71 %	107.7	80.47 %	109.93	82.66 %	421.30
nfa_500	80.88 %	47.54	82.62 %	123.02	82.36 %	125.75	84.29 %	512.55
dfa_100	69.12 %	18.59	61.18 %	56.36	60.94 %	56.52	62.94 %	156.89
dfa_200	77.18 %	25.83	63.52 %	98.81	63.00 %	102.32	64.88 %	432.88
dfa_300	88.53 %	25.10	65.49 %	128.97	65.01 %	134.89	66.37 %	706.64
dfa_400	94.42 %	21.36	68.77 %	149.42	68.14 %	156.63	69.07 %	903.32
dfa_500	97.88 %	18.75	72.34 %	161.57	72.12 %	169.92	72.41 %	1027.42



(a) Experimentos ER

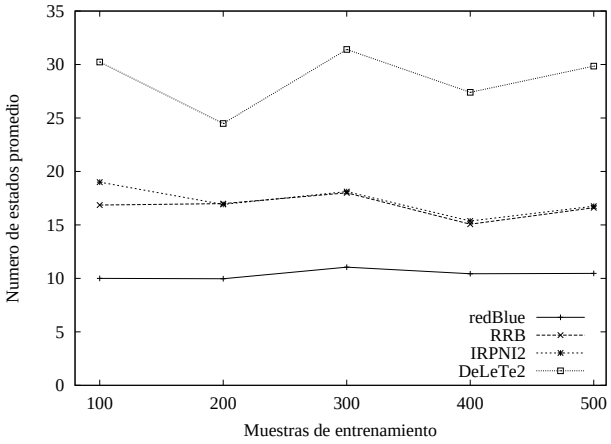


(b) Experimentos NFA

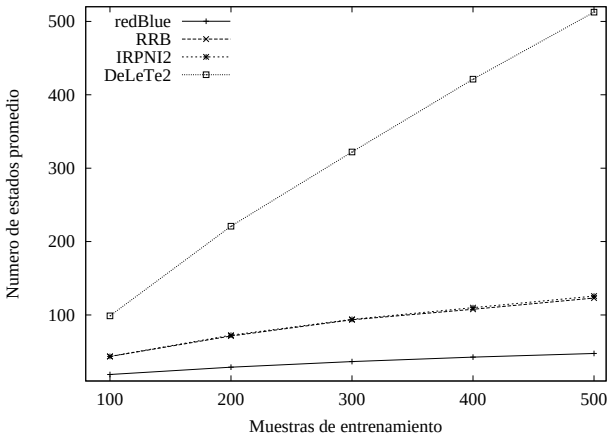


(c) Experimentos DFA

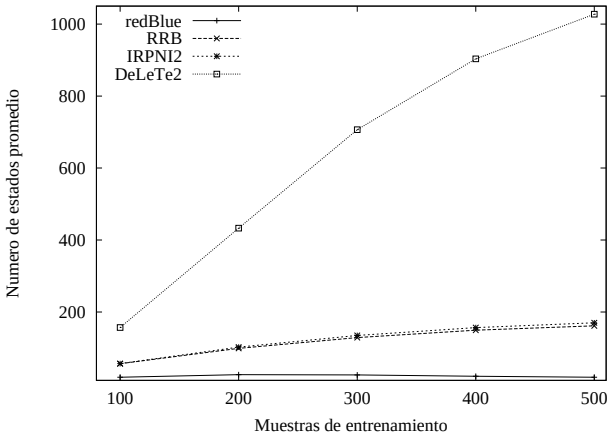
Figura 5.12: Comparación de Tasas de Reconocimiento Promedio entre *redBlue*, *RRB*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2



(a) Experimentos ER

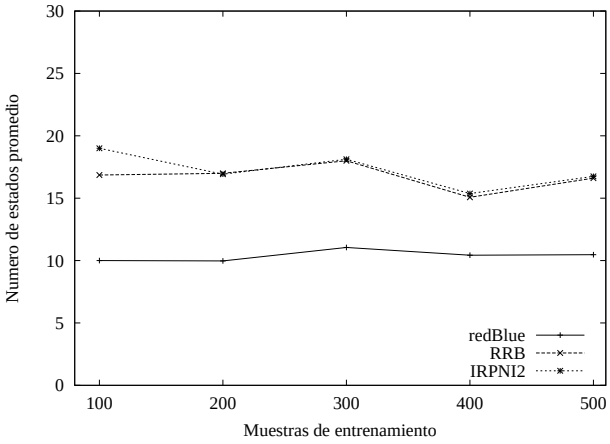


(b) Experimentos NFA

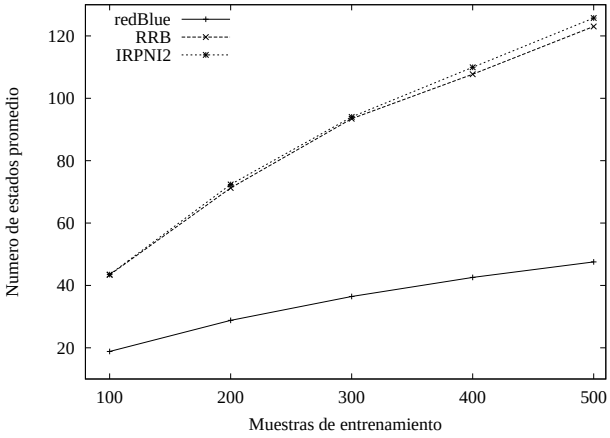


(c) Experimentos DFA

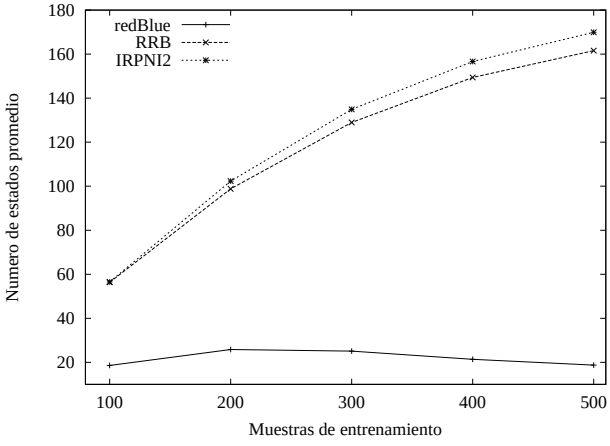
Figura 5.13: Comparación de Tamaño Promedio de Hipótesis entre *redBlue*, *RRB*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2



(a) Experimentos ER



(b) Experimentos NFA



(c) Experimentos DFA

Figura 5.14: Comparación de Tamaño Promedio de Hipótesis entre *redBlue*, *RRB* e *IRPNI2* en el Corpus Extendido2

5.2.4. Mezcla No Determinista, Experimentos con Algoritmo *NRPNI*

El algoritmo *NRPNI* es un procesamiento que se ejecuta sobre el resultado de otro algoritmo de inferencia, que en este caso puede ser *RPNI*, *IRPNI2* o *IRPNI1*. Al resultado de ejecutar *NRPNI* a partir del resultado de *RPNI*, se le llama *NRPNI*. Si el punto de partida es el resultado de *IRPNI2* (resp. *IRPNI1*), el algoritmo resultante se llama *NRPNI2* (resp. *NRPNI1*).

Experimentación con el Corpus de DeLeTe2

El Cuadro 5.8 muestra las cifras en cuanto a tasa de reconocimiento y tamaño de los autómatas de los algoritmos *NRPNI*, *NRPNI2*, *NRPNI1*, *IRPNI2* y *DeLeTe2* ejecutados sobre el corpus de DeLeTe2.

En la Figura 5.15 se aprecia cómo todas las versiones no deterministas *NRPNI* tienen un comportamiento similar en cuanto a tasa de reconocimiento, siendo el *NRPNI2* la versión que obtiene las tasas más altas de los tres. En la experimentación ER, el *NRPNI2* iguala y aun supera levemente a *IRPNI2*, pero en la experimentación NFA es *IRPNI2* el claro ganador. Como ya se ha notado en otros experimentos, se aprecia cierta tendencia a la convergencia de los algoritmos en ER, cosa que no se aprecia en NFA. En cuanto a los tamaños de las hipótesis, la Figura 5.16 muestra un comportamiento similar en las tres versiones NRPNI, siendo en todo caso *NRPNI1* quien genera las hipótesis más grandes. En la Figura 5.17 se muestran de nuevo los resultados de tamaños de hipótesis obtenidos, pero excluyendo los de *DeLeTe2* para poder ver mejor la relación entre el comportamiento de los demás algoritmos.

Resultados con el Corpus Extendido2

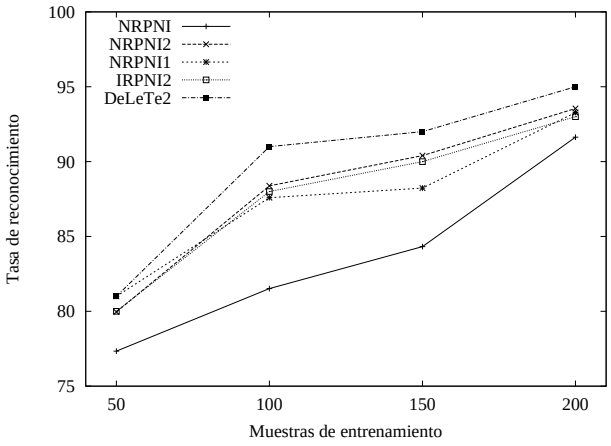
De la experimentación con el corpus de DeLeTe2 se desprende que el algoritmo *NRPNI1* tiene el desempeño más pobre de las tres alternativas: *NRPNI*, *NRPNI2* y *NRPNI1*, por eso no se ha tomado en cuenta para esta experimentación. El Cuadro 5.9 presenta los resultados obtenidos.

La Figura 5.18 muestra cómo en el caso de los experimentos ER los algoritmos *NRPNI* y *NRPNI2* logran igualar y superar al *IRPNI2* en cuanto a tasa de acierto; en el último bloque de pruebas, con 500 muestras de entrenamiento, el comportamiento de los tres algoritmos es muy cercano. Notar que *NRPNI2* comienza con una ventaja de aproximadamente un punto porcentual con respecto a *IRPNI2*, ventaja que va decreciendo a medida que aumentan el número de muestras de entrenamiento. En los experimentos NFA se puede ver como *NRPNI* tiene un desempeño inferior a *IRPNI2* y *NRPNI2* siendo estos dos muy similares entre sí. Sin embargo, en los experimentos DFA *NRPNI* supera claramente a *IRPNI2* que a su vez supera a *NRPNI2*.

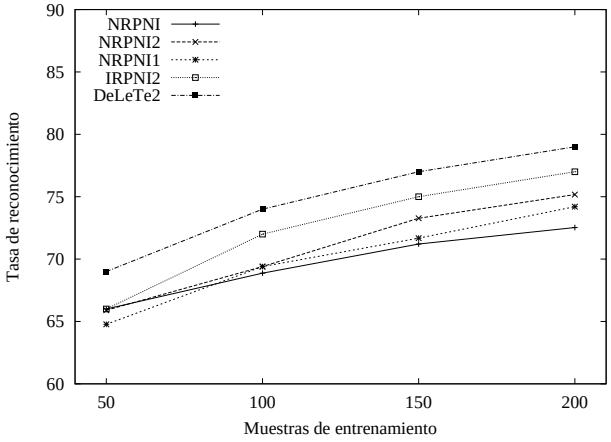
En cuanto al tamaño de las hipótesis, la Figura 5.19 muestra cómo en el caso ER *IRPNI2* genera las hipótesis más grandes, sin embargo la diferencia entre

Cuadro 5.8: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *NRPNI*, *NRPNI2*, *NRPNI1*, *IRPNI2* y *DeLeTe2* en el Corpus de DeLeTe2

Iden. Iden.	*NRPNI*		*NRPNI2*		*NRPNI1*		*IRPNI2*		*DeLeTe2*	
	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.	Tasa Rec.	Tam. Prom.
er_50	77.34 %	8.93	79.96 %	16.93	81.00 %	17.63	80.03 %	20.73	81.68 %	32.43
er_100	81.52 %	13.6	88.38 %	19.03	87.59 %	23.2	88.68 %	22.56	91.72 %	30.73
er_150	84.32 %	14.9	90.40 %	25.26	88.23 %	33.73	90.61 %	29.10	92.29 %	60.96
er_200	91.63 %	12.8	93.55 %	27.36	93.28 %	32.93	93.39 %	31.10	95.86 %	47.73
nfa_50	66.00 %	13.5	65.90 %	29.6	64.77 %	34.4	66.47 %	34.90	69.80 %	71.26
nfa_100	68.87 %	21.2	69.40 %	52.13	69.40 %	58.33	72.79 %	56.13	74.82 %	149.13
nfa_150	71.21 %	27.66	73.27 %	71.56	71.68 %	80.96	75.60 %	76.63	77.14 %	218.26
nfa_200	72.53 %	32.56	75.17 %	88.63	74.20 %	95.7	77.23 %	93.26	79.42 %	271.3

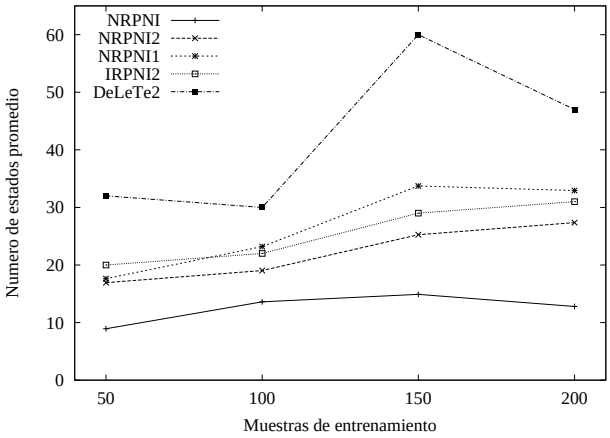


(a) Experimentos ER

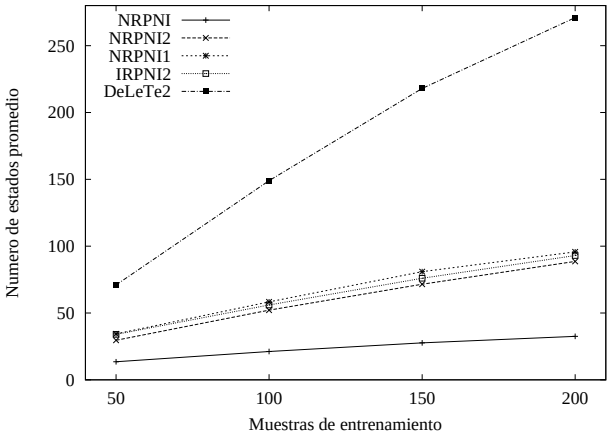


(b) Experimentos NFA

Figura 5.15: Comparación de Tasas de Reconocimiento Promedio entre *NRPNI*, *NRPNI2*, *NRPNI1*, *IRPNI2* y *DeLeTe2* en el Corpus de DeLeTe2

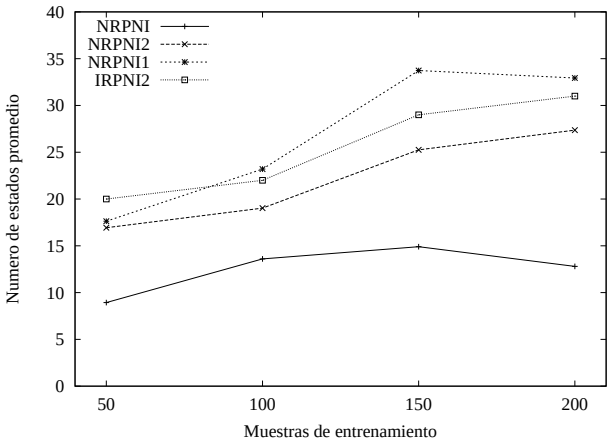


(a) Experimentos ER

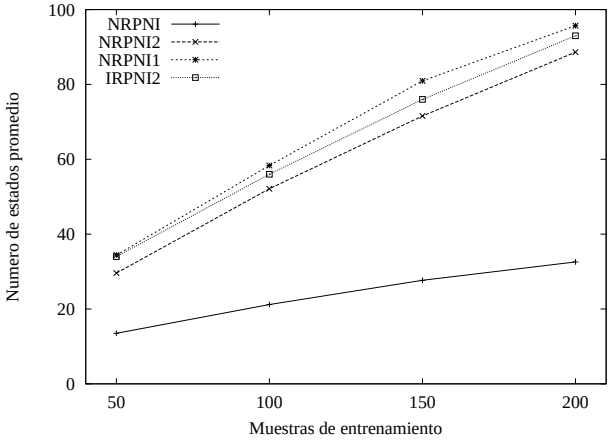


(b) Experimentos NFA

Figura 5.16: Comparación de Tamaño Promedio de Hipótesis entre *NRPNI*, *NRPNI2*, *NRPNI1*, *IRPNI2* y *DeLeTe2* en el Corpus de DeLeTe2



(a) Experimentos ER

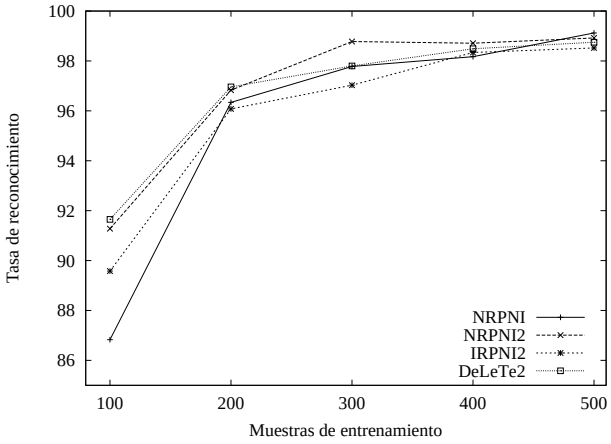


(b) Experimentos NFA

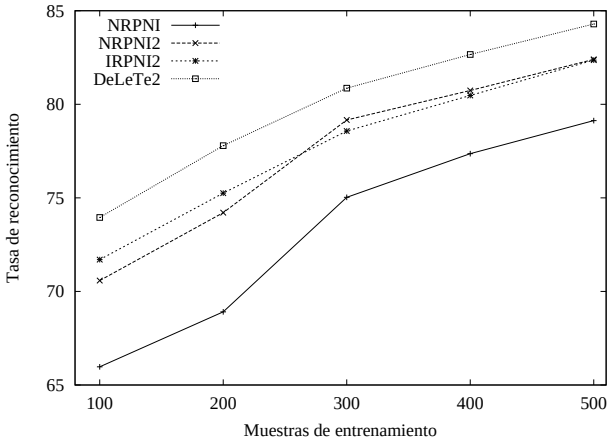
Figura 5.17: Comparación de Tamaño Promedio de Hipótesis entre *NRPNI*, *NRPNI2*, *NRPNI1* y *IRPNI2* en el Corpus de DeLeTe2

Cuadro 5.9: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *NRPNI*, *NRPNI2*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2

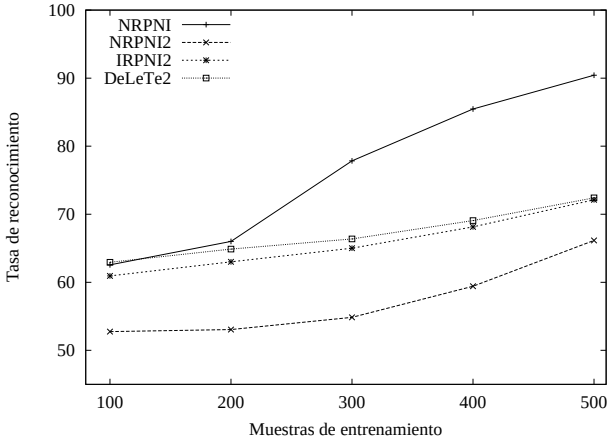
Iden.	*NRPNI*		*NRPNI2*		*IRPNI2*		*DeLeTe2*	
	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom	Tasa de Rec.	Tamaño Prom
er_100	86.83 %	10.4	91.28 %	16.13	89.58 %	19.00	91.65 %	30.23
er_200	96.34 %	8.43	96.83 %	12.23	96.08 %	16.92	96.96 %	24.48
er_300	97.78 %	8.53	98.78 %	9.06	97.03 %	18.12	97.80 %	31.41
er_400	98.17 %	8.93	98.71 %	9.86	98.34 %	15.37	98.49 %	27.40
er_500	99.12 %	8.13	98.92 %	10.8	98.52 %	16.75	98.75 %	29.85
nfa_100	65.97 %	20.36	70.58 %	37.33	71.70 %	43.49	73.95 %	98.80
nfa_200	68.91 %	31.6	74.21 %	63.53	75.25 %	72.36	77.79 %	220.93
nfa_300	75.03 %	39.3	79.16 %	83.26	78.57 %	94.04	80.86 %	322.13
nfa_400	77.36 %	47.2	80.74 %	95.83	80.47 %	109.93	82.66 %	421.30
nfa_500	79.13 %	54.92	82.4 %	120.7	82.36 %	125.75	84.29 %	512.55
dfa_100	62.56 %	20.63	52.75 %	52.5	60.94 %	56.52	62.94 %	156.89
dfa_200	66.00 %	31.9	53.05 %	102.06	63.00 %	102.32	64.88 %	432.88
dfa_300	77.84 %	34.7	54.85 %	135.06	65.01 %	134.89	66.37 %	706.64
dfa_400	85.47 %	31.86	59.42 %	159.13	68.14 %	156.63	69.07 %	903.32
dfa_500	90.43 %	30.03	66.15 %	166.76	72.12 %	169.92	72.41 %	1027.42



(a) Experimentos ER

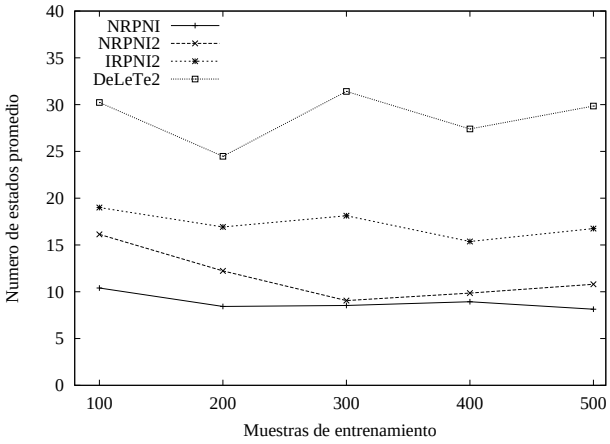


(b) Experimentos NFA

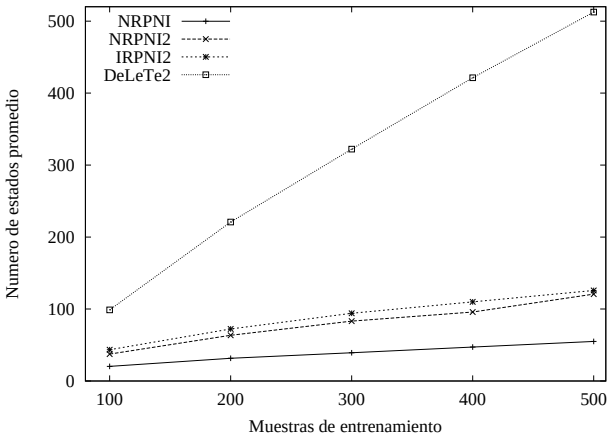


(c) Experimentos DFA

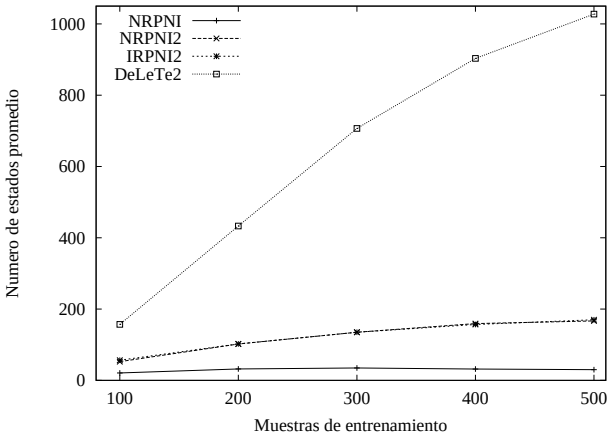
Figura 5.18: Comparación de Tasas de Reconocimiento Promedio entre *NRPNI*, *NRPNI2*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2



(a) Experimentos ER



(b) Experimentos NFA



(c) Experimentos DFA

Figura 5.19: Comparación de Tamaño Promedio de Hipótesis entre *NRPNI*, *NRPNI2*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2

los tamaños promedio se reduce a menos de 10 estados que no es una diferencia que conduzca a descartar ninguna de las opciones. En los experimentos NFA y DFA es clarísima la ventaja de las hipótesis producidas por *NRPNI* por ser notoriamente más pequeñas que las producidas por los otros métodos.

Resumiendo, aunque los primeros experimentos muestran mejores resultados en tasa de acierto para *NRPNI2*, a medida que aumentan las muestras de entrenamiento su desempeño va siendo cada vez más parecido al de *IRPNI2*. En cuanto a *NRPNI*, sus resultados en experimentos ER son muy parecidos a los de los otros algoritmos, en NFA se queda unos cuatro puntos porcentuales por debajo de ellos y en DFA los supera en casi 20 puntos porcentuales. En un contexto mixto, donde se deban inferir lenguajes regulares que pueden tener muchas o pocas relaciones de inclusión entre estados, *NRPNI* puede ser una alternativa a considerar. En cuanto a los tamaños de las hipótesis, *NRPNI2* produce hipótesis más pequeñas que *IRPNI2* en todos los experimentos del corpus. Sin embargo, dados a escoger entre el algoritmo *IRPNI2* y el *NRPNI2*, el primero sigue siendo de mayor interés dado que no requiere ejecutar el código de *NRPNI* y obtiene resultados bastante semejantes. En cuanto a *NRPNI* sus hipótesis son las más pequeñas a lo largo de esta experimentación.

5.2.5. Mezcla No Determinista, Experimentos con Algoritmo *MRIA*

Tanto *DeLeTe2* como *MRIA* buscan la inferencia de RFSAs, por eso, ambos utilizan intensivamente la relación de inclusión entre estados. *DeLeTe2* hace énfasis en detectar la mayor cantidad posible de relaciones de inclusión, valiéndose para ello de la propiedad transitiva, ello permite deducir nuevas inclusiones entre parejas de estados a partir de las relaciones conocidas de cada uno de ellos con otros terceros. Por su parte, *MRIA* supone que entre un par de estados se da la relación de inclusión y posteriormente verifica si las muestras de entrenamiento corroboran o contradicen tal suposición. Esta diferencia de enfoque explica porqué el algoritmo *MRIA* es mucho más simple que *DeLeTe2* y por lo tanto tiene menor complejidad temporal y espacial que éste.

Resultados con el Corpus de DeLeTe2

El Cuadro 5.10 muestra los resultados de comparar *MRIA* con el algoritmo de referencia al ejecutarlos sobre el corpus de DeleTe2.

En la Figura 5.20 se aprecia que en ER *DeLeTe2* obtiene mejores tasas de acierto que *MRIA*; sin embargo, en NFA hay una primera etapa en la cual *MRIA* es superior y posteriormente *DeLeTe2* es ligeramente superior, la diferencia entre ambos es de aproximadamente un punto porcentual. Al considerar los tamaños de la hipótesis obtenidas, la Figura 5.21 muestra que en NFA las hipótesis producidas por *MRIA* son consistentemente más pequeñas que las de *DeLeTe2* (llegan a ser un orden de magnitud más pequeñas) lo cual bien puede ser una razón de peso para utilizarlo a pesar de su ligera desventaja en tasa de

Cuadro 5.10: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *MRIA* y *DeLeTe2* en el Corpus de DeLeTe2

Iden.	MRIA		DeLeTe2	
	Tasa Reconocimiento	Tamaño Promedio	Tasa Reconocimiento	Tamaño Promedio
er_50	80.85 %	20.96	81.68 %	32.43
er_100	87.67 %	32.53	91.72 %	30.73
er_150	90.21 %	40,3	92.29 %	60.96
er_200	92.85 %	44.16	95.71 %	47.73
nfa_50	70.90 %	32.13	69.80 %	71.26
nfa_100	77.01 %	65	74.82 %	149.13
nfa_150	75.94 %	100.96	77.14 %	218.26
nfa_200	78.85 %	125.53	79.42 %	271.3

reconocimiento. En los experimentos ER *DeLeTe2* obtiene hipótesis más grandes que *MRIA* pero no en la proporción que se aprecia en NFA y esto sumado a sus indiscutible superioridad en tasa de reconocimiento hace que *DeLeTe2* sea el mejor de los dos para este caso.

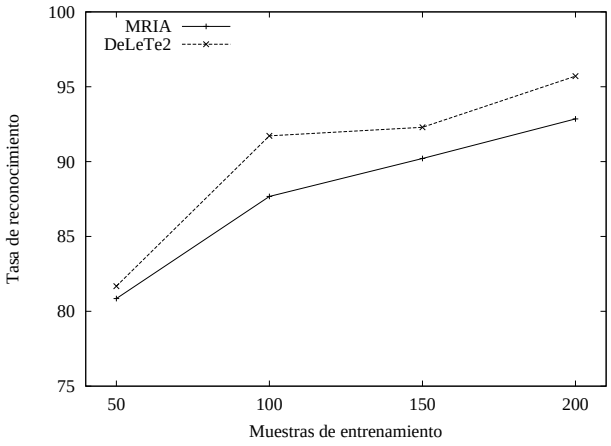
Vale comentar que previamente a la obtención de los resultados que se reportan en el Cuadro 5.10 se realizaron varias pruebas con el Corpus de DeLeTe2 que permitieron establecer la forma de aplicar el heurístico de poda de arcos en el algoritmo *MRIA*. Al final había dos posibilidades: aplicar el heurístico de poda cuando se obtiene consistencia antes de concluir todas las comparaciones posibles y la otra sólo realizar la poda cuando llega al final (ver Algoritmo 37). Las dos experimentaciones que permitieron elegir una de las dos opciones se describen a continuación.

- Se invirtieron las muestras positivas y negativas en los cuatro bloques de experimentos NFA y se ejecutaron las dos variaciones posibles del algoritmo *MRIA* con el fin de ver cual de ellas se comportaba mejor en ese contexto. Los resultados confirmaron que la opción más eficaz es la que realiza la poda únicamente cuando llega hasta el final, existiendo una diferencia promedio de dos puntos porcentuales en las tasas de reconocimiento de ambas versiones.

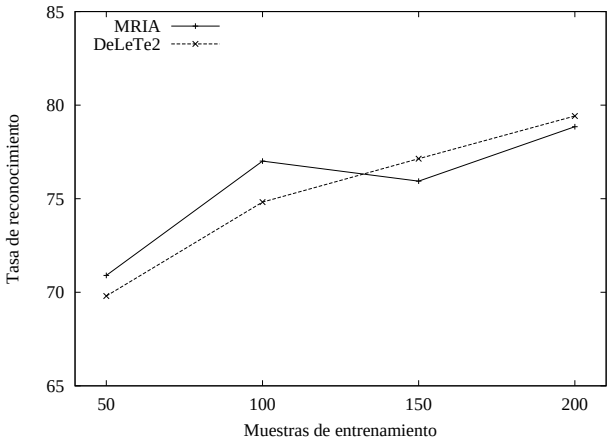
- Se aplicaron las dos versiones de *MRIA* al bloque e10 del corpus de DFAs aleatorios, formado por autómatas no deterministas; se encontró que de nuevo la versión que poda sólo al final sobrepasa en dos puntos porcentuales a la otra versión.

Estas pruebas complementarias llevaron a determinar la forma final del algoritmo *MRIA* en cuanto al momento en que se realiza la poda de arcos.

En otra experimentación se modifican los autómatas objetivo de NFA en cuanto a la probabilidad de los estados de ser iniciales y finales. Según Denis, los autómatas originales tienen P_I, P_F=0.5, en este caso se exploraron valores

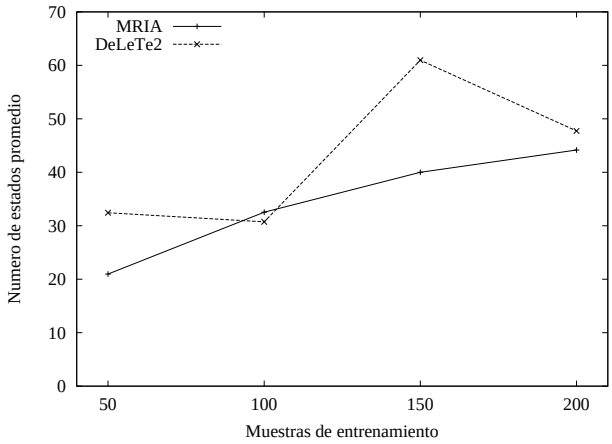


(a) Experimentos ER

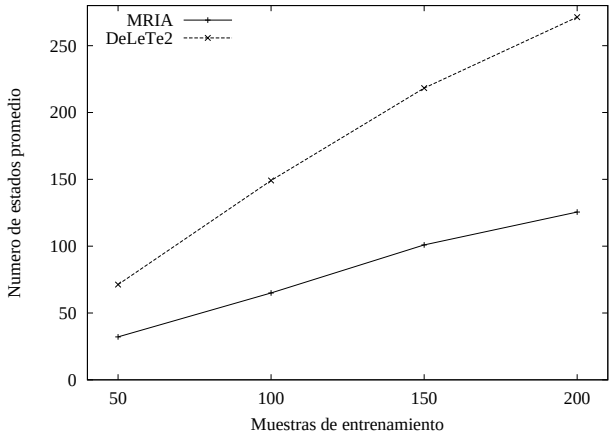


(b) Experimentos NFA

Figura 5.20: Comparación de Tasas de Reconocimiento Promedio entre *MRIA* y *DeLeTe2* en el Corpus de DeLeTe2



(a) Experimentos ER



(b) Experimentos NFA

Figura 5.21: Comparación de Tamaño Promedio de Hipótesis entre *MRIA* y *DeLeTe2* en el Corpus de DeLeTe2

de 0.05, 0.1, 0.12 y 0.15. En cada caso se registra la tasa de reconocimiento obtenida por los algoritmos *MRIA* y *DeLeTe2*, los Cuadros 5.11 y 5.12 muestran los resultados obtenidos.

Cuadro 5.11: Comparación de Tasas de Reconocimiento Promedio entre *MRIA* y *DeLeTe2* con Probabilidades P_I y P_F Menores o Iguales a 0,1 en los Experimentos NFA del Corpus de DeLeTe2

	P_I, P_F=0.05		P_I, P_F=0.1	
Iden.	*MRIA*	*DeLeTe2*	*MRIA*	*DeLeTe2*
nfa_50	95.71 %	95.64 %	86.5 %	85.56 %
nfa_100	93.23 %	93.39 %	88.80 %	87.65 %
nfa_150	93.05 %	92.63 %	89.29 %	89,45 %
nfa_200	93.38 %	94.37 %	85.11 %	86.15 %

Cuadro 5.12: Comparación de Tasas de Reconocimiento Promedio entre *MRIA* y *DeLeTe2* con Probabilidades P_I y P_F Superiores a 0,1 en los Experimentos NFA del Corpus de DeLeTe2

	P_I, P_F=0.12		P_I, P_F=0.15	
Iden.	*MRIA*	*DeLeTe2*	*MRIA*	*DeLeTe2*
nfa_50	82.57 %	80.88 %	81.35 %	77.11 %
nfa_100	87.66 %	86.54 %	83.31 %	82.07 %
nfa_150	84.08 %	82.19 %	82.56 %	82.42 %
nfa_200	86.56 %	87.02 %	82.92 %	82.23 %

En los Cuadros 5.11 y 5.12 se aprecia una mayor cercanía entre las tasas de acierto obtenidas por ambos algoritmos, lo que indica que disminuir la probabilidad de los estados de ser iniciales o finales afecta positivamente al algoritmo *MRIA* y negativamente al algoritmo *DeLeTe2*. Basta con observar que en repetidas ocasiones el algoritmo *MRIA* supera a *DeLeTe2*, especialmente cuando hay pocas muestras de entrenamiento. El buen comportamiento de *MRIA* cuando hay pocas muestras de entrenamiento, es una característica del algoritmo que se aprecia en todas las experimentaciones realizadas. Sin embargo, en el bloque nfa_200 *DeLeTe2* es el ganador en tres de los cuatro casos, en general se observa que la pendiente de la función de aprendizaje de *MRIA*, en cuanto a tasa de reconocimiento, es inferior a la del algoritmo *DeLeTe2* por lo que éste termina por alcanzarlo y superarlo.

Experimentación con el Corpus Extendido2

Al ejecutar los algoritmos *MRIA*, *DeLeTe2* e *IRPNI2* sobre los datos del Corpus Extendido2 se obtuvieron los resultados que se muestran en el Cuadro 5.13.

Cuadro 5.13: Comparación de Tasas de Reconocimiento y Tamaño Promedio de las Hipótesis entre *MRIA*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2

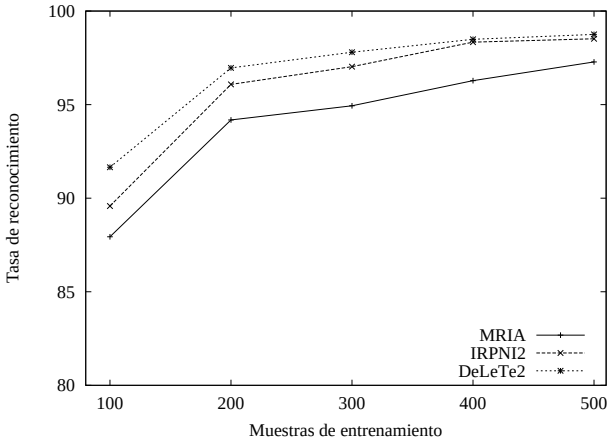
Iden.	*MRIA*		*IRPNI2*		*DeLeTe2*	
	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom.	Tasa de Rec.	Tamaño Prom
er_100	87.94 %	31.30	89.58 %	19.00	91.65 %	30.23
er_200	94.18 %	33.86	96.08 %	16.92	96.96 %	24.48
er_300	94.94 %	43.63	97.03 %	18.12	97.80 %	31.41
er_400	96.28 %	40.36	98.34 %	15.37	98.49 %	27.40
er_500	97.28 %	42.83	98.52 %	16.75	98.75 %	29.85
nfa_100	73.50 %	62.09	71.70 %	43.49	73.95 %	98.80
nfa_200	75.89 %	125.58	75.25 %	72.36	77.79 %	220.93
nfa_300	77.53 %	179.06	78.57 %	94.04	80.86 %	322.13
nfa_400	78.09 %	234.00	80.47 %	109.93	82.66 %	421.30
nfa_500	79.09 %	285.78	82.36 %	125.75	84.29 %	512.55
dfa_100	69.61 %	63.01	60.94 %	56.52	62.94 %	156.89
dfa_200	70.78 %	131.49	63.00 %	102.32	64.88 %	432.88
dfa_300	71.37 %	178.55	65.01 %	134.89	66.37 %	706.64
dfa_400	71.85 %	228.84	68.14 %	156.63	69.07 %	903.32
dfa_500	71.99 %	272.87	72.12 %	169.92	72.41 %	1027.42

En la Figura 5.22 se aprecia cómo *MRIA* no es competitivo en experimentos ER, sin embargo en NFA y DFA sí lo es porque cuando hay poca información de entrenamiento obtiene tasas de reconocimiento comparables o superiores a los otros algoritmos. Parece que el algoritmo *MRIA*, en presencia de poca información, toma decisiones que le permiten construir hipótesis de calidad, pero al obtener más información de entrenamiento el algoritmo no refina tan eficazmente su hipótesis como ocurre en los otros algoritmos y esto se refleja en que sus tasas de acierto crecen más lentamente que las de ellos. Es bastante ilustrativo el resultado de los experimentos DFA, notar que desde el primer incremento *MRIA* ya produce una tasa cercana al 70 % mientras que los otros algoritmos necesitan 500 muestras para superar esa tasa. En los experimentos NFA, el comportamiento es similar para pocas muestras de entrenamiento, pero en nfa_500 los demás algoritmos han superado a *MRIA*.

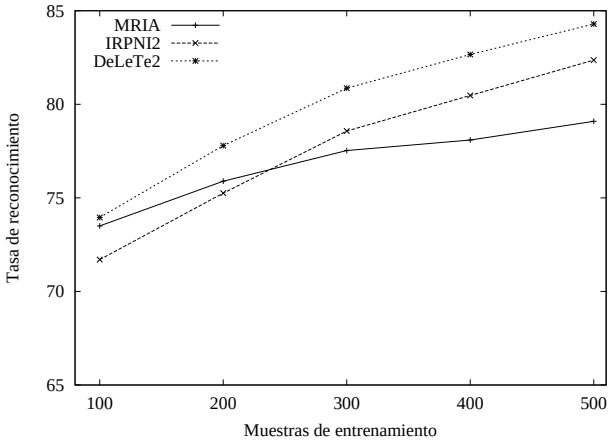
Al considerar el tamaño de las hipótesis obtenidas, en la Figura 5.23, se observa que en experimentos ER *MRIA* es poco apropiado, ya que produce las hipótesis más grandes. En los otros experimentos sus tamaños son significativamente inferiores a los de *DeLeTe2* y superiores a los de *IRPNI2*.

Experimentación con otros córpora

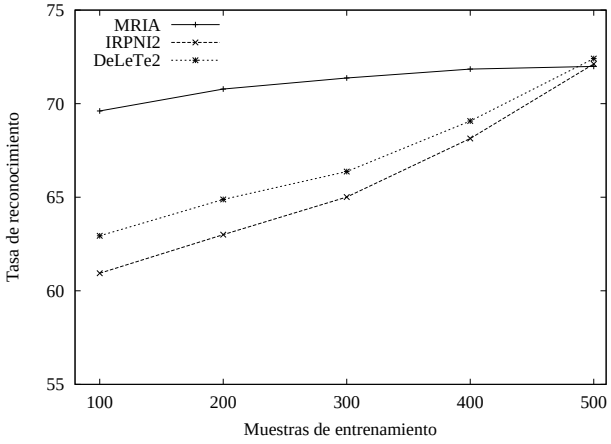
Utilizando el corpus que se construyó para la experimentación del Algoritmo *OIL* (ver Sección 3.5.2), se ejecutaron los algoritmos *MRIA* y *DeLeTe2*; en este caso sólo se evaluó tasa de reconocimiento. Se usaron los conjuntos de entrenamiento de 20, 50 y 100 muestras y se utilizaron 1000 muestras diferentes



(a) Experimentos ER

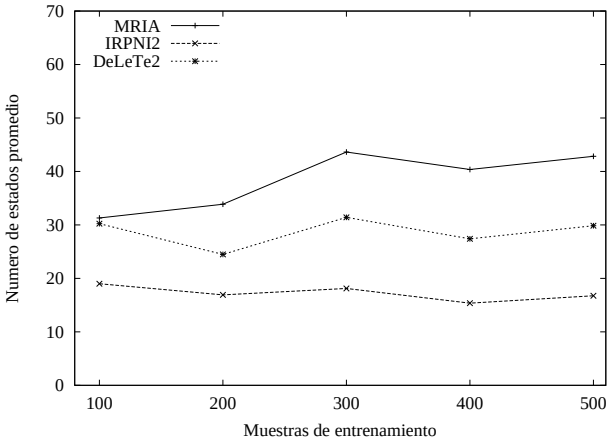


(b) Experimentos NFA

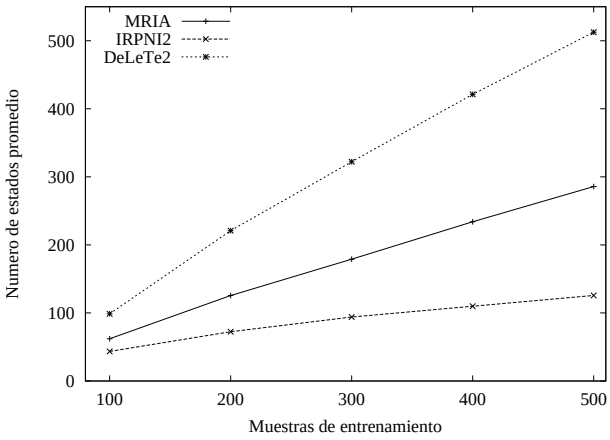


(c) Experimentos DFA

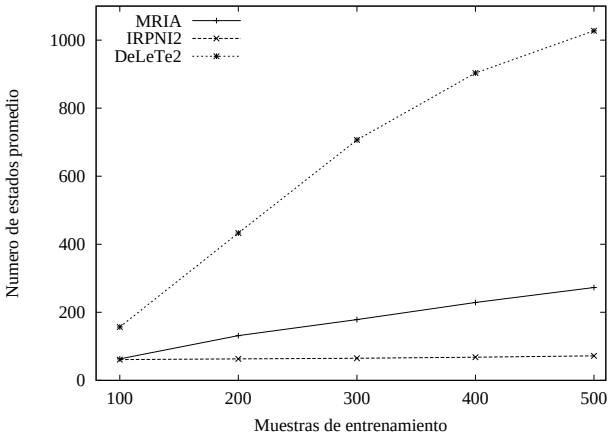
Figura 5.22: Comparación de Tasas de Reconocimiento Promedio entre *MRIA*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2



(a) Experimentos ER



(b) Experimentos NFA



(c) Experimentos DFA

Figura 5.23: Comparación de Tamaño Promedio de Hipótesis entre *MRIA*, *IRPNI2* y *DeLeTe2* en el Corpus Extendido2

a las de entrenamiento para prueba. Los resultados no se reportan en detalle ya que muestran el mismo comportamiento ya conocido, con una ventaja de *DeLeTe2* sobre *MRIA* que varía entre medio punto porcentual y dos y medio puntos porcentuales.

En resumen, en los experimentos realizados sobre NFAs, se aprecia que el algoritmo *MRIA* genera hipótesis de inferencia considerablemente más pequeñas que *DeLeTe2*, en ocasiones la diferencia es de un orden de magnitud. En cuanto a las tasas de reconocimiento, *MRIA* muestra una gran eficacia en presencia de poca información de entrenamiento, superando en este caso a *DeLeTe2*; esta característica se acentúa al disminuir la probabilidad de los estados de ser iniciales y finales. A medida que hay más información disponible, *DeLeTe2* va mejorando sus tasas de acierto hasta superar a *MRIA*. Observar en los gráficos que el ritmo al que aumentan las tasas de reconocimiento de *DeLeTe2* con respecto a *MRIA* es inferior al ritmo con que aumentan los tamaños de las hipótesis y que además *DeLeTe2* tiene una complejidad temporal que es un orden de magnitud mayor que la de *MRIA*. En general, *MRIA* se comporta bien en la inferencia de lenguajes NFA y DFA, mostrando una capacidad interesante para producir buenas hipótesis con poca información de entrenamiento, en tanto que el tamaño de dichas hipótesis es inferior al algoritmo de referencia *DeLeTe2*. Al aumentar la cantidad de información de entrenamiento disponible los otros algoritmos llegan a superar sus tasas de reconocimiento.

5.3. Sumario

En este capítulo se han comparado los algoritmos *IRPNI2*, *IRPNI1*, *RBIR*, *RRB*, *NRPNI*, *NRPNI2*, *NRPNI1* y *MRIA* con los algoritmos de referencia *RPNI*, *redBlue* y *DeLeTe2*. Las comparaciones se han realizado utilizando varios córpora: el mismo que se usó para evaluar *DeLeTe2*, a partir del cual se construyeron los córpora *Extendido1* y *Extendido2*; también el corpus de DFAs aleatorios.

Cada algoritmo es ejecutado sobre uno o más córpora y en cada caso se miden la tasa de acierto sobre muestras de prueba y el tamaño de la hipótesis de inferencia obtenida. Estos resultados se promedian por bloque de experimentos. El objetivo de estas mediciones es establecer si las estrategias de inferencia de los nuevos algoritmos permiten mejorar la tasa de acierto y/o disminuir el tamaño de las hipótesis obtenidas, con respecto a los algoritmos de referencia, tal como se afirma en la hipótesis de esta investigación.

Los algoritmos consideran la mezcla de estados determinista y no determinista; también se consideran diferentes formas de ordenar los estados para ser mezclados: en orden lexicográfico, en un orden establecido a partir de las muestras de entrenamiento, en orden aleatorio, considerando exhaustivamente todas las parejas posibles. Se aplican dos formas de obtener información acerca de las relaciones de inclusión: definición de valores de salida y adición de nuevas transiciones al modelo.

A partir del algoritmo de referencia *RPNI* surgieron los algoritmos *IRPNI2*

y *IRPNI1*. Se considera superior a *IRPNI2* ya que produce tasas de acierto más altas e hipótesis de inferencia más pequeñas que *IRPNI1*. *IRPNI2* llega a obtener tasas de acierto comparables con las de *DeLeTe2* en experimentos ER y DFA, en tanto que queda unos tres puntos porcentuales por debajo en experimentos NFA; en cuanto al tamaño de los modelos generados *IRPNI2* produce hipótesis significativamente más pequeñas que *DeLeTe2* en toda la experimentación y la diferencia llega a ser de un orden de magnitud en experimentos NFA y DFA. *IRPNI2* es un algoritmo que puede competir con *DeLeTe2* en tasas de acierto y que lo supera claramente en cuanto al tamaño de las hipótesis obtenidas; adicionalmente es un algoritmo mucho más simple, lo cual se aprecia al comparar tanto sus cotas de complejidad como sus tiempos de ejecución con los de *DeLeTe2*.

A partir del algoritmo de referencia *redBlue* surgen los algoritmos *RBIRX*, *RBIRZ* y *RRB*. *RBIRX* tiene menor complejidad temporal que *RBIRZ* y también produce hipótesis de menor tamaño, en cuanto a tasas de acierto, ambas versiones producen resultados similares, siendo ligeramente inferiores, aunque comparables con los resultados obtenidos por *IRPNI2*. *RRB* muestra un comportamiento levemente superior a *IRPNI2*; teniendo en cuenta que ejecutar *IRPNI2* es sólo un paso dentro de la estrategia *RRB*, queda a juicio del potencial usuario determinar si amerita realizar el resto de los pasos para obtener esa ganancia o directamente usar el *IRPNI2* y sacrificar algunas décimas tanto en tasa de reconocimiento como en tamaño de las hipótesis obtenidas.

El algoritmo *NRPNI* mejora levemente las tasas de acierto de los algoritmos a partir de los cuales se ejecuta (por ejemplo, *RPNI*, *IRPNI2* y *IRPNI1* en experimentos ER) sin embargo, en algunos grupos de experimentos NFA y en casi todos los experimentos DFA causa una ligera disminución en dichas tasas. En todos los casos produce también una disminución en el tamaño de las hipótesis obtenidas lo cual es positivo. Considerando la inferencia de DFAs, *NRPNI* resulta ser mejor opción que *NRPNI2* o el mismo *IRPNI2*, tanto en tasas de acierto como en el tamaño de las hipótesis de inferencia, lo que sugiere que este algoritmo podría tener interés en contextos donde no se conozca por anticipado la naturaleza de los lenguajes que se van a aprender, pudiendo ellos provenir de expresiones regulares, NFAs ó DFAs.

El algoritmo *MRIA* muestras sus mejores resultados en la inferencia de lenguajes objetivo provenientes de NFAs, allí iguala o supera a los algoritmos de referencia cuando hay pocas muestras de entrenamiento, pero a medida que hay más información disponible, el algoritmo va siendo menos competitivo. En cuanto al tamaño de las hipótesis que produce, son más pequeñas que las que produce *DeLeTe2* y son más grandes que las que produce *IRPNI2*.

Capítulo 6

Conclusiones

Esta investigación se inició con el propósito de establecer si el uso de NFAs como modelo de representación, permite mejorar la velocidad de convergencia, la tasa de acierto y/o disminuir el tamaño de las hipótesis obtenidas durante el proceso de inferencia gramatical de algunas subclases de lenguajes regulares. Los lenguajes regulares de interés son aquellos generados aleatoriamente a partir de expresiones regulares y NFAs.

Al considerar nuevamente la hipótesis de investigación a la luz de los experimentos realizados y sus correspondientes resultados, se puede afirmar que ninguno de los algoritmos propuestos mostró mayor velocidad de convergencia atribuíble al uso de un modelo de representación no determinista, esta condición se ve más afectada por otras variables entre las que destaca el orden en que se realizan las mezclas. El estudio de la convergencia cobró particular interés en esta investigación y permitió concluir que un algoritmo de mezcla de estados converge en el límite independientemente del orden en que se realicen las mezclas, tanto en el caso de inferir DFAs como NFAs [GdPAR07]. Este resultado es importante, porque muestra que la técnica EDSM (Evidence Driven State Merging) es convergente y por lo tanto lo son los algoritmos que la utilizan. Además, al mostrar que esta propiedad se cumple también en la inferencia de modelos no deterministas se abre la posibilidad de diseñar nuevos algoritmos para este contexto en el que se conocen pocos resultados previos exitosos.

En cuanto a la mejora en la tasa de reconocimiento, la estrategia de extender algoritmos clásicos para inferir RFSAs (Residual Finite State Automata) mostró mejoras en dichas tasas al comparar la versión extendida con la versión inicial, en el contexto de los lenguajes regulares construidos aleatoriamente a partir de expresiones regulares y NFAs [GRCA05, GRCA06, ARCG05, AGR06]. También se desarrollaron algunos algoritmos desde cero, en esos casos, se obtuvieron tasas en el mismo rango de los algoritmos extendidos y del algoritmo de referencia *DeLeTe2* [AGR06, ARG07]. Al comparar la tasa de desempeño de las versiones extendidas con las del algoritmo de referencia *DeLeTe2* se obtuvieron valores semejantes, aunque ligeramente inferiores, siendo los algoritmos *RRB* e *IRPNI2* [GRCA05] los que muestran las tasas más altas. En condiciones específicas, como por ejemplo variando la probabilidad de que un estado sea inicial o final en la generación de los lenguajes objetivo, fue posible superar

ligeramente los resultados de *DeLeTe2* con el algoritmo *MRIA* [ARG07]. Los algoritmos *RBIRX* y *RBIRZ* alcanzan tasas de acierto tan cercanas a *IRPNI2* que no es fácil establecer cuál tiene mejor desempeño. Estos resultados permiten afirmar que el uso de modelos no deterministas permite obtener tasas de acierto más altas en la identificación de lenguajes regulares aleatorios generados a partir de expresiones regulares y NFAs, con respecto a los resultados obtenidos con algoritmos basados exclusivamente en la detección de relaciones de equivalencia entre estados y que generan autómatas deterministas.

Las hipótesis obtenidas por todos los algoritmos diseñados en esta investigación son sistemáticamente más pequeñas que las producidas por el algoritmo de referencia *DeLeTe2*, que tiene tasas de acierto similares. Los inmensos tamaños de las hipótesis de *DeLeTe2* sorprenden sobre todo debido a que Denis et. al. resaltan la posibilidad de obtener hipótesis pequeñas como una de las motivaciones para desarrollar dicho algoritmo. No es útil en este caso compararse con los otros algoritmos de referencia ya que ellos obtienen tasas de acierto muy inferiores y en esas condiciones obtener una hipótesis pequeña no es una ventaja. Resumiendo, la experimentación muestra que el uso de modelos no deterministas permite encontrar hipótesis de inferencia hasta un orden de magnitud más pequeñas que las que consigue *DeLeTe2* y conservando tasas de acierto similares [GRCA05, GRCA06, ARCG05, AGR06, ARG07], siendo este el aspecto de la hipótesis de trabajo en el cual se logró una mayor aportación.

Al estudiar la inferencia de modelos no deterministas, se dio especial importancia al estudio de los RFSA y por lo tanto al estudio de las relaciones de inclusión entre los lenguajes residuales asociados a los estados de un RFSA o un DFA. Esto llevó a establecer algunas ideas de diseño que condujeron al desarrollo no sólo de algoritmos que infieren RFSAs sino también algoritmos que infieren DFAs. De todos los algoritmos diseñados y evaluados destaca el *IRPNI2* como el mejor de ellos en términos generales, esto se debe a que tiene una estrategia simple y puede implementarse eficientemente (su complejidad temporal en peor caso es un orden de magnitud inferior a la de *DeLeTe2*), obtiene tasas de acierto semejantes a las de *DeLeTe2* e hipótesis un orden de magnitud más pequeñas que éste. En el caso específico de los experimentos ER, el algoritmo *NRPNI2* obtiene resultados ligeramente superiores a *DeLeTe2* en tasas de acierto y genera hipótesis cuyo tamaño es entre la mitad y una tercera parte de las de éste.

Los algoritmos basados en relaciones de inclusión, tanto si infieren modelos deterministas como no deterministas, tienen un comportamiento pobre en caso que dichas relaciones sean escasas. Los experimentos confirman que tanto *DeLeTe2* como los algoritmos diseñados en esta investigación, tienen un desempeño inferior al de los algoritmos de referencia *RPNI* y *redBlue* en la identificación de lenguajes objetivo generados a partir de DFAs aleatorios. Este hecho limita la utilidad de los algoritmos al tipo de lenguaje objetivo que se quiere identificar, lo cual no siempre se puede conocer anticipadamente.

Aunque se exploraron diferentes tipos de mezcla, diversos ordenamentos de los estados a mezclar y varias formas de considerar la información que aportan las relaciones de inclusión, las tasas de acierto alcanzadas son muy similares. Esto se puede interpretar como una consecuencia de haber aprovechado ya la

información que aportan dichas relaciones haciendo que no sea posible mejorar aún más a menos que otro tipo de información sea tenido en cuenta adicionalmente.

La subclase de los RFSA permitió explorar la inferencia de modelos no deterministas manteniendo la existencia de una única representación canónica hacia la cual converger. Esto fue positivo en cuanto al diseño de los algoritmos pero negativo debido a que el RFSA canónico puede ser tan grande como el DFA mínimo correspondiente. La búsqueda de nuevos algoritmos que infieran modelos no deterministas debería partir del estudio de otras subclases de NFAs con propiedades útiles para lograr la convergencia.

La principal línea de trabajo futuro sigue siendo la búsqueda de nuevos algoritmos de inferencia gramatical para lenguajes regulares. Explorar nuevas subclases de NFAs u otros tipos de información adicional que pueda involucrarse en los algoritmos sería una posible manera de proceder con el propósito de mejorar las tasas de acierto y disminuir aun más los tamaños de las hipótesis obtenidas en la inferencia de NFAs ó DFAs para representar lenguajes objetivos que son expresiones regulares o NFAS.

Dado que este trabajo se ha enfocado en un tipo concreto de lenguajes regulares, la posibilidad de establecer una aplicación específica en la que se deban inferir expresiones regulares o NFAs sería una buena forma de continuar esta línea de trabajo, porque es muy posible que se puedan refinar los algoritmos si se tiene una tarea concreta en mente.

Bibliografía

[ACS04] John Abela, François Coste, and Sandro Spina. Mutually Compatible and Incompatible Merges for the Search of the Smallest Consistent DFA. *Lecture Notes in Artificial Intelligence*, 3264:28–39, 2004.

[ADN70] A. Arnold, A. Dicky, and M.Ñivat. A Note about Minimal Nondeterministic Automata. *Bulletin EATCS*, 47:166–169, 1970.

[AGR06] Gloria Alvarez, Pedro García, and José Ruiz. A Merging States Algorithm for Inference of RFSAs. *Lecture Notes in Artificial Intelligence*, 4201:340–341, 2006.

[Ang78] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

[Ang87] Dana Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1987.

[Ang90] Dana Angluin. Negative Results for Equivalence Queries. *Machine Learning*, 5:121–150, 1990.

[Ang92] Dana Angluin. Computational Learning Theory: Survey and Selected Bibliography. *ACM Computing Surveys*, pages 351–368, 1992.

[ARCG05] Gloria Alvarez, José Ruiz, Antonio Cano, and Pedro García. Nondeterministic Regular Positive Negative Inference NRPNI. In Juan Francisco Diaz, Camilo Rueda, and Antal A. Buss, editors, *XXXI Conferencia Latinoamericana de Informatica CLEI 2005*, pages 239–249, October 2005.

[ARG07] Gloria Alvarez, José Ruiz, and Pedro García. Inferencia gramatical de lenguajes regulares mediante autómatas no deterministas: Comparación de dos algoritmos recientes. *Revista Epiciclos*, 2007.

[BJR06] Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines with parameters. *Lecture Notes in Computer Science*, 3922(0):107–121, 2006.

[BL05] Josh Bongard and Hod Lipson. Active coevolutionary learning of deterministic finite automata. *Journal of Machine Learning Research*, 6:1651–1678, 2005.

[BO05] Miguel Bugalho and Arlindo Oliveira. Inference of regular languages using state merging algorithms with search. *Pattern Recognition*, 38:1457–1467, 2005.

[Car70] C Carrez. *On the Minimalization of Non-deterministic Automata*. PhD thesis, Laboratoire de Calcul de la Faculté des Sciences de L'Universite de Lille, 1970.

[CF00] François Coste and Daniel Fredouille. Efficient Ambiguity Detection in C-NFA. A step Towards the Inference of Non Deterministic Automata. In *ICGI*, pages 25–38, 2000.

[CF03a] François Coste and Daniel Fredouille. Unambiguous Automata Inference by Means of State-merging Methods. *Lecture Notes in Computer Science*, 2837:60–71, 2003.

[CF03b] François Coste and Daniel Fredouille. What is the Search Space for the Inference of Non-deterministic, Unambiguous and Deterministic Automata? *Internal Report Project Symbiose, INRIA*, 4907:1–41, 2003.

[CFKdlH04] François Coste, Daniel Fredouille, Christopher Kermorvant, and Colin de la Higuera. Introducing Domain and Typing Bias in Automata Inference. *Lecture Notes in Artificial Intelligence*, 3264:115–126, 2004.

[CFW06] Alexander Clark, Christophe Costa Florêncio, and Chris Watkins. Lenguages as hyperplanes: Grammatical inference with string kernels. *Lecture Notes in Artificial Intelligence*, 4212:90–101, 2006.

[CFWS06] Alexander Clark, Christophe Costa Florêncio, Chris Watkins, and Mariette Serayet. Planar languages and learnability. *Lecture Notes in Artificial Intelligence*, 4201:148–160, 2006.

[Cho03] Ben Choi. Inductive inference by using information compression. *Computational Intelligence*, 19(2), 2003.

[CK02] Orlando Cichello and Stefan C. Kremer. Beyond EDSM. *Lecture Notes in Artificial Intelligence*, 2484:37–48, 2002.

[CK03] Orlando Cichello and Stefan C. Kremer. Inducing Grammars from Sparce Data Sets: A Survey of Algorithms and Results. *Journal of Machine Learning Research*, 4:603–632, 2003.

[CK05] François Coste and Goulven Kerbellec. A Similar Fragments Merging Approach to Learn Automata on Proteins. *IRISA Publication Interne*, 1735, 2005.

[CK06] François Coste and Goulven Kerbellec. Learning automata on protein sequences. In *Jornees Ouvertes en Biologie Informatique Mathematiques JOBIM'06*, 2006.

[CO94] Rafael C. Carrasco and José Oncina. Learning Stochastic Regular Grammars by Means of a State Merging Method. *Lecture Notes in Artificial Intelligence*, 862:139–150, 1994.

[CP05] Jean Mark Champarnaud and Thomas Paranthoën. Random Generation of DFAs. *Theoretical Computer Science*, 330(2):221–235, 2005.

[CRG02] Antonio Cano, José Ruiz, and Pedro García. Inferring Subclasses of Regular Languages Faster Using RPNI and Forbidden Configurations. *Lecture Notes in Artificial Intelligence*, 2484:28–36, 2002.

[dlH05] Colin de la Higuera. A Bibliographical Study of Grammatical Inference. *Pattern Recognition*, 38:1332–1348, 2005.

[dlHOV96] Colin de la Higuera, José Oncina, and Enrique Vidal. Identification of DFA: Data-Dependent Versus Data-Independent Algorithms. *Lecture Notes in Artificial Intelligence*, 1147:313–325, 1996.

[DLT00] Francois Denis, Aurelien Lemay, and Alain Terlutte. Learning Regular Languages Using Non deterministic Finite Automata. *Lecture Notes in Artificial Intelligence*, 1891:39–50, 2000.

[DLT02] Francois Denis, Aurelien Lemay, and Alain Terlutte. Residual Finite State Automata. *Fundamenta Informaticae*, XX:1–30, 2002.

[DLT04] Francois Denis, Aurelien Lemay, and Alain Terlutte. Learning Regular Languages Using RFSAs. *Theoretical Computer Science*, 313:267–294, 2004.

[DMV94] Pierre Dupont, Laurent Miclet, and Enrique Vidal. What is the Search Space of the Regular Inference. *Proceedings of the Second International Colloquium on Grammatical Inference*, pages 25–37, 1994.

[dPGR06] Manuel Vázquez de Parga, Pedro García, and José Ruiz. A family of algorithms for non deterministic regular languages inference. *LNCS*, 4094:265–274, 2006.

[Dup94] Pierre Dupont. Regular gramátical inference from positive and negative samples by genetic search: the gig method. *Lecture Notes in Artificial Intelligence*, 862:236–245, 1994.

[Dup96] Pierre Dupont. Incremental Regular Inference. *Lecture Notes in Artificial Intelligence*, 1147:222–237, 1996.

[Gar06] Pedro García. Método para Construir el RFSA canónico a partir de un NFA dado. Conversación sostenida con el autor, 2006.

[GCR00] Pedro García, Antonio Cano, and José Ruiz. A Comparative Study of Two Algorithms for Automata Identification. *Lecture Notes in Artificial Intelligence*, 1891:115–126, 2000.

[GdP05] Pedro García and Manuel Vázquez de Parga. A note about mergible states in large nfa. *Bulletin of the EATCS*, 87:181–184, 2005.

[GdPAR07] Pedro García, Manuel Vázquez de Parga, Gloria I. Álvarez, and José Ruiz. Universal automata and nfa learning. En proceso de evaluación, 2007.

[GJP06] Olga Grinchtein, Bengt Jonsson, and Paul Pettersson. Inference of event-recording automata using timed decision trees. *Lecture Notes in Computer Science*, 4137(0), 2006.

[Gol67] E. Mark Gold. Language Identification in the Limit. *Information and Control*, 10(5):447–474, 1967.

[Gol78] E. Mark Gold. Complexity of Automaton Identification from Given Data. *Information and Control*, 37(3):302–320, 1978.

[GRC04a] Pedro García, José Ruiz, and Antonio Cano. Identification of Residual Finite State Automata. *Internal Report DSIC, Universidad Politécnica de Valencia*, 2004.

[GRC04b] Pedro García, José Ruiz, and Antonio Cano. Non Deterministic RPNI. *Internal Report DSIC-II/10/04, Universidad Politécnica de Valencia*, 2004.

[GRCA05] Pedro García, José Ruiz, Antonio Cano, and Gloria Alvarez. Inference Improvement by Enlarging the Training Set while Learning DFAs. *Lecture Notes in Computer Science*, 3773:59–70, 2005.

[GRCA06] Pedro García, José Ruiz, Antonio Cano, and Gloria Alvarez. Is Learning RFSAs Better Than Learning DFAs? *Lecture Notes in Computer Science*, 3845:343–344, 2006.

[HMU01] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Langages and Computation*. Pearson Education. Addison Wesley, http://www-db.stanford.edu/ ullman/ialc.html, second edition, 2001.

[JP98] Hugues Juillé and Jordan B. Pollack. A Sampling-Based Heuristic for Tree Search Applied to Grammar Induction. In *Tenth Conference on Innovative Applications of Artificial Intelligence AAAI98*, pages 26–33, 1998.

[KBBdB06] Raymond Kosala, Hendrik Blockeel, Maurice Bruynooghe, and Jan Van den Busshe. Information extraction from structured documents using k-testable tree automaton inference. *Data & Knowledge Engineering*, 58:129–158, 2006.

[Lan92] Kevin J. Lang. Random DFA's can be Approximately Learned from Sparse Uniform Examples. *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pages 45–52, 1992.

[Lan99] Kevin J. Lang. Faster algorithms for finding minimal consistent dfas. *Journal*, page 0, 1999.

[Lom01] S. Lombardy. *Approche Structurelle de Quelques Problémes de la Théorie des Automates*. PhD thesis, Ecole N. S. des Télécomunications, 2001.

[LPP98] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo One DFA Learning Competition and a New Evidence-driven State Merging Algorithm. *Lecture Notes in Artificial Intelligence*, 1433:1–12, 1998.

[LR05] Simon M. Lucas and T. Jeff Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1063–1074, 2005.

[Myh57] J. Myhill. Finite Automata and the Representation of Events. Technical Report 57-624, WADC, 1957.

[Ner58] A.Ñerode. Linear Automaton Transformation. In *American Mathematical Society*, volume 9, pages 541–544, 1958.

[OG92] José Oncina and Pedro García. Inferring Regular Languages in Polinomial Updated Time. *Pattern Recognition and Image Analysis*, pages 49–61, 1992.

[OS01] Arlindo Oliveira and Joao Silva. Efficient algorithms for the inferencie of minimum size dfas. *Machine Learning*, 44(1-2):93–119, 2001.

[PCS05] Nicolas Pernot, Antoine Cornuéjols, and Michèle Sebag. Phase Transitions within Grammatical Inference. In *IJCAI*, pages 811–816, 2005.

[PH97] Rajesh G. Parekh and Vasant G. Honavar. Learning DFA from Simple Examples. *Lecture Notes in Artificial Intelligence*, 1316:116–131, 1997.

[PH00] R. Parekh and V. Honavar. Grammar Inference, Automata Induction, and Language Acquisition. 2000.

[PN02] Pravin Pawar and G.Ñagaraja. Regular gramatical inference: A genetic algorithm approach. *Lecture Notes in Artificial Intelligence*, 2275:429–435, 2002.

[PO99] Jorge M. Pena and Arlindo L. Oliveira. A new algoritm for the reduction of incompletely specified finite state machines. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(11):1619–1632, 1999.

[Pol05] Libor Polák. Minimalizations of NFA Using the Universal Automaton. *Int. J. Found. Comput. Sci.*, 16(5):999–1010, 2005.

[SJ03] Marc Sebban and Jean-Christophe Janodet. On state merging in grammatical inference: A statistical approach for dealing with noisy data. In *ICML Twentieth International Conference on Machine Learning*, pages 688–695, 2003.

[TB73] Boris A. Trakhtenbrot and Ya M. Barzdin. Finite Automata: Behaviour and Synthesis. *North Holland Publishing Company, Amsterdam*, 1973.

[Tom82] M. Tomita. Construction of finite autómata from examples using hill climbing. In *Proc. of the 4th Annual Cognitive Science Conference*, pages 105–108, 1982.

[Val84] Leslie G. Valiant. A Theory of the Learnable. *Communication of ACM*, 27:1134–1142, 1984.

[Yok94] T. Yokomori. Learning Non-deterministic Finite Automata from Queries and Counterexamples. *Machine Intelligence*, 13:169–189, 1994.