Comparación de dos algoritmos recientes para inferencia gramatical de lenguajes regulares mediante autómatas no deterministas

Gloria I. Alvarez*[§], José Ruiz, Pedro García****

**Grupo DESTINO, Departamento de Ciencias e Ingeniería de la Computación,*
Pontificia Universidad Javeriana, Cali, Colombia
***Grupo en Teoría de Lenguajes, Computabilidad y Criptografía,*
Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia, España
§ e-mail: galvarez@javerianacali.edu.co

(Recibido: Agosto 22 de 2008 - Aceptad o: Abril 15 de 2009)

Resumen

El desarrollo de nuevos algoritmos, que resulten convergentes y eficientes, es un paso necesario para un uso provechoso de la inferencia gramatical en la solución de problemas reales y de mayor tamaño. En este trabajo se presentan dos algoritmos llamados *DeLeTe2* y *MRIA*, que implementan la inferencia gramatical por medio de autómatas no deterministas, en contraste con los algoritmos más comúnmente empleados, los cuales utilizan autómatas deterministas. Se consideran las ventajas y desventajas de este cambio en el modelo de representación, mediante la descripción detallada y la comparación de los dos algoritmos de inferencia con respecto al enfoque utilizado en su implementación, a su complejidad computacional, a sus criterios de terminación y a su desempeño sobre un cuerpo de datos sintéticos.

Palabras clave: Inferencia gramatical, Lenguajes regulares, Autómatas no deterministas, Autómatas finitos residuales.

Comparison of two recent algorithms for grammatical inference of regular languages by means of non-deterministic automata

Abstract

The development of new algorithms that turn out to be convergent and efficient, is a required step for a fruitful use of grammatical inference in the solution of real-world and greater-size problems. In this work, we present two algorithms known as *DeLeTe2* and *MRIA*, which do grammatical inference by means of non-deterministic automata, in contrast to the algorithms more commonly used, which make use of deterministic automata. We consider the advantages and disadvantages of such a change of representation model by means of a detailed description and comparison of the two inference algorithms with regard to the approach used for their implementation, their computational complexity, their termination criteria, and their performance on a corpus of synthetic data.

Keywords: Grammatical inference, Regular languages, Non-deterministic automata, Residual finite automata.

1. Introducción

La inferencia gramatical de lenguajes regulares, que durante años ha sido un problema de interés teórico, se ha convertido en un problema de interés práctico dada la variedad de contextos de aplicación que en la actualidad requieren de sus resultados. Por ejemplo, para modelar el habla, se utiliza inferencia gramatical como un paso necesario en la construcción de sistemas de diálogo entre los seres humanos y los computadores y en sistemas de traducción automática. También es útil la inferencia gramatical en problemas de robótica y sistemas de control, biología computacional, manejo de documentos, compresión y minería de datos. En general, la inferencia gramatical puede aportar a la solución de problemas de reconocimiento estructural de patrones (De la Higuera, 2005).

El desarrollo de nuevos algoritmos de inferencia, precisos y eficientes, es el paso necesario para que la inferencia gramatical pueda ser aprovechada en la solución de problemas de mayor tamaño. A continuación se presentan y comparan dos estrategias de inferencia gramatical que aplican conceptos teóricos y que muestran potencial para servir de base a futuros desarrollos de utilidad práctica.

Este artículo hace un recuento de la historia reciente de una aproximación particular a la inferencia gramatical, que es la que proviene de la teoría de funciones recursivas: desde los trabajos de Gold (1967, 1978) hasta el presente. En la Sección 2, se hace un breve recuento de trabajos destacados en esta área durante los ultimos 30 años. En la Sección 3, se presenta la inferencia de autómatas no deterministas con sus beneficios y dificultades. En la Sección 4, se presentan algunas definiciones formales y convenciones de notación utilizadas en el resto del artículo. En las Secciones 5 y 6 se describen dos algoritmos de inferencia de autómatas no deterministas: *DeLeTe2* y *MRIA*. En la Sección 7, se comparan ambas estrategias desde diversos ángulos. Por último, en la Sección 8 se presentan las conclusiones de este trabajo.

2. Antecedentes

El problema de la inferencia gramatical consiste en construir un modelo o representación de un lenguaje objetivo a partir de unas muestras de entrada. La entrada del problema está formada por un conjunto no vacío de muestras positivas y un conjunto, que puede ser vacío, de muestras negativas. El lenguaje objetivo es un lenguaje formal, que en este caso será siempre un lenguaje regular. Las muestras positivas son cadenas de símbolos que pertenecen al lenguaje objetivo y las muestras negativas son cadenas que no pertenecen a él. Si el problema de la inferencia gramatical se plantea como un problema de minimización (encontrar una hipótesis de tamaño mínimo), se sabe que es un problema de programación no determinísticamente polinomial y completo (NP-completo).

Existen varias aproximaciones a la solución de este problema: se puede abordar desde la teoría de funciones recursivas, desde la estadística o desde una mezcla de ambos. Existen enfoques que utilizan como entrada únicamente las muestras de entrenamiento, mientras que otros requieren información adicional.

El campo de la inferencia gramatical de lenguajes regulares ha evolucionado mucho en los últimos 30 años, tanto en cuanto a resultados teóricos como en el desarrollo de algoritmos de uso práctico. A continuación, se presenta una relación comentada de trabajos que se consideran importantes dentro de este campo de estudio.

Gold (1967, 1978) 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 (Gold, 1978). También propuso el modelo de identificación en el límite (Gold, 1967), que es una valiosa herramienta para demostrar la corrección de algoritmos de inferencia gramatical. Además, Gold (1978) propuso un algoritmo de inferencia que, dado un conjunto de ejemplos suficientemente representativo, puede construir en tiempo polinomial el autómata determinista mínimo.

Este algoritmo es particularmente interesante, ya que varios algoritmos posteriores pueden entenderse como modificaciones suyas (García et al., 2004a).

En la misma época, Trakhtenbrot & Barzdin (1973) propusieron su algoritmo de inferencia, 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 similar, aunque utilizan representaciones muy diferentes de la información (García et al., 2000).

En la década de 1980 aparecieron nuevos resultados negativos que confirmaron la dificultad teórica del problema de inferencia gramatical de lenguajes regulares. Angluin (1988) 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 (De la Higuera, 2005).

Angluin (1988) también demostró que la tarea sigue siendo difícil aún si se supone que el algoritmo puede formular preguntas de pertenencia o de equivalencia a un oráculo (Angluin, 1990). 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 de los años 90 y hasta el presente. Por ejemplo, la inferencia mediante autómatas deterministas estocásticos (Carrasco & Oncina, 1994); 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 (Parekh & Honavar, 1997; Parekh & Honavar, 2000; Angluin, 1988; Angluin, 1990; Angluin, 1992); otras formas de involucrar información del entorno en el proceso de inferencia (Coste et al., 2004) y el uso de técnicas de inteligencia computacional como redes neuronales y algoritmos genéticos (Dupont et al., 1994).

A comienzos de la década de los años 90 apareció el algoritmo *RPNI* (Oncina & García, 1992) y Lang (1992) propuso una modificación al algoritmo de Trakhtenbrot y Brazdin (1973) llamada *Traxbar* al cual se hace referencia en el trabajo de Cicchello & Kremer (2003). Estos algoritmos, gracias a su capacidad de generalización pueden inferir autómatas deterministas mínimos en tiempo polinomial. En el caso del algoritmo *RPNI*, la precisión de la aproximación depende de la calidad de la información en el entrenamiento y en el caso en que el conjunto de entrenamiento sea característico (Oncina & García, 1992), se demuestra que el algoritmo converge al DFA minimal del lenguaje objetivo. Además de los esfuerzos por generar nuevos algoritmos, algunos investigadores han estudiado el espacio de búsqueda del problema de inferencia gramatical (Dupont et al., 1994; Coste & Fredouille, 2003b) y su relación con el espacio de búsqueda de otros problemas NP-completos (Pernot et al., 2005).

En los años siguientes continuaron apareciendo mejoras a estos algoritmos (Dupont, 1996; Cano et al., 2002). Lang et al. (1998) propusieron el concurso *Abbadingo* en el cual compitieron diferentes tipos de algoritmos de inferencia gramatical sobre un conjunto de problemas de inferencia de lenguajes regulares. Dos algoritmos fueron los ganadores: uno de ellos 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, Lang et al. (1998), basándose en los resultados del concurso, propusieron su algoritmo *Red-Blue* descrito en el trabajo de Cicchello & Kremer (2003). El algoritmo *EDSM* (*Evidence Driven State Merging*) de R. Price, que fue uno de los ganadores, también ha dado origen a posteriores estudios y mejoras (Cicchello & Kremer, 2003; Abela et al., 2004).

Una posibilidad que eventualmente ha sido considerada, es la de inferir autómatas no deterministas (NFA) en lugar de autómatas deterministas. Yokomori (1994) publicó un trabajo en este tema cuyo método utilizaba consultas a un oráculo y contraejemplo (De la Higuera, 2005). También lo hicieron Coste & Fredouille (2000, 2003a, 2003b) y Coste et al. (2004) quienes plantean la inferencia de NFAs como un caso particular de la inferencia de autómatas finitos no ambiguos.

A partir del año 2000, Denis et al. (2000, 2001) 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 et al. (2000) propusieron un algoritmo que converge a él, llamado algoritmo *DeLeTe*. Desafortunadamente, pruebas experimentales muestran que sus hipótesis con frecuencia son inconsistentes con las muestras de entrenamiento. Posteriormente, presentaron el programa *DeLeTe2* (Denis et al., 2004). Nótese que esos autores reportaron los resultados de ejecutar dicho programa, pero no explicaron el algoritmo utilizado (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).

En la actualidad, se conocen pocos grupos trabajando en el tema de la inferencia de NFAs. Algunos buscan métodos de minimización de NFAs que conduzcan a una forma canónica (Polák, 2005). Otros apuntan al desarrollo de algoritmos de inferencia para problemas específicos como es el caso de un algoritmo para identificación de proteinas (Coste & Kerbellec, 2005). También se ha experimentado con algoritmos que converjan a RFSAs (García et al., 2004a; García et al., 2004b; Alvarez et al., 2005); con algoritmos que infieren UFAs (*Unambiguous Finite Automata*) (Abela et al., 2004; Coste & Fredouille, 2003a, 2003b); y con extensiones a la estrategia *RPNI* que realizan mezclas no deterministas de estados (García et al., 2005, 2006).

3. Inferencia de autómatas no deterministas

Aunque en la última década han aparecido algunos trabajos sobre la inferencia de lenguajes regulares mediante autómatas no deterministas (NFAs), éste no es un tema en el que abunden los aportes. La escasez de trabajos se puede explicar en la mayor complejidad de este enfoque con respecto a la inferencia de DFAs. El hecho de 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, dadas 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 minimal, es decir, aquél autómata determinista que identifica el lenguaje objetivo con un número mínimo de estados. La teoría establece que cada lenguaje regular tiene asociado un único DFA minimal. Al pasar a la inferencia de NFAs, y procediendo también de forma iterativa, se observa que no existe un único NFA minimal asociado a cada lenguaje regular sino varios NFAs mínimos 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 de 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 mínimos que lo representan es menor o igual que el tamaño del DFA minimal que lo representa. En ocasiones, el número de estados de un NFA mínimo puede ser exponencialmente menor que el del mencionando DFA minimal. Obtener representaciones precisas y a la vez pequeñas es una condición necesaria para poder aplicar la inferencia en problemas de tamaño real.

La estrategia que se viene siguiendo en los últimos años para realizar inferencia de NFAs, consiste en proponer subclases de NFAs que conserven propiedades importantes de los DFAs como la existencia de un único minimal para cada lenguaje regular. Por ejemplo, en los trabajos de Coste & Fredouille (2000, 2003a, 2003b), se propone la subclase de los UFAs (*Unambiguous Finite Automata*) y Denis et al. (2001, 2004) y Alvarez et al. (2006) proponen la subclase de los RFSAs (*Residual Finite State Automata*). Todas estas aproximaciones han sido evaluadas experimentalmente y reportan un mejor comportamiento que la inferencia de DFAs para ciertos tipos de lenguajes regulares. El mejor comportamiento puede significar una mayor precisión en la aproximación al lenguaje objetivo, un menor tiempo de convergencia o un 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.

Puestos ante problemas prácticos, los algoritmos de inferencia gramatical eficientes, como *RPNI* o TraxBar, son útiles en la solución de una amplia variedad de problemas. Pero estos algoritmos no escalan suficientemente bien para ser utilizados en problemas reales, caracterizados por la gran longitud de las muestras y el gran tamaño del alfabeto y del autómata objetivo. De una parte, la construcción del modelo se hace más lenta y de otra el modelo resultante crece y se hace más pesado de manipular. Estos algoritmos aproximan el autómata minimal, pero esto no quiere decir que el modelo resultante sea pequeño. Al realizar inferencia de autómatas no deterministas, se pretende disminuir (puede ser que exponencialmente) el tamaño del autómata resultante y también aumentar la precisión del modelo, con respecto a los resultados que se obtienen infiriendo autómatas deterministas. No es el objetivo principal disminuir el tiempo de construcción del modelo.

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 (Denis et al., 2004), (Coste & Fredouille, 2003a).

4. Notación y definiciones básicas

Con el fin de facilitar la lectura de la presentación de los algoritmos, se proponen a continuación algunas definiciones importantes y también algunas explicaciones sobre la notación utilizada. Las definiciones no contenidas en esta sección, pueden encontrarse en el trabajo de Hopcroft et al. (2001). Definiciones y antecedentes referentes a los autómatas residuales RFSAs, pueden encontrarse en los trabajos de Denis et al. (2000, 2001, 2004).

Un monoide libre es un conjunto sobre el cual está definida una operación binaria, tal que el resultado de aplicar la operación a dos elementos del conjunto produce otro elemento del conjunto; además, la operación binaria debe ser asociativa y el conjunto debe tener un elemento neutro. Sea A un alfabeto finito y sea A^* el monoide libre generado por A con la concatenación como operación interna y **?** como elemento neutro. Un *lenguaje* L sobre A es un subconjunto de A^*. Los elementos de L se llaman *palabras*. La longitud de

una palabra se denota como $|w|$. Dado $x \square A^*$, si $x = uv$ con $u, v \square A^*$, entonces u (o v) se denomina *prefijo* (o *sufijo*) de x. Pr (L) [o Suf (L)] denota el conjunto de prefijos (o sufijos) de L. El producto de dos lenguajes $L_1, L_2 \subseteq A^*$ se define como $L_1 \cdot L_2 = \{u_1u_2 \mid u_1 \square L_1 \wedge u_2 \square L_2\}$. Algunas veces, $L_1 \cdot L_2$ se denotará simplemente como L_1L_2. A lo largo de este artículo, el orden lexicográfico en A^* se denotará como $\square$. Suponiendo que A está totalmente ordenado por $<$ y dados $u, v \square A^*$ con $u = u_1 \ldots u_m$ y $v = v_1 \ldots v_n$, entonces $u \square v$ si y sólo si $|u| < |v|$ o $|u| = |v|$ y $\square j, 1 \leq j \leq n, m$ tal que $u_1 \ldots u_j = v_1 \ldots v_j$ y $u_{j+1} < v_{j+1}$.

Un autómata no determinista (NFA) es una 5-tupla $M = (Q, A, \delta, I, F)$ donde Q es el conjunto (finito) de estados, A 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 $Q \times A$ en 2^Q. La extensión de esta función a palabras también se denota como δ. Una palabra x es aceptada por M si $\delta(I, x) \cap F \neq \varnothing$. El conjunto de palabras aceptadas por M se denota como L(M). Un autómata determinista (DFA) es un NFA tal que $|I| = 1$ y $|\delta(q,a)| \leq 1, \square q \square Q, \square a \square A$. Dos autómatas son equivalentes si ellos reconocen el mismo lenguaje.

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

Dado un conjunto finito de palabras D_+, el *árbol aceptor de prefijos de* D_+ se define como el autómata $M = (Q, A, \delta, q_0, F)$ donde $Q = \Pr(D_+)$, $q_0 = \varepsilon, F = D_+$ y $\delta(u,a) = ua, \square u, ua \square Q$.

Una máquina de Moore es una 6-tupla $M = (Q, A, B, \delta, q_0, \Phi)$, donde Q es el conjunto de estados, A (o B) es el alfabeto de entrada (o salida), δ es una función parcial que va de $Q \times A$ en Q, q_0 es el estado inicial y Φ es una función que va de Q en B, llamada *función de salida*. A lo largo de este artículo $B = \{0,1,?\}$. Una máquina de Moore no que δ va de $Q \times A$ en 2^Q y que el conjunto de estados iniciales es I. El autómata asociado a una máquina de Moore $M = (Q, A, B, \delta, I, \Phi)$ es $M = (Q, A, \delta, I, F)$ donde $F = \{q \square Q : \Phi(q) = 1\}$. 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 $t_M : A^* \perp B$ definida como $t_M(x) = \Phi(\delta(q_0, x))$, para todo $x \square A^*$ tal que $\delta(q_0, x)$ está definida.

Dados dos conjuntos disyuntos de palabras D_+ y D_-, se define el (D_+, D_-)-*árbol de prefijos de Moore* (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 $u, ua \square Q$ y $a \square A$. Para todo estado u, el valor de la función de salida asociado a u es 1, 0 ó ? (indefinido) dependiendo si u pertenece a D_+, a D_- ó a $Q - (D_+ \cup D_-)$, respectivamente. El tamaño de la muestra (D_+, D_-) es $\sum_W |w|$ donde $W \in D_+ \cup D_-$.

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

4.1 Autómatas de estado finito residuales

La *derivada* de un lenguaje L por una palabra u, también llamada *lenguaje residual* de L con respecto a u es $u^{-1}L = \{v \square A^* : uv \square L\}$. Un lenguaje residual $u^{-1}L$ es *compuesto* si $u^{-1}L = \square_{(v^{-1}L \subseteq u^{-1}L)} v^{-1}L$.

Un lenguaje residual es *primo* si no es compuesto, es decir, si no es igual a la unión de los lenguajes residuales que contiene estrictamente.

Si $A = (Q, A, \delta, I, F)$ es un NFA y $q \square Q$, se define el lenguaje aceptado por M desde el estado q como $L(M,q) = \{v \square A^* : \delta(q,v) \cap F \neq \varnothing\}$.

Un autómata de estado finito residual (RFSA) es un autómata $A = (Q, A, \delta, I, F)$ tal que, para cada $q \square Q, L(M,q)$ es un lenguaje residual del lenguaje L reconocido por M (Denis et al., 2001). Así, $\square q \square Q, ?u \square A^*$ tal que $L(M,q) = u^{-1}L$. En otras

palabras, un autómata de estado finito residual M es un autómata no determinista tal que todos sus estados definen lenguajes residuales de $L(M)$.

Se ha demostrado que para cada lenguaje regular existe un único RFSA canónico que lo representa. Formalmente, dado un lenguaje $L \subseteq A^*$, el RFSA *canónico* de L es el autómata $M=(Q,A,\delta,I,F)$ donde:

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

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 y sea $u, v \in \Pr(D_+)$; se dice que $u \,?\, v$ si no existe una palabra w tal que $uw \in D_+$ y $vw \in D_-$. Se dice que $u \,\mathbf{?}\, v$ (u y v son estados no obviamente diferentes) si $u \,?\, v$ y $v \,?\, u$ (notación introducida por Gold).

4.2 Ejemplo de RFSA

El siguiente ejemplo ha sido tomado del trabajo de Denis et al. (2001). Sea $A = \{0,1\}$ y sea $L = A^*0A$. Los tres autómatas de la Figura 1 reconocen a L. Los estados con valor de salida 1 (o 0) se han dibujado con líneas gruesas (o delgadas).

En el primer autómata de la Figura 1, M_1 no es ni DFA ni RFSA. Los lenguajes residuales asociados a los estados son $L(M_1,1) = A^*0A$, $L(M_1,2) = A$ y $L(M_1,3) = \varepsilon$. Se puede ver que $L(M_1, 3) = \varepsilon$ y $\nexists u \in A^*$ tal que $L(M_1,3) = u^{-1}L$.

El autómata M_2 de la Figura 1 reconoce a L y es el DFA minimal, por lo tanto es un RFSA. En este caso, $L(M_2,1) = A^*0A$, $L(M_2,2) = A^*0A + A$, $L(M_2,3) = A^*0A + A + \varepsilon$, $L(M_2,4) = A^*0A + \varepsilon$.

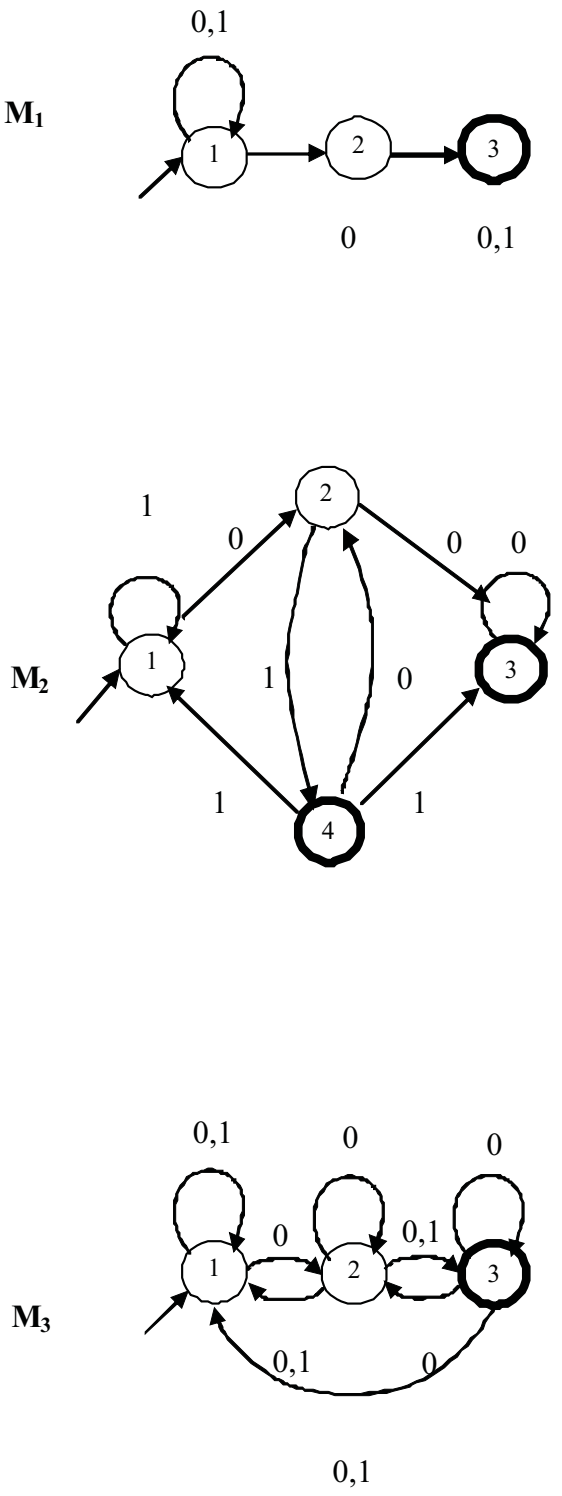

Figura 1. M_1 es un autómata que reconoce a $L=A^*0A$ pero no es ni NFA ni RFSA. M_2 es un DFA que reconoce a L y también es un RFSA. M_3 es el RFSA canónico para L.

El autómata M_3 de la Figura 1 es el RFSA canónico de L, el cual no es un DFA. Los lenguajes residuales asociados con los estados son: $L(M_3, 1) = \varepsilon^{-1}L$, $L(M_3, 2) = 0^{-1}L$ y $L(M_3, 3) = 01^{-1}L$.

En los dos algoritmos que se presentarán en las secciones siguientes se infieren RFSAs. Ambos utilizan la relación de inclusión entre lenguajes residuales como elemento básico en su estrategia de solución. La relación de inclusión se da cuando el lenguaje residual asociado a un estado x está contenido en el lenguaje residual asociado a un estado y, y se denota $x ? y$. Esto significa que todos los sufijos que concluyen en estados de aceptación a partir de x también existen a partir de y y también llegan a estados de aceptación. En el algoritmo *DeLeTe2*, las relaciones de inclusión conocidas se van registrando y se aplica la propiedad transitiva para descubrir nuevas relaciones y posiblemente también equivalencias ($x ? y$ y $y ? x$). El algoritmo *MRIA* surge de la siguiente reflexión: si $x ? y$ y todos los prefijos de y se convierten en prefijos de x esto no afectará el conjunto de cadenas aceptadas por el autómata, ya que simplemente se estarán generando caminos alternativos para aceptar cadenas que ya venían siendo aceptadas. Sin embargo, estos arcos nuevos, sí modificarán el comportamiento posterior del autómata ante nuevas cadenas, una vez ellos participen en agrupamientos de estados. De este modo, las relaciones de inclusión modificarán el proceso de generalización del autómata causado por el agrupamiento de estados.

5. Algoritmo *DeLeTe2*

El algoritmo *DeLeTe2* se presenta en el trabajo de Denis et al. (2004) como la implementación de una mejora a su predecesor, el algoritmo *DeLeTe*. Ambos son algoritmos de inferencia gramatical para lenguajes regulares, cuyas hipótesis de inferencia se expresan mediante RFSAs. *DeLeTe2* pretende mejorar la falta de consistencia de las hipótesis generadas por *DeLeTe* con respecto a las muestras de entrenamiento. No se conoce ninguna publicación que suministre detalles acerca de cómo está implementado el algoritmo *DeLeTe2* 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.

En primer lugar se observa, que aunque teóricamente *DeLeTe2* es una evolución de *DeLeTe*, no se encuentran rastros del segundo en el primero. Las características más importantes del algoritmo *DeLeTe2*, según lo que se puede desprender del código fuente que se conoce son las siguientes:

• 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 positivas 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 dichos lenguajes, lo cual es novedoso.

• El algoritmo propone hipótesis de inferencia representadas mediante RFSAs que siempre son consistentes con las muestras de entrenamiento, mejorando así la falencia 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, sus resultados son pobres.

• 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 minimal del lenguaje objetivo.

• Experimentalmente, se ha observado que el tamaño de las hipótesis converge al tamaño del DFA minimal del lenguaje objetivo. Esto se contradice con la expectativa de procurar acercarse al RFSA canónico.

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, 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 el caso en 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 ? y y x ?/ 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 asignación de 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 el caso en 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 en ciertas condiciones, los estados cuyos lenguajes residuales son superconjuntos se pueden considerar sucesores). Cuando se termina de comparar un estado con todos los demás, se verifica la consistencia del autómata actual con las muestras positivas; en el momento en que se obtenga consistencia, el algoritmo termina. DeLeTe2 se muestra en el Algoritmo 1.

Algoritmo 1. DeLeTe2 (M, D_+,D_-)

Require M = (Q, Σ, {0, 1 , ?}, δ, q_0, F)

```
1: states = Q   // En recorrido por anchura
2: Inicializar relaciones de no inclusión
3: for all q1 ☐ states do
4:       if q1 está activo then
5:             for all q2 ☐ states, q2 « q1 do
6:                   if q2 está activo then
7:                         if q1 ≠ q2 then
8:                               if get_relation(q1,q2) = desconocida then
9:                                     put_inclusion(M, q1, q2)
10:                              end if
11:                              if get_relation(q2,q1) = desconocida then
12:                                    put_inclusion(M, q2, q1)
13:                              end if
14:                        end if
15:                  end if
16:            end for
17:            if consistenciaPositiva(M, D+) then
18:                  Return M
19:            end if
20:      end if
21: end for
22: Return M
```

Cada estado almacena sus relaciones de inclusión y no inclusión con respecto a todos los demás estados del autómata: esta información se inicializa marcando las relaciones de no inclusión obvias en el autómata inicial: si $\Phi(x) = 1$ y $\Phi(y) = 0$ entonces $x \not? y$ (línea 2). La propagación de la relación desconocida que se supone relación de inclusión (líneas 8-9 y 11-12) es la parte más costosa del algoritmo y se realiza en la función *put_inclusion*. La función *put_inclusion* transforma *M* cuando realiza agrupamientos de estados, pero sobre todo modifica el registro de relaciones de inclusión asociado con cada estado. Esta función compara iterativamente los estados $q1$ y $q2$ que son sus parámetros de entrada, contra cada uno de los demás estados ($q3$), con el fin de inferir nuevas relaciones de *inclusión* o de *no inclusión* a partir de las relaciones conocidas hasta el momento. Durante ese proceso también se verifica que no haya inconsistencias; en el caso en que se detecten, todo el procesamiento da marcha atrás, ya que la relación de inclusión que inicialmente se supuso no es posible. Este proceso conjuga dos elementos importantes: cuando se establece una relación de inclusión, ella puede incrementar el conocimiento de las relaciones conocidas en los casos que se describen en las Tablas 1 y 2. Las nuevas relaciones de inclusión o de no inclusión son propagadas en el autómata con el mismo procedimiento que la pareja original.

Las relaciones de inclusión se propagan hacia adelante en el autómata (hacia los hijos) mientras que las relaciones de no inclusión lo hacen hacia atrás (hacia los padres). Una vez se han propagado todos los efectos de la nueva relación se verifica si la nueva información obtenida permite el agrupamiento de algún par de estados, si es el caso, se realiza la fusión.

Las Tablas 1 y 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 la Tabla 1 se presentan los casos en los que se pueden detectar nuevas relaciones de no inclusión y en la Tabla 2 los casos en los cuales se pueden detectar nuevas relaciones de inclusión. Todas las deducciones aplican la propiedad transitiva de la relación de inclusión.

Una cota superior de la complejidad en tiempo de la función *put_inclusion* es $O(|Q|^2)$, 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. Nótese que aunque algunas veces este método deshace decisiones tomadas, el algoritmo no realiza retro-rastreo (*backtracking*).

Tabla 1. Casos a considerar para relaciones de no inclusión suponiendo que la relación inicial que se quiere probar es $q1 \not? q2$.

Si	Entonces
$q1 ? q3 \wedge q3 ? q2$	$q1 ? q2 \wedge$ deshacer
$q1 ? q3 \wedge \neg (q3 \not? q2)$	$q3 \not? q2$
$q3 ? q2 \wedge \neg (q1 ? q3)$	$q1 \not? q3$

Tabla 2. Casos a considerar para relaciones de inclusión suponiendo que la relación inicial que se quiere probar es $q1 ? q2$.

Si	Entonces
$q2 ? q3 \wedge q1 \not? q3$	$q1 \not? q2 \wedge$ deshacer
$q3 ? q1 \wedge q3 \not? q2$	$q1 \not? q2 \wedge$ deshacer
$q2 ? q3 \wedge \neg (q1 \not? q3)$	$q1 ? q3$
$q3 ? q1 \wedge \neg (q3 ? q2)$	$q3 ? q2$
$q1 \not? q3 \wedge \neg (q2 \not? q3)$	$q2 \not? q3$
$q3 \not? q2 \wedge \neg (q3 \not? q1)$	$q3 \not? q1$

Algoritmo 2. MRIA(D_+,D_-)

```
1: M := PTA(D+)
2: S = {}
3: list := {q0,q1,…, qr} // estados de M en orden lexicográfico, q0 = λ
4: for q ∈ list do
5:       agrupado = false
6:       for p « q (en orden lexicográfico) do
7:                Compare(M, p ,q)
8:                if ¬ agrupado then
9:                         Compare(M, q, p)
10:               end if
11:               if ¬ agrupado y p? q y q? p then
12:                        Merge(M, p, q)
13:                        agrupado = True
14:               end if
15:      end for
16:      if agrupado then
17:               Q = Q – {q}
18:      else
19:               S = S ∪ {q}
20:               H := Autómata inducido por S en M
21:               if H es consistente con D + , D– then
22:                        Return H
23:               end if
24:      end if
25: end for
26: H := Autómata inducido por S en M
27: Return H
```

Simplemente, si el procesamiento causa una inconsistencia, deshace el procesamiento que la causó, pero continúa realizando las siguientes comparaciones sin reintentar.

6. Algoritmo MRIA

El algoritmo *MRIA* (*Merging Residuals Inference Algorithm*, Alvarez et al., 2006) es un algoritmo de mezcla de estados que, a partir de una muestra completa, produce como salida un autómata de estado finito no determinista (RFSA) consistente con la entrada.

El Algoritmo 2, que comienza construyendo el árbol aceptor de prefijos de las muestras positivas, consta de dos ciclos anidados: el ciclo externo itera sobre cada estado *q* del árbol en orden lexicográfico y 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* ? *q*); para ello, se invoca la función *Compare* (*M*,*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* (*M*,*q*,*p*). Si la respuesta es positiva, los estados *p* y *q* son agrupados mediante la función *Merge* (*M*,*p*,*q*). Vale resaltar que el agrupamiento no se propaga; así que, en este caso, no se mantiene el determinismo. 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*. Nótese que aún 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. Para controlar este aspecto, se utiliza la variable booleana *agrupado* en el algoritmo.

La relación de inclusión entre estados se implementa con la función *Compare* (*M*,*p*,*q*) (ver Algoritmo 3) que envía al estado *p* todas las transiciones que llegan al estado *q*, las cuales permanecerán en el caso en que el autómata resultante sea consistente con las muestras negativas. Nótese también que si la relación se cumple y *q* es el estado inicial, *p* se vuelve inicial.

El agrupamiento de estados se realiza en la función *Merge* (*M*,*p*,*q*) (ver Algoritmo 4). 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 remitidas a *p*. Si *q* es estado inicial, *p* se convierte en estado inicial.

Nótese que el estado *q* no es eliminado en este momento, ya que es posible que aún deba ser comparado con otros estados anteriores a él en el orden lexicográfico. El estado será eliminado una vez se completen las comparaciones, y sólo en el caso en que haya sido agrupado con algún otro estado.

Algoritmo 3. Compare (A,p,q)
Require: A = (Q, $\sum$, δ, *I*, *F*), p, q $\in$ Q
I ' = I
δ ' = δ
for all (r, a, q) $\in$ δ **do**
    δ := δ $\cup$ {(r, a, p)}
end for
if q $\in$ I **then**
    I := I $\cup$ {p}
end if
if A no es consistente con D_ **then**
    I = I '
    δ = δ '
end if

Algoritmo 4. Merge (A, p ,q)
Require: A = (Q, $\sum$, δ, *I*, *F*)
if p y q son compatibles (i.e. p $\notin$ F $\vee$ q $\in$ F) **then**
    if q es final **then**
        F = F $\cup$ {p}
        for all (r, a, q) $\in$ δ **do**
            δ := δ $\cup$ {(r, a, p)}
            δ := δ $-$ {(r, a, q)}
        end for
        for all (q, a, s) $\in$ δ **do**
            δ := δ $\cup$ {(p, a, s)}
            δ := δ $-$ {(q a, s)}
        end for
        if q $\in$ I **then**
            I : = I $\cup$ {p}
        end if
        if q $\in$ F **then**
            F := F $\cup$ {p}
        end if
    end if
end if

7. Comparación entre los algoritmos DeLeTe2 y MRIA

Los dos algoritmos en estudio, *DeLeTe2* y *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 dos 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 por qué el algoritmo *MRIA* es mucho más simple que *DeLeTe2* y por lo tanto tiene menor complejidad temporal y espacial que éste.

Una cota superior de la complejidad en tiempo de *DeLeTe2* es $O(|Q|^4)$, ya que el ciclo principal, que se puede ver en el Algoritmo 1 tiene un costo $O(|Q|^2)$ e internamente realiza a lo sumo dos ejecuciones de la función *put_inclusion* [cuyo costo en el peor caso se puede estimar en $O(|Q|^2)$, ya que a partir de un par de estados, al propagar los efectos de la relación de inclusión establecida, se recorrerá todo el autómata ($O(|Q|)$) y en cada caso se buscan transitividades con todos los estados del autómata ($O(|Q|)$)]. El costo espacial del algoritmo es cuadrático en $|Q|$ ya que se almacenan explícitamente todas las relaciones de inclusión entre estados.

Una cota superior de la complejidad en tiempo del algoritmo *MRIA* es $O(|Q|^3)$, ya que el ciclo principal tiene complejidad cuadrática y el costo de la comparación y el agrupamiento son lineales. Este algoritmo también es lineal en espacio con respecto a la misma variable $|Q|$, es decir, el número de estados del autómata.

El criterio de terminación en ambos algoritmos es el encuentro de una hipótesis consistente con las muestras de entrenamiento. En ambos casos se garantiza que al final esta consistencia se obtiene; sin embargo, ambos intentan obtener la consistencia antes del final de las comparaciones, porque esto permite proponer un modelo más pequeño, lo cual, según el criterio de Occam es preferible.

Se han realizado experimentos con el fin de analizar el comportamiento de los dos algoritmos. El programa que implementa el algoritmo *DeLeTe2* ha sido suministrado por sus autores. Los datos de entrenamiento y prueba son los mismos utilizados por Denis et al. (2004). Las características del conjunto de muestras aplicado son las siguientes:

• Los lenguajes objetivo son expresiones regulares y NFAs.

• La distribución de probabilidad de los métodos de generación de muestras son diferentes en cada caso: expresiones regulares o NFAs.

Tabla 3. Resultados de inferencia obtenidos con los algoritmos MRIA y DeLeTe2.

Identificador	*MRIA*		*DeLeTe2*	
	Tasa de reconocimiento	Tamaño promedio	Tasa de 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 %	40.30	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 %	65.00	74.82 %	149.13
nfa_150	75.94 %	100.96	77.14 %	218.26
nfa_200	78.85 %	125.53	79.42 %	271.30

• El método de generación de expresiones regulares considera el conjunto de operadores: $Op = \{\varnothing, 0, 1, *, \cdot, +\}$. Se escoge una cota superior para el número de operadores utilizados y se define una distribución de probabilidad p sobre Op. El operador raíz se escoge mediante la distribución p. Si el operador es 0-ario la expresión termina, si es 1-ario el procedimiento se llama recursivamente con parámetro $n_{op}-1$ y si es binario se llama dos veces con parámetros $[n_{op}/2]$ y $[n_{op}-1/2]$, respectivamente. Los valores de los parámetros usados en este experimento son: $n_{op} = 100, p_? = 0.02, p_0 = p_1 = 0.05, p_* = 0.13, p_\cdot = 0.5$ y $p_+ = 0.25$.

• El método de generación de NFAs escoge aleatoriamente el número de estados n, el tamaño del alfabeto $|\Sigma|$, el número de transiciones por estado n_δ y la probabilidad de que un estado sea inicial o final. Cada estado tiene exactamente n_δ sucesores. El símbolo y estado destino de cada transición se escogen aleatoriamente. Los valores de los parámetros usados en este conjunto de experimentos fueron los siguientes: $n = 10$, $|\Sigma| = 2, n_\delta = 2, p_I = p_F = 0.5$.

La Tabla 3 reporta dos tipos de experimentos, dependiendo de la fuente de las muestras de entrenamiento y prueba: er_* si las muestras se etiquetaron a partir de lenguajes representados mediante expresiones regulares y nfa_* si vienen de NFAs. El número en el identificador del experimento representa el número de muestras de entrenamiento. Cada experimento consta de 30 lenguajes diferentes a ser aprendidos. Cada lenguaje se prueba con un conjunto de 1,000 muestras diferentes a las de entrenamiento. La Tabla 3 reporta la tasa de reconocimiento y el tamaño promedio de la hipótesis inferida. Estos resultados se calculan como sigue. Cada hipótesis de inferencia etiqueta cada muestra de prueba como perteneciente o no al lenguaje que ella acepta. Si esta clasificación coincide con la clasificación real de dicha muestra, se considera un acierto y se incrementa un contador. Al final, el número de aciertos se divide entre el número total de muestras de prueba y este es el valor reportado como tasa de reconocimiento. El tamaño promedio se calcula sumando el número de estados de las hipótesis generadas y dividiendo por el número de lenguajes aprendidos, en este caso 30.

Vale la pena mencionar que los resultados se obtienen usando una variación de *MRIA* que busca evitar la sobregeneralización en el proceso de inferencia. Este heurístico de poda se adicionó luego de observar que las tasas de reconocimiento del algoritmo base 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.

La Tabla 3 resume el desempeño de los algoritmos *DeLeTe2* y *MRIA*. Se puede observar que *MRIA* obtiene tasas de reconocimiento muy similares a *DeLeTe2*, mientras produce hipótesis de inferencia cuyo tamaño es más pequeño. En el caso de los NFAs, el tamaño se reduce en más de un 50 %. Se observa también que los NFAs son los lenguajes objetivo donde *MRIA* obtiene sus mejores resultados comparativos, tanto respecto a tasas de desempeño como en cuanto a disminución en el tamaño de las hipótesis.

8. Conclusiones

La inferencia de NFAs es posible a pesar de las dificultades teóricas que conlleva. Las ventajas de este tipo de inferencia se encuentran en la mayor precisión de los modelos obtenidos, si se compara con métodos de inferencia de DFAs y en ciertos tipos de lenguajes regulares que se expresan como expresiones regulares o NFAs. Esta mayor precisión se refleja en mejores tasas de reconocimiento.

Los algoritmos *DeLeTe2* y *MRIA* infieren RFSAs; sin embargo, ya que tienen estrategias claramente diferentes, su complejidad y sus resultados prácticos también difieren en forma sustancial. *MRIA* tiene menor complejidad teórica, tanto temporal como espacial y aunque no supera las tasas de reconocimiento de *DeLeTe2*, obtiene

tasas muy similares con hipótesis de tamaño hasta 50 % inferior.

DeLeTe2 mejora en tasa de reconocimiento a los algoritmos de inferencia de DFAs para estos tipos de lenguajes regulares; *MRIA* mejora a *DeLeTe2* en cuanto al tamaño de las hipótesis, al tiempo que mantiene la mejora en tasa de reconocimiento con respecto a los DFAs. Sin embargo, las pruebas se realizaron con lenguajes objetivo de 100 estados aproximadamente y en ellos el tiempo de inferencia pasó de unos pocos seg para 50 muestras de entrenamiento a más de 1 h para 200 muestras, así que estos algoritmos aún no están en condiciones de resolver algunos problemas reales con lenguajes objetivo de cientos o miles de estados en tiempos de cómputo razonables.

9. Referencias bibliográficas

Abela, J., Coste, F., & Spina, S. (2004). Mutually compatible and incompatible merges for the search of the smallest consistent DFA. *Lecture Notes in Computer Science* 3264, 28–39.

Alvarez, G. I., Ruiz, J., Cano, A., & García, P. (2005). Non-deterministic regular positive negative inference NRPNI. En: J. F. Díaz, C. Rueda, & A. A. Buss (editors), *XXXI Conferencia Latinoamericana de Informatica CLEI 2005*, Cali, p. 239–249.

Alvarez, G., García, P., & Ruiz, J. (2006). A merging states algorithm for inference of RFSAs. *Lecture Notes in Computer Science* 4201, 340-341.

Angluin, D. (1988). Queries and concept learning. *Machine Learning* 2 (4), 319–342.

Angluin, D. (1990). Negative results for equivalence queries. *Machine Learning* 5 (2), 121–150.

Angluin, D. (1992). *Computational learning theory: survey and selected bibliography*. In Proceedings of the 24th Annual ACM Symposium on Theory of Computing, p. 351–369.

Cano, A., Ruiz, J., & García, P. (2002). Inferring subclasses of regular languages faster using RPNI and forbidden configurations. *Lecture Notes in Computer Science* 2484, 754–758.

Carrasco, R. C., & Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. *Lecture Notes in Computer Science* 862, 139–152.

Cicchello, O., & Kremer, S. C. (2003). Inducing grammars from sparse data sets: a survey of algorithms and results. *The Journal of Machine Learning Research* 4, 603–632.

Coste, F., & Fredouille, D. (2000). Efficient ambiguity detection in C-NFA, a step towards the inference of non deterministic automata. *Lecture Notes in Computer Science* 1891, 243–244.

Coste, F., & Fredouille, D. (2003a). Unambiguous automata inference by means of state-merging methods. *Lecture Notes in Computer Science* 2837, 60–71.

Coste, F., & Fredouille, D. (2003b). What is the search space for the inference of non-deterministic, unambiguous and deterministic automata? *Internal Report, Project Symbiose, INRIA*, RR-(4907), p. 1–41.

Coste, F., Fredouille, D., Kermorvant, C., & De la Higuera, C. (2004). Introducing domain and typing bias in automata inference. *Lecture Notes in Computer Science* 3264, 115–126.

Coste, F., & Kerbellec, G. (2005). A similar fragments merging approach to learn automata on proteins. *Lecture Notes in Computer Science* 3720, 522-529.

De la Higuera, C. (2005). A bibliographical study of grammatical inference. *Pattern Recognition*, 38 (9), 1332–1348.

Denis, F., Lemay, A., & Terlutte, A. (2000). Learning regular languages using non deterministic finite automata. *Lecture Notes in Computer Science* 1891, 213–214.

Denis, F., Lemay, A., & Terlutte, A. (2001). Residual finite state automata. *Lecture Notes in Computer Science* 2010, 144–157.

Denis, F., Lemay, A., & Terlutte, A. (2004). Learning regular languages using RFSAs. *Theoretical Computer Science* 313 (2), 267–294.

Dupont, P., Miclet, L., & Vidal, E. (1994). What is the search space of the regular inference?. *Lecture Notes in Computer Science* 862, 25–37.

Dupont, P. (1996). Incremental regular inference. *Lecture Notes in Computer Science* 1147, 222–237.

García, P., Cano, A., & Ruiz, J. (2000). A Comparative study of two algorithms for automata identification. *Lecture Notes in Computer Science* 1891, 257–258.

García, P., Ruiz, J., & Cano, A. (2004a). Identification of residual finite state automata. *Internal Report, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, España.*

García, P., Ruiz, J., & Cano, A. (2004b). Non deterministic RPNI. *Internal Report, Departamento de Sistemas Informáticos y Computación (DSIC-II/10/04), Universidad Politécnica de Valencia, España.*
http://www.dsic.upv.es/docs/bib-dig/informes/etd-07012004-122345/LATICGI.pdf

García, P., Ruiz, J., Cano, A., & Alvarez, G. (2005). Inference improvement by enlarging the training set while learning DFAs. *Lecture Notes in Computer Science* 3773, 59–70.

García, P., Ruiz, J., Cano, A., & Alvarez, G. (2006). Is learning RFSAs better than learning DFAs?. *Lecture Notes in Computer Science* 3845, 343–344.

Gold, E. M. (1967). Language identification in the limit. *Information and Control* 10 (5), 447–474.

Gold, E. M. (1978). Complexity of automaton identification from given data. *Information and Control* 37 (3), 302–320.

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). *Introduction to automata theory, languages and computation*. Second edition, Addison Wesley.

Lang, K. J. (1992). *Random DFAs can be approximately learned from sparse uniform examples*. In Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory, p. 45–52.

Lang, K. J., Pearlmutter, B. A., & Price, R. A. (1998). Results of the Abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science* 1433, 1–12.

Oncina, J., & García, P. (1992). Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis* 2 (1), 49–61.

Parekh, R., & Honavar, V. (1997). Learning DFA from simple examples. *Lecture Notes in Computer Science* 1316, 116–131.

Parekh, R., & Honavar, V. (2000). Grammar inference, automata induction, and language acquisition. In: R. Dale, H. Moisl, H. Somers (editors), *Handbook of natural language processing*, Marcel Dekker, p. 746-785.

Pernot, N., Cornuéjols, A., & Sebag, M. (2005). *Phase transitions within grammatical inference*. In Proceedings of the 2005 International Joint Conferences on Artificial Intelligence (*IJCAI 05)*, Edimburg, p. 811–816.

Polák, L. (2005). Minimalizations of NFA using the universal automaton. *Lecture Notes in Computer Science* 3317, 325-326.

Trakhtenbrot, B. A., & Barzdin, Ya. M. (1973). *Finite automata: behaviour and synthesis*. Amsterdam: North Holland Publishing Company.

Yokomori, T. (1994). Learning non-deterministic finite automata from queries and counterexamples. *Machine Intelligence* 13, 169–189.