RECONOCIMIENTO DE LENGUAJES REGULARES DESCONOCIDOS EN TIEMPO POLINOMIAL

DIEGO RAMÍREZ AND THOMAS GÓMEZ

1. Introducción

En el siguiente documento se explorará un algoritmo presentado en el paper *Learning Regular Sets from Queries and Counterexamples* de *Dana Angluin* para encontrar autómatas regulares que reconozcan un lenguaje regular dado una muestra del lenguaje regular. En particular, el artículo tiene dos objetivos: explicar de manera concisa y clara el funcionamiento del algoritmo y su relación con la teoría de la computación; también entender el desempeño en la práctica del algoritmo. Lo anterior se logrará con la siguiente estructura: primeron se desarrollarán los conceptos más relevantes para entender el algoritmo; segundo, se explicará el funcionamiento del algoritmo; tercero, se presentarán los puntos clave de una implementación del algoritmo; luego se ofrecerán ejemplos relevantes de la aplicación de este algoritmo; y finalmente presentaremos nuestras conclusiones.

2. Conceptos Clave

A. Aprendiz. En este documento, el algoritmo de aprendizaje cuyo objetivo es reconocer el lenguaje regular se lamará .ªprendiz".

B. Maestro. La fuente de ejemplos y contraejemplos del lenguaje regular la llamaremos Maestro. Usando al Maestro se entrena al aprendiz.

B.1. Maestro Mínimo Adecuado. Un Maestro Mínimo Adecuado tiene el objetivo de ayudar al Aprendiz a reconocer el lenguaje, sin "dar demasiada ayuda". Con lo último, nos referimos a que las capacidades del Maestro deben ser razonables en un contexto práctico. Decimos que un maestro mínimo adecuado debe ser capaz de responder en tiempo polinomial:

1. $MEMBER(\omega)$ Una *consulta de pertenencia* de una cadena $\omega \in \Sigma^*$ al lenguaje regular desconocido.
2. $EQUIV(M)$ Una *conjetura* que consiste de una descripción M de un lenguaje regular S. Si el lenguaje regular y S son el mismo, el algoritmo responde que sí; si el lenguaje regular y

S son diferentes, se devuelve un *contraejemplo* (una cadena en la diferencia simétrica).

Nótemos que, para que un Maestro sea capaz de responder una conjetura, es necesario que los lenguajes regulares se presenten en un cierto lenguaje; en este caso, los lenguajes regulares se presentarán como autómatas finitos deterministas. La idea es que, suponiendo que existe un Autómata Finito Determinista de n estados que reconoce el Lenguaje Regular desconocido, el Maestro T^* responde cada pregunta en tiempo polinomial sobre n.

La razón por la cuál necesitamos que un Maestro responda las conjeturas se debe a que está demostrado que el problema de aprendizaje usando sólo preguntas del primer tipo es NP-Difícil.

C. Tablas de Observación. Dado un lenguaje regular desconocido U sobre el alfabeto Σ, el Aprendiz A^* tiene la información de sobre cadenas que pertecen y no pertenecen a U, organizada en una *Tabla de Observación*, compuesta por:

C.1. Conjunto Cerrado por Prefijos. Sea $\omega \in \Sigma^*$, p es un prefijo de ω si y solo si $\exists x : \omega = px$.Decimos que $S \subseteq \Sigma^*$ es cerrado por prefijos si y solo si cada prefijo de cada miembro del conjunto, es también miembro del conjunto. Por ejemplo, si tenemos el conjunto:

$$S = \{abba, aa\}$$

el conjunto más pequeño que es cerrado por prefijos es:

$$S^* = \{\lambda, a, ab, abb, abba, aa\}$$

(Nóte que λ está en todo conjunto cerrado por prefijos no vacío.)

C.2. Conjunto Cerrado por Sufijos. $E \subseteq \Sigma^*$ es definido análogamente.

C.3. Mapa. Una función finita $T : (S \cdot E) \cup (S \cdot \Sigma \cdot E) \longrightarrow \{0, 1\}$. Es decir, para una cadena $\omega = sae$ dónde $s \in S$ (s un prefijo), $a \in \Sigma \cup \{\epsilon\}$ (a es un carácter ó es vacía) y $e \in E$ (e es un sufijo), $T(\omega = 1)$ si y solo si $\omega \in U$.

La tabla la denotamos por la tupla (S, E, T). Una forma intuitiva de ver esta tabla, es una matriz con filas $S \cup (S \cdot A)$ y columnas E, dónde la entrada de la fila s y columna e es $T(s \cdot e)$.

C.4. Tabla de Observación Cerrada. La idea de las filas de la tabla de observación es que cada una represente un estado del autómata finito determinista (cada entrada de la fila distingue experimentos para estos estados). Una tabla de observación es cerrada si $\forall t \in S \cdot A, \exists s \in S : fila(t) = fila(s)$, es decir, todos los estados que conocemos están en S.

C.5. Tabla de Observación Consistente y Cerrada. Una tabla de observación es consistente si para $\forall s_1, s_2 \in S$, se cumple que $fila(s_1) = fila(s_2)$ entonces $fila(s_1 a) = fila(s_2 a)$. (Los estados de aceptación se mantienen consistentes).

3. Algoritmo de Aprendizaje

A. Autómata Inducido por la Tabla de Observación. A partir de una tabla de observación consistente y cerrada (S, E, T) se puede generar el autómata $M(S, E, T)$ sobre Σ de la siguiente manera:

$$Q = \{fila(s) : s \in S\}$$

$$q_0 = fila(\epsilon)$$

$$F = \{fila(s) : s \in S, T(s) = 1\}$$

$$\delta(fila(s), a) = fila(s \cdot a)$$

Todas las propiedades en la construcción de la tabla son usadas para demostrar varias propiedades del autómata. Esto con la meta final de demostar que cualquier autómata que sea consistente con T pero que no es equivalente con $M(S, E, T)$ debe tener más estados.

B. Funcionamiento del Aprendiz. Describiremos el funcionamiento del aprendiz en el siguiente pseudocódigo:

Esencialmente, lo que está haciendo el algoritmo es completar (S, E, T) dependiendo del problema. Si (S, E, T) es inconsistente, entonces se debe añadir un sufijo para que $fila(s_1) \neq fila(s_2)$. Si (S, E, T) no es cerrado, entonces se tiene que añadir a S el estado que está en $S \cdot A$ pero no está en S. Finalmente, una vez tenemos una tabla consistente y cerrada, computamos el autómata, el cual le preguntamos al maestro si es correcto o no. Si no es correcto, se toma un contraejemplo y se corrige la tabla.

Algorithm 1 Aprendiz

$S := \{\lambda\}$
$E := \{\lambda\}$
Construct the initial observation table (S, E, T) using $MEMBER(s)$
while true **do**
    while (S, E, T) no es cerrado o no es consistente: **do**
        if (S, E, T) is not consistent **then**
            sea $s_1, s_2 \in S, a \in A, e \in E$ t.q. $row(s_1) = row(s_2)$ but $T(s_1ae) \neq T(s_2ae)$
            añadir ae a E
        else
            sea $s_1 \in S, a \in A$ t.q. $\nexists s_2 \in S : fila(s_1a) = fila(s_2)$
            añadir s_1a a S
        end if
        actualizar (S, E, T) usando $MEMBER(s)$
    end while
    $c := EQUIV(M(S, E, T))$
    if $c = true$ **then**
        return M
    else
        añadir c y sus prefijos a S
        actualizar (S, E, T) usando $MEMBER(s)$
    end if
end while

B.1. Es Correcto. Suponiendo que el Maestro sea Mínimo Adecuado, entonces es fácil ver que si el algoritmo retorna algún autómata, entonces es correcto.

B.2. Termina y es Eficiente. Observemos que, sea n el número de valores de las diferentes filas de la tabla, cualquier autómata consistente con la tabla (S, E, T) debe tener al menos n estados.

Cuando $s \cdot a$ es añadido a S porque la tabla no es cerrada, se está añadiendo un estado. Cuando $a \cdot e$ es añadido a E porque la tabla no es consistente, se está añadiendo un estado. Cuando se encuentra que la conjetura M es falsa, se está añadiendo *por lo menos* un estado. Luego, las repeticiones del loop interno y el loop externo suman a lo sumo $n - 1$. Ya que cada operación del algoritmo se puede hacer en tiempo finito, el algoritmo termina.

Sea n el número de estados de una máquina que reconoce a U, y m la longitud máxima de un contraejemplo presentado por el Maestro, queremos ver que el algoritmo termina en tiempo polinomial sobre n, m.

Nótemos que ya que solamente se agregan cadenas a E cuando la tabla es inconsistente, el tamaño de E es a lo sumo n. Cada vez que la tabla no es cerrada, se agrega un elemento a S, y cuando la conjetura es falsa, se agrega a lo sumo m elementos a S. Luego el tamaño de S es a lo sumo mn, y el tamaño de $S \cup S \cdot \Sigma$ es a lo sumo $mn(|\Sigma| + 1)$. Luego el número de entradas de la tabla es a lo sumo $mn^2(|\Sigma| + 1)$, el cuál es el número total de llamados a $MEMBER(s)$ que se hacen. Ya que el tamaño de las cadenas representadas en la tabla es $O(n + m)$, el tamaño total de la tabla es a lo sumo $O(mn^3 + m^2n^2)$.

Aunque se puede implementar de manera más eficiente, verificar que (S, E, T) es cerrado se puede hacer de la siguiente manera:

Algorithm 2 Cerradura

```
1: procedure ES_CERRADO(S,E,T)
2:     for sa ∈ S · Σ do
3:         for x ∈ S do
4:             if fila(sa) = fila(x) then
5:                 break
6:             end if
7:         end for
8:         if No se encontró x ∈ S : fila(x) = (sa) then
9:             return sa
10:        end if
11:    end for
12:    return true
13: end procedure
```

El procedimiento evidentemente es correcto y termina. Hay a lo sumo $|\Sigma|mn$ elementos en $S \cdot \Sigma$ y mn elementos en S; luego hay a lo sumo $m^2n^2|\Sigma|$ comparaciones. Ya que hay a lo sumo n elementos en cada fila, entonces cada comparación es de tiempo $O(m + n)$. Luego el procedimiento ocurre en tiempo $O(m^3n^2 + m^2n^3)$. Este procedimiento es llamado a lo sumo $n - 1$ veces.

Por otro lado, verificar que (S, E, T) es consistente se puede hacer de la siguiente manera:

Algorithm 3 Consistencia

```
1: procedure ES_CONSISTENTE(S,E,T)
2:     for  s_1, s_2 ∈ S : s_1 ≠ s_2 do
3:         if fila(s_1) = fila(s_2) then
4:             for a ∈ Σ do
5:                 for e ∈ E do
6:                     if T(s_1 a e) ≠ fila(s_2 a e) then
7:                         return a,e
8:                     end if
9:                 end for
10:            end for
11:        end if
12:    end for
13:    return true
14: end procedure
```

El loop más externo se repite a lo sumo $\binom{nm}{2} \leq n^2m^2$ veces, el loop intermedio $|\Sigma|$ veces y el loop interno a lo sumo n veces. Las operaciones dentro del loop interno se pueden hacer en tiempo $O(m)$. Luego el procedimiento toma $O(m^3n^3)$. Este procedimiento es llamado a lo sumo $n-1$ veces.

Entonces, sea $k_1, k_2 : MEMBER(s) \in TIME(m^{k_1}n^{k_2})$ y $k_3, k_4 : EQUIV(M) \in TIME(m^{k_3}n^{k_4})$, tenemos que la complejidad temporal del algoritmo Aprendiz es: $O(m^{k_1+1}n^{k_2+2}+m^{k_3}n^{k_4+1}+m^3n^4)$. Si asumimos que $m \leq p(n)$ para algún polinomio p, entonces $L^* \in P(n)$.

Finalmente, cabe notar que el número de estados del autómata M retornado por L^* es mínimo.

4. Implementación

A. El Maestro. Aunque en el artículo de Angluin se habla de Maestros estocásticos (es decir, que con gran probabilidad llegan a una respuesta correcta), en este documento se utlizará un Maestro estocástico. El Maestro en este caso, podrá llamar a un autómata determinista que reconozca el lenguaje que se quiere aprender para determinar si un miembro hace o no parte del lenguaje, y siempre que los autómatas no sean equivalentes, podrá en tiempo polinomial extraer un elemento w de la diferencia simétrica de longitud a lo sumo n^2-1, lo cuál lo

convierte en determinista.

```
class Teacher:
    def member(self, w: str):
        return self.dfa.accept(w)

    def equiv(self, M: DFA):
        return symmetric_difference(self.dfa, M)
```

El método de diferencia simétrica devuelve un elemento w de la diferencia simétrica si no es vacía; en caso de ser vacía, el método retorna $None$. Esta implementación utiliza la misma idea que el punto 3 de la tarea 4: para ver si dos autómatas son diferentes, se prueban todas las cadenas relevantes para ver si en algún momento un autómata acepta la cadena pero el otro no. Sin embargo, la manera en la que se implementa en la Tarea 4 implica que necesitamos probar todas las cadenas de longitud a lo sumo $n^2 - 1$, lo cuál hace que el tiempo del algoritmo sea de complejidad temporal $\Omega(2^{n^2})$.

Para lograr computar la diferencia en tiempo polinomial, el algoritmo hace uso de una tabla dinámica, donde se guardan las combinaciones de estados (q_1, q_2) ya visitados; de esta manera, si volvemos a esa combinación de estados, sabemos que no debemos visitar las cadenas que tienen como prefijo la cadena actual. De esta manera, se hacen a lo sumo n^2 llamadas de $\_\_symmetric\_difference$.

Ya que las operaciones dentro del método (diferentes a la recursión) son de complejidad a lo sumo $O(n)$, entonces el algoritmo tiene complejidad $O(n^3)$.

```
def __symmetric_difference(dfa1: DFA, dfa2: DFA, q0_1,
    q0_2, Q: dict)->str:
    if (q0_1 in dfa1.F) != (q0_2 in dfa2.F):
        return ""
    Q[(q0_1, q0_2)] = True
    Sigma = list(dfa1.d[dfa1.q0].keys())
    for a in Sigma:
        q1 = dfa1.d[q0_1][a]
        q2 = dfa2.d[q0_2][a]
        if (q1,q2) not in Q:
            s = __symmetric_difference(dfa1,dfa2,
                q1, q2, Q)
            if s != None:
                return a+s
```

```
    return None

def symmetric_difference(dfa1: DFA, dfa2: DFA) -> str:
    return __symmetric_difference(dfa1, dfa2, dfa1.q0,
        dfa2.q0, {})
```

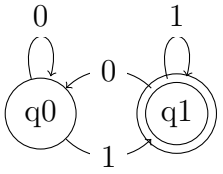
B. El Aprendiz. La implementación del aprendiz se sigue en su mayor parte del pseudocódigo. Lo único que cabe destacar es la implementación de la inducción del autómata. Aunque la idea es esencialmente la misma que en la teoría, para implementarlo en la práctica se necesitó de algunos pasos extra.

La implementación de la función de transición fue hecha con diccionarios, lo cuál implica que los estados de un autómata deben ser objetos de tipo *hashable*, por lo cuál no se podían utilizar directamente las filas (las cuales son listas) como estados del autómata. Para esto, se enumeraron las filas, y utilizamos los enumeradores como los estados del automata.

```
def __induct_DFA(S: list, E: list, T: dict,
Sigma: list) -> DFA:
    i = 0
    q0 = 0
    Q = []
    F = {}
    d = {}
    for s in S:
        q = [T[s+e] for e in E]
        if q not in Q:
            Q.append(q)
            if T[s]:
                F[i] = q
            transitions = {}
            for a in Sigma:
                transitions[a] = [T[s+a+e] for e in E]
            d[i] = transitions
            i+= 1
    new_Q = list(range(len(Q)))
    for i in d:
        for a in d[i]:
            d[i][a] = Q.index(d[i][a])
    return DFA(new_Q,q0,list(F.keys()),d)
```

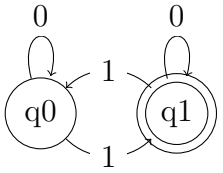
5. Ejemplos

A. Ejemplo 1. Sea $L_1 = \{w \in \{0,1\}^* : \text{w termina en } 1\}$. Un autómata que reconoce este lenguaje es:



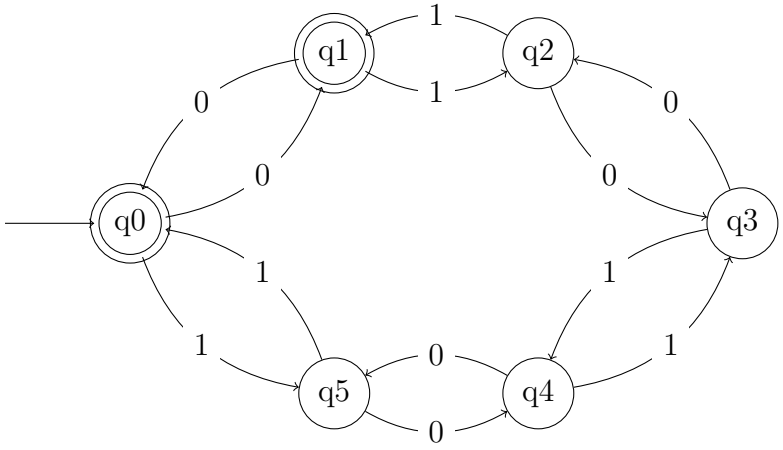
Como es de esperar, el Aprendiz devuelve un autómata isomorfo a este.

B. Ejemplo 2. Sea $L_2 = \{w \in \{0,1\}^* : \text{w tiene un número impar de } 1s\}$. Un autómata que reconoce este lenguaje es:

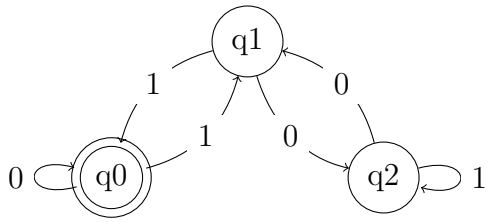


Como es de esperar, el Aprendiz devuelve un autómata isomorfo a este.

C. Ejemplo 3. Sea $L_2 = \{w \in \{0,1\}^* : \text{el número en binario que w representa es divisible entre 3}\}$. Un autómata que reconoce este lenguaje es:

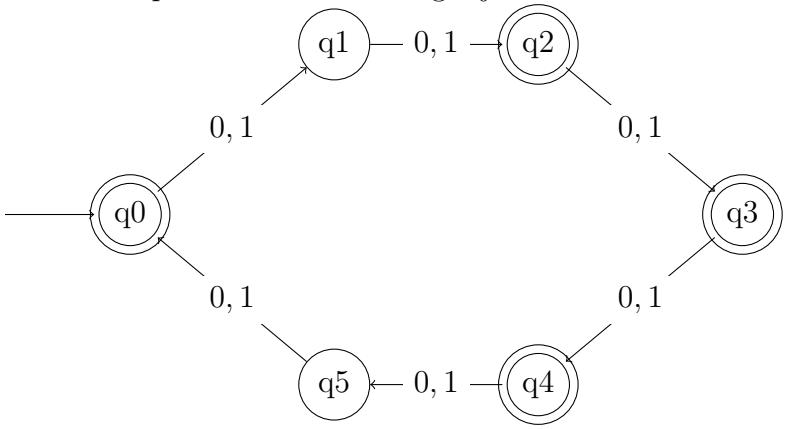


Por su parte, el Aprendiz retorna el autómata:



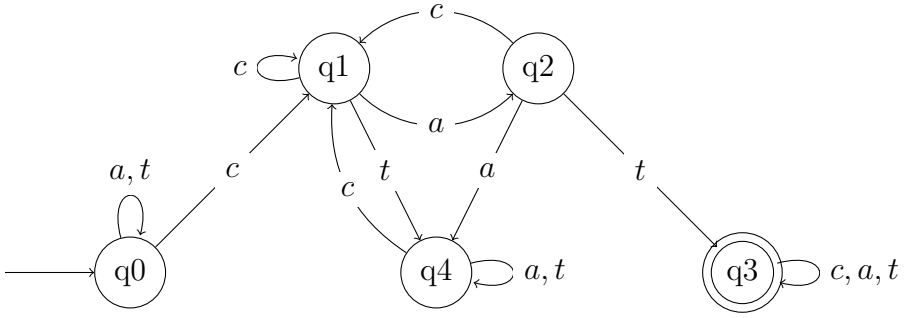
Este autómata es equivalente al primer autómata y tiene menos estados.

D. Ejemplo 4. Sea $L_4 = \{w \in \{0,1\}^* : |w| = 6n+k, k \in \{0,2,3,4\}\}$. Un autómata que reconoce este lenguaje es:

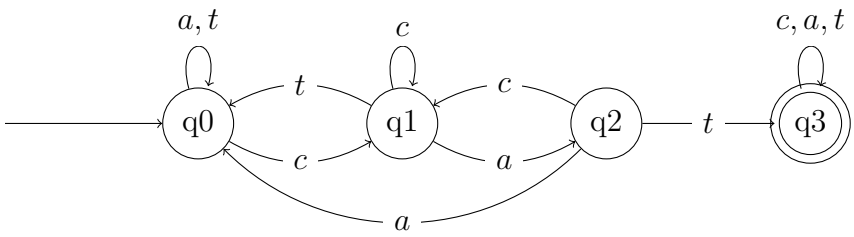


Como es de esperar, el Aprendiz devuelve un autómata isomorfo a este.

E. Ejemplo 5. Sea $L_4 = \{w \in \{c,a,t\}^* : w$ contiene la subcadena $cat\}$. Un autómata que reconoce este lenguaje es:



Por su parte, el Aprendiz retorna el autómata:



Este autómata es equivalente al primer autómata y tiene menos estados.

6. Discusión y Conclusiones

Aunque sea NP-Difícil encontrar un autómata finito determinista que reconozca un lenguaje regular desconocido U usando unicamente un oráculo de pertenencia, como pudimos ver a lo largo del documento, es mucho más fácil encontrar el autómata finito determinista si se tiene un oráculo de diferencia simétrica. Esto es útil para aplicaciones de procesamiento de grandes volumenes de datos que pueden ser estructurados como parte de un lenguaje regular (ya que el procesamiento de lenguajes regulares es lineal sobre la entrada). Además, esta respuesta es óptima, ya que devuelve el autómata con menor número de estados posibles (luego es eficiente en el tamaño).

Referencias

[1] Dana Angluin (1987), Learning Regular Sets from and Counterexamples, Information and Computation 75 pages 87-108.

[2] E. Mark Gold (1978). Complexity of automaton identification from given data, Information and Control 31 pages 302-320.

[3] Diego Ramirez (2022). Tarea 4 Introducción a la Teoría de la Computation

[4] Michael Sipser (2013) Introduction to the Theory of Computation Third Edition, Cenage Learning.

Departamento de Matemáticas, Universidad de los Andes, Bogotá, Colombia

Email address: d.ramirezg@uniandes.edu.co

Departamento de Matemáticas, Universidad de los Andes, Bogotá, Colombia

Email address: t.gomezs2@uniandes.edu.co