Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2223 - Teoría de Autómatas y Lenguajes Formales

Ayudantía 4

Franco Bruña y Dante Pinto
10 de Septiembre, 2021

Pregunta 1

1. Usando el Teorema de Myhill-Nerode, demuestre que el lenguaje $L = \{ww^r \mid w \in \Sigma^*\}$ no es regular.

   Sabemos para que un lenguaje sea regular, debemos tener una cantidad finita de clases de equivalencia bajo las relaciones de Myhill-Nerode, por lo que si queremos demostrar que el lenguaje no es regular, bastará con que encontremos una cantidad infinita de clases de equivalencia.

   Para este lenguaje, podemos considerar la relación $\equiv_L$ definida como $u \equiv_L v$ si $\forall w \in \Sigma^* : u \cdot w \in L \leftrightarrow v \cdot w \in L$.

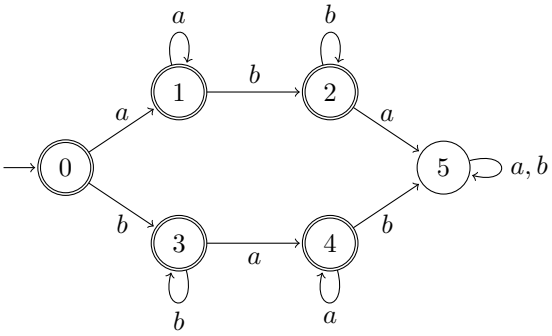
   Luego, podemos tomar la palabra $u_n = a^n b$ (para un n cualquiera); es claro que $w_n = a^n bba^n$ estará en el lenguaje, sin embargo, si concateno a u_n cualquier otra cosa por la derecha, este no será el caso. De la misma manera, si ahora considero la parte derecha de la palabra ba^n, si quiero concatenarla a alguna palabra u tal que $u \cdot ba^n \in L$, es claro que la única palabra que cumplirá esto será $u = u_n$, lo que significa entonces que $[a^n b]_L = \{a^n b\}$

   Finalmente, dado que podemos definir u_n para cada n natural, necesitaremos infinitas clases de equivalencia para representar los u_n, lo que significa $\equiv_L$ tendrá una cantidad infinita de clases de equivalencia y por tanto el lenguaje L no será regular.

2. Considere el lenguaje L dado por la expresión regular $a^*b^* + b^*a^*$. Construya una expresión regular para cada clase de equivalencia de la relación $\equiv_{\mathcal{A}}$.

   Recordando la definición de $\equiv_{\mathcal{A}}$, tenemos que $u \equiv_{\mathcal{A}} v$ si $\hat{\delta}(q_0, u) = \hat{\delta}(q_0, v,$ por lo que buscamos expresiones regulares asociadas a cada estado del autómata $\mathcal{A}$.

   Para esto, debemos encontrar un DFA mínimo y luego revisar las clases de equivalencia para las palabras dadas por cada uno de sus estados. Un DFA que define el lenguaje es:



El autómata definido es mínimo, lo que puede demostrarse usando el algoritmo de minimización, pues al aplicarlo sobre el autómata, nos daremos cuenta que todos los estados son distinguibles.

Luego, para encontrar las clases de equivalencia tomamos una palabra cualquiera que termina su ejecución en cada estado y definimos:

- $[\varepsilon]_{\mathcal{A}} = \varepsilon$
- $[a]_{\mathcal{A}} = a^+$
- $[b]_{\mathcal{A}} = b^+$
- $[ab]_{\mathcal{A}} = a^+b^+$
- $[ba]_{\mathcal{A}} = b^+a^+$
- $[aba]_{\mathcal{A}} = (a^+b^+a \ + \ b^+a^+b)\Sigma^*$

Pregunta 2 (I1 2018)

Sean L y R dos lenguajes. Decimos que L es un prefijo de R si para cada palabra $u \in R$ existe una palabra $v \in L$ tal que v es un prefijo de u.
Escriba un algoritmo que recibe como input dos autómatas A y B y responde `TRUE` si $L(A)$ es prefijo de $L(B)$ y `FALSE` en otro caso. Explique la correctitud de su algoritmo.
Para una posible solución podemos introducir el símbolo $\preceq_p$ tal que, con $u, v \in \Sigma^*$, $u \preceq_p v$ ssi u es prefijo de v. Con esta intuición, decimos que dos lenguajes L y R,

$$L \preceq_p R \ \leftrightarrow \ \forall v \in R.\ \exists u \in L.\ u \preceq_p v,$$

es decir $L \preceq_p R$ ssi L es prefijo de R. Es claro entonces que

$$\neg(L \preceq_p R) \ \leftrightarrow \ \exists v \in R.\ \forall u \in L.\ \neg u \preceq_p v.$$

A partir de esta noción podemos definir un nuevo lenguaje a partir de L y R:

$$L \not\preceq_p R = \{v \in R \mid \forall u \in L.\ \neg u \preceq_p v\}.$$

La idea será crear un autómata $\mathcal{C}$ a partir de $\mathcal{A}$ y $\mathcal{B}$ tal que $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \not\preceq_p \mathcal{L}(\mathcal{B})$, usar el algoritmo de *non emptiness* para verificar si $\mathcal{L}(\mathcal{C}) \neq \varnothing$ y en base a eso entregar una respuesta. Esto nos dará la respuesta buscada ya que, si existe al menos una palabra en este lenguaje, significa que existe alguna palabra de R que no tiene un prefijo en L y por ende L no será prefijo de R. Así dados

$$\mathcal{A} = (Q_{\mathcal{A}}, \Sigma, \delta_{\mathcal{A}}, q_0^{\mathcal{A}}, F_{\mathcal{A}})$$
$$\mathcal{B} = (Q_{\mathcal{B}}, \Sigma, \delta_{\mathcal{B}}, q_0^{\mathcal{B}}, F_{\mathcal{B}}),$$

definimos un nuevo autómata $\mathcal{C}$:

$$\mathcal{C} = (Q_{\mathcal{A}} \times Q_{\mathcal{B}}, \Sigma, \delta, (q_0^{\mathcal{A}}, q_0^{\mathcal{B}}), Q_{\mathcal{A}} \times F_{\mathcal{B}}),$$

tal que

$$\delta((p_1, q_1), a) = (\delta_{\mathcal{A}}(p_1, a), \delta_{\mathcal{B}}(p_2, a)) \leftrightarrow p_1 \notin F_{\mathcal{A}}.$$

Luego de esto se debe demostrar que $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \not\preceq_p \mathcal{L}(\mathcal{B})$. El algoritmo que se pide es el siguiente:

function PREF_LENG($\mathcal{A}, \mathcal{B}$)
  $\mathcal{C} \leftarrow$ construir $\mathcal{C}$ tal que $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{A}) \not\preceq_p \mathcal{L}(\mathcal{B})$
  if `non_empty`($\mathcal{C}$) **then**
    return `FALSE`
  end if
  return `TRUE`
end function

Pregunta 3

Sea L un lenguaje regular sobre el alfabeto Σ. Demuestre que el siguiente lenguaje:

$$L^{\exists n} = \{w \in \Sigma^* \mid \exists n \in \mathbb{N}.\ w^n \in L\}$$

es regular usando autómatas finitos en dos direcciones.

Observando el lenguaje $L^{\exists n}$ y comparándolo con el lenguaje original L, podemos ver que las palabras w en $L^{\exists n}$ son tales que al concatenarlas una cantidad arbitraria (n) de veces consigo mismas, estas forman una palabra de L.

Además, sabemos que L es un lenguaje regular, lo que significa que existe un DFA A que lo define. Por lo anterior, intuitivamente, podemos pensar que el autómata para $L^{\exists n}$ deberá ejecutar el autómata original A sobre la palabra w y, si llega al final de la palabra, deberá volver a leerla, continuando la ejecución desde donde se quedó (una cantidad arbitraria de veces).

Dado que estamos trabajando con autómatas en dos direcciones, otra manera de ver lo anterior es pensar que ejecutaremos el autómata A hasta que se acabe la palabra, "pausaremos" la ejecución, haremos *rewind* de la palabra hasta volver al principio y continuaremos la ejecución.

Para hacer lo anterior, crearemos una copia de cada estado del autómata, que representarán los estados *rewind*, y haremos que desde cualquier estado, al leer la marca final, el autómata pase al estado *rewind* equivalente, retroceda hasta leer la marca inicial y luego vuelva al estado original para continuar la ejcución.

La construcción asociada a esto, asumiendo que $A = (Q, \Sigma, \delta, q_0, F)$ es:

$A' = (Q', \Sigma, \vdash, \dashv, \delta', q_0, F'$
$Q' = Q \uplus Q_r \uplus q_f \quad \wedge \quad Q_r = q_r \mid q \in Q$
$F' = \{q_f\}$
Sea $a \in \Sigma$, δ' :

- $\delta'(q_0, \vdash) = (q_0, \rightarrow)$
- $\delta'(q, a) = (\delta(q, a), \rightarrow)$
- $\delta'(q, \vdash) = (q_r, \leftarrow)$, $q \notin F$
- $\delta'(q, \vdash) = (q_f, \rightarrow)$, $q \in F$
- $\delta'(q_r, a) = (q_r, \leftarrow)$

- $\delta'(q_r, \vdash) = (q, \rightarrow)$

Finalmente, podemos demostrar que el lenguaje que define A es equivalente a $L^{\exists n}$ de la forma tradicional (usando las ejecuciones). Para una demostración completa de este proceso, pueden revisar la pregunta 2 de la ayudantía 3.

¿El autómata encontrado termina su ejecución para todas las palabras?. Si no es el caso, diseñe un algoritmo que reciba el $2DFA$ y una palabra w como input y retorne `TRUE` si el autómata acepta la palabra y `FALSE` en caso contrario.

Podemos ver que nuestro autómata no terminará sus ejecuciones para las palabras que no pertenecen al lenguaje.

Para diseñar el algoritmo que buscamos, tenemos que analizar las ejecuciones del $2DFA$. Observándolas, podemos notar que para que el autómata se quede ejecutando por siempre, esto debe ocurrir en un ciclo, pues tenemos una cantidad finita de estados, así que podríamos terminar nuestro algoritmo cuando nos topemos un ciclo, pero antes de decidir si hacer esto, debemos tener cuidad con no eliminar ejecuciones de aceptación.

Considere entonces una ejecución de acpetación cualquiera ρ, que tenga un ciclo. Esto implica entonces que en algún momento de la ejecución llegamos a una configuración (q_c, i_c), luego pasamos por otras configuraciones, volvemos a (q_c, i_c) y de ahí en adelante continuamos la ejecución hasta aceptar la palabra. Dado que q_c representa un estado, i_c representa una posición del lector y estamos trabajando con un autómata determinista; lo anterior no puede ocurrir, pues para cada estado q_c, δ definirá una única posible transición leyendo la letra de la posición i_c, por lo que de encontrar un ciclo en la ejecución, nos quedaremos en el para siempre.

Habiendo encontrado lo anterior, podemos escribir el siguiente algoritmo:

```
Eval(A, w){
q=q_0; i=1; conf={(q, i)}
while True:
    if q in F AND i=n+1:
        return True
    q, d = delta(q, a_i)
    i+=d
    if (q, i) in conf:
        return False
    conf.add(q, i)
}
```

Pregunta 4

Sea $w = a_1 a_2 \cdots a_n \in \Sigma^*$ (los a_i son letras) e $I \subseteq \{1, \ldots, |w|\}$. Diremos que la w_I denota a la palabra $a_{i_1} a_{i_2} \cdots a_{i_k}$ con $i_1 < i_2 < \cdots < i_k$ e $I = \{i_1, i_2, \ldots, i_k\}$.

Para $u, v \in \Sigma^*$ definimos el conjunto $u \uparrow v$ como:

$$u \uparrow v = \{w \in \Sigma^* \mid \exists I, J \subseteq \{1, \ldots, |w|\}.\ I \cup J = \{1, \ldots, |w|\}.\ w_I = u \text{ y } w_J = v\}.$$

Así, para dos lenguajes $L_1, L_2 \subseteq \Sigma^*$, se define

$$L_1 \uparrow L_2 = \{w \in \Sigma^* \mid \exists u \in L_1.\ \exists v \in L_2.\ w \in u \uparrow v\}.$$

Demuestre que si L_1 y L_2 son lenguajes regulares, luego $L_1 \uparrow L_2$ también es regular.

En general para resolver este tipo de problemas, primero debemos entender bien la forma que tienen las palabras del lenguaje y luego, a partir de esta forma deberíamos planear qué hacer con los autómatas originales para definir el lenguaje nuevo.
En este caso, podemos ver que si tenemos palabras $u = a_1 a_2 ... a_n$ y $v = b_1 b_2 ... b_m$, las palabras w del lenguaje tendrán la forma:

$$w = a_1 b_1 b_2 a_2 a_3 b_4 a_4 ... b_m a_n = a_1 b_1 b_2 b_3 a_3 a_4 a_4 ... b_m a_n$$

Donde toda letra de w viene de u (como a_1 y a_4), de v (como b_1 y b_4) o de ambas (como $a_2 = b_3$ y $a_4 = b_4$).
De esta forma, eligiendo los índices correctos de w, podemos proyectar la palabra con solo estos índices I y formar $w_I = u$ o, análogamente $w_J = v$.
Sabiendo esto, es mucho más claro lo que debemos hacer para encontrar un autómata que defina el lenguaje, pues bastará con que ejecutemos los autómatas originales en paralelo y decidamos de manera no determinista cuándo avanzar el primero, el segundo o ambos.
Formalizando esta idea, suponiendo que tenemos los DFA $A_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ y $A_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$, podemos construir $A = (Q_1 \times Q_2, \Sigma, \Delta, (q_{01}, q_{02}), F_1 \times F_2)$ con Δ dado por:

$$\begin{aligned}\Delta = &\left\{\big((p,q),\ a\ ,(\delta_1(p), q)\big) \mid a \in \Sigma\right\} \\ &\cup \left\{\big((p,q),\ a\ ,(p, \delta_2(q))\big) \mid a \in \Sigma\right\} \\ &\cup \left\{\big((p,q),\ a\ ,(\delta_1(p), \delta_2(q))\big) \mid a \in \Sigma\right\}\end{aligned}$$

Finalmente, podemos demostrar que el autómata define el mismo lenguaje usando sus ejecuciones. Para una demostración completa de este proceso, pueden revisar la pregunta 2 de la ayudantía 3.