Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2223 – Teoría de Autómatas y Lenguajes Formales
Segundo semestre de 2024
Profesor: Cristian Riveros
Ayudante: Amaranta Salas

Ayudantia 8

Myhill-Nerode, 2DFA y Algoritmos de Evaluación de Auntómatas

Problema 1

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

Solución

1. 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.

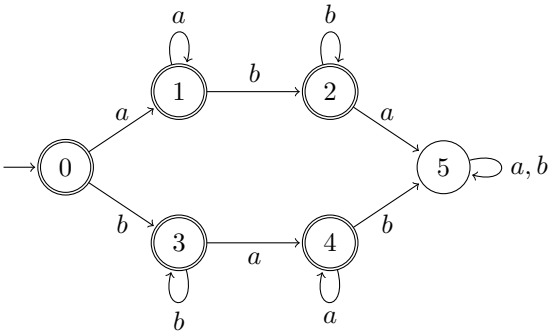
   Para este lenguaje, podemos considerar la relación $\equiv_L$ definida como $u \equiv_L v$ si $\forall w \in \Sigma^* : u \cdot w \in L \leftrightarrow v \cdot w \in L$.

   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. 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:

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

Problema 2

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.

Solución

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 $\mathcal{A}$ que lo define. Por lo anterior, intuitivamente, podemos pensar que el autómata para $L^{\exists n}$ deberá ejecutar el autómata original $\mathcal{A}$ 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 $\mathcal{A}$ 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 ejecución.
La construcción asociada a esto, asumiendo que $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ es:

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

Sea $a \in \Sigma$, δ' estará dado por:

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

Finalmente, podemos demostrar que el lenguaje que define $\mathcal{A}'$ es equivalente a $L^{\exists n}$ de la forma tradicional (usando las ejecuciones).

Problema 3

Considere el siguiente problema:

Problema:	#RUNS-NFA
Input:	Un NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ y $w \in \Sigma^*$.
Output:	$\mid \{\rho \mid \rho$ es una ejecución de aceptación de $\mathcal{A}$ sobre $w\} \mid$.

Esto es, el problema #RUNS-NFA consiste en, dado un autómata finito no-determinista $\mathcal{A}$ y dado una palabra w, contar todas las ejecuciones de aceptación de $\mathcal{A}$ sobre w. Por ejemplo, usted puede comprobar que si $w \notin L(\mathcal{A})$, entonces el output con $\mathcal{A}$ y w es 0.
Escriba un algoritmo que resuelva #RUNS-NFA en tiempo $\mathcal{O}(|\mathcal{A}| \cdot |w|)$ donde $|\mathcal{A}|$ es el número de estados y transiciones de $\mathcal{A}$. Demuestre la correctitud de su algoritmo.

Solución

Una posible solución para esta pregunta es proponiendo la siguiente modificación al algoritmo *eval-NFAonthefly* visto en clases.

En lugar de llevar dos conjuntos S y S_{old} para llevar los estados en los que van las ejecuciones al leer la i-ésima letra de w, se llevará un **contador** por cada estado que acumula la **cantidad de ejecuciones** que hay en cada estado al leer la i-ésima letra. En el nuevo algoritmo, S y S_{old} corresponderán a arreglos de tamaño $|Q| = m$, donde cada entrada del arreglo le corresponde a un estado $j \in Q$, y esta llevará la cuenta de ejecuciones que han llegado a dicho estado j.

Sin pérdida de generalidad, suponga que el autómata $\mathcal{A}$ tiene como conjunto de estados $Q = \{0, 1, \ldots, m-1\}$. Si los m estados de $\mathcal{A}$ no son números, estos pueden ser ordenados arbitrariamente y ser asignados a números. También suponemos, sin pérdida de generalidad, que $\mathcal{A}$ es un autómata sin ϵ-transiciones. Luego, escribimos el siguiente algoritmo:

#RUNS-NFA($\mathcal{A} = (Q, \Sigma, \Delta, I, F)$, $w = a_1 a_2 \ldots a_n$)

```
S, S_old ← Arreglos de largo m, y entradas con valor 0
for j = 0 to m − 1 do
    if j ∈ I then
        S[j] ← 1
    end if
end for
for i = 1 to n do
    S_old ← S
    S ← Arreglo de largo m, y entradas con valor 0
    for (j1, a_i, j2) ∈ Δ do
        S[j2] ← S[j2] + S_old[j1]
    end for
end for
c ← 0
for j ∈ F do
    c ← c + S[j]
end for
return c
```

Para la demostración de la correctitud del algoritmo, usaremos inducción sobre el tamaño de w, para mostrar que en toda iteración el arreglo S lleva correctamente la cantidad de ejecuciones que han llegado a cada estado.

CB: Antes de empezar a leer letras de w, se da que a los estados iniciales de $\mathcal{A}$ ha llegado una única ejecución, y todos los demás estados aún no les llega ninguna ejecución. Esto se ve representado correctamente al inicializar las entradas de S como 1 si le corresponden a un estado inicial, y 0 en caso contrario.

HI: Suponemos que luego de procesar $a_1 \cdots a_i$ se cumple que, para todo estado j, $S[j]$ es igual al número de ejecuciones de $\mathcal{A}$ sobre $a_1 \cdots a_i$ que llegan al estado j.

TI: Por HI, S lleva correctamente la cantidad de ejecuciones por estado en la i-ésima iteración. En la iteración $i+1$, se asigna a S_{old} los valores de S, y S se reinicializa con valores 0. Luego, por cada transición (j_1, a_{i+1}, j_2) del autómata, a $S[j_2]$ se le suma la cantidad de ejecuciones a j_1 al leer $a_1 \cdot a_i$. Al realizar esto sobre todas las transiciones del autómata, se cumple que $S[j]$ es igual al número de ejecuciones de $\mathcal{A}$ sobre $a_1 \cdots a_{i+1}$ para todo $j \in Q$.

Ya demostrado que el S lleva correctamente la cantidad de ejecuciones que llegan a cada estado para cualquier largo de w, la cantidad de ejecuciones de aceptación será la suma de la cantidad de ejecuciones que llegaron a estados finales del autómata después de leer las n letras de w. Esto se guarda en la variable c al final del algoritmo, y se retorna dicho valor. Luego, el algoritmo propuesto retorna correctamente la cantidad de ejecuciones de aceptación de $\mathcal{A}$ sobre w.