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 11

Repaso I2

Problema 1

Para una gramática libre de contexto $\mathcal{G} = (V, \Sigma, P, S)$ decimos que $\mathcal{G}$ tiene un *loop* si existe una variable $X \in V$ tal que $X \stackrel{*}{\Rightarrow} \alpha X \beta$ para algún $\alpha, \beta \in (V \cup \Sigma)^*$. Demuestre que si $\mathcal{G}$ no tiene un loop, entonces $\mathcal{L}(\mathcal{G})$ es un lenguaje regular.

Solución

Por demostrar que si una gramática libre de contexto no tiene loops, entonces define un lenguaje regular. Sea $\mathcal{G} = (V, \Sigma, P, S)$ una CFG sin loops y $G_{\mathcal{G}} = (V, E)$ un grafo tal que $|V| = n$ y $(X, Y) \in E$ si, y solo si, $X \to \alpha Y \beta \in P$.

Por demostrar que si $\mathcal{G}$ no tiene loops, entonces $G_{\mathcal{G}}$ tampoco. Por contrapositivo, si $G_{\mathcal{G}}$ tiene loops, entonces existe un camino $X = X_1 \to X_2 \to X_3 \to \cdots \to X_m = X$. Luego, existen las producciones $X_1 \to \alpha_2 X_2 \beta_2$, $X_2 \to \alpha_3 X_3 \beta_3$, $\ldots$, $X_{m-1} \to \alpha_m X_m \beta_m$. Componiéndolas se obtiene $X \stackrel{*}{\Rightarrow} \alpha X \beta$.

Como $G_{\mathcal{G}}$ no tiene loops, entonces define un grafo dirigido acíclico (DAG). Así, se puede definir la secuencia $X_1, X_2, \ldots, X_n$ de las variables en V ordenadas topológicamente tal que, con $i < j \leq n$, desde X_i no se puede llegar a X_j (al revés de la definición convencional).

Sea $\mathcal{G}_X = (V, \Sigma, P, X)$. Por demostrar, utilizando inducción fuerte, que $\forall i \in \{1, n\}.\mathcal{L}(\mathcal{G}_{X_i})$ es finito.

- Caso base: Desde X_1 no se puede llegar a ninguna variable, es decir, todas las producciones son de la forma $X \to w$ con $w \in \Sigma^*$. Como P es finito entonces, $\mathcal{L}(\mathcal{G}_{X_1})$ también lo es.
- Caso inductivo: Sea X_i una variable en la secuencia ordenada. Por hipótesis de inducción $\forall k < i.\mathcal{L}(\mathcal{G}_{X_k})$ es finito. Gracias al orden topológico y que $\mathcal{G}$ no tiene loops, entonces las producciones de X_i son de la forma $X_i \to \alpha$ con $\alpha \in (\Sigma \cup \{X_1, \ldots, X_{i-1}\})^*$. Luego, considerando las producciones de X_i de la forma $X_i \to \alpha_1 | \alpha_2 | \ldots | \alpha_l$, si se reemplaza cada aparición de X_k con $k < i$ en cada $\alpha_1, \alpha_2, \ldots, \alpha_l$ por toda posible palabra en $\mathcal{L}(\mathcal{G}_{X_k})$ se logra definir $\mathcal{L}(\mathcal{G}_{X_i})$ a través de producciones de la forma $X_i \to w$ con $w \in \Sigma^*$. Como los $\mathcal{L}(\mathcal{G}_{X_k})$ son finitos y las producciones originales de X_i son finitas, entonces las producciones con reemplazo son finitas. Por lo tanto, $\mathcal{L}(\mathcal{G}_{X_i})$ es finito.

Finalmente, $\mathcal{L}(\mathcal{G}_{X_S}) = \mathcal{L}(\mathcal{G})$ es finito y todo lenguaje finito es regular.

Problema 2

Demuestre que el siguiente lenguaje NO es regular:

$$L = \{a^n \# a^m \mid n \neq m\}$$

Solución

Si usamos el teorema de *Myhill-Nerode* nos basta con encontrar una secuencia infinita de palabras pertenecientes a Σ^* tal que para todo par $w_1 \neq w_2$ se tiene que $w_1 \notin [\![w_2]\!]_{\equiv_L}$. Ahora consideremos $A = \{a^i \mid i \in \mathbb{N} \setminus \{0\}\}$, $a^j \in A$ y $a^k \in A$ con $j \neq k$. Es claro ver que $a^j \notin [\![a^k]\!]_{\equiv_L}$ ya que si consideramos $w = \#a^k$ tenemos que:

$$a^j \cdot w \in L \wedge a^k \cdot w \notin L$$

Finalmente, como A tiene la misma cardinalidad que $\mathbb{N}$ en particular existen infinitas palabras a^i y por ende una cantidad infinita de clases de equivalencia distintas en $\equiv_L$ lo que demuestra que L no es regular.

Problema 3

Considere el siguiente problema:

Problema:	#suffix-DFA
Input:	Un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ y $w = a_1 \dots a_n \in \Sigma^*$.
Output:	$\mid \{i \in \{1, \dots, n\} \mid a_i \dots a_n \in L(\mathcal{A})\} \mid$.

Esto es, el problema #suffix-DFA consiste en, dado un autómata finito determinista $\mathcal{A}$ y dado una palabra w, contar todos los sufijos de w que son aceptados por $\mathcal{A}$.
Escriba una algoritmo que resuelva #suffix-DFA en tiempo $O(|\mathcal{A}| \cdot |w|)$ 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 modificar el algoritmo *eval-NFAonthefly* visto en clases. Específicamente, 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** de los sufijos 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 sufijos de la palabra que al ejecutar $\mathcal{A}$ llegan al estado j.
Sin pérdida de generalidad, suponga que el autómata $\mathcal{A}$ tiene como conjunto de estados $Q = \{0, 1, \dots, m-1\}$. 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:

#SUFFIX-DFA($\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, $w = a_1 a_2 \dots a_n$)

```
S, S_old ← Arreglos de largo m, y entradas con valor 0
for i = 1 to n do
    S_old ← S
    S ← [0]                                  // arreglo de 0s
    S_old[0] ← S_old[0] + 1
    for j = 0 to m − 1 do
        S[δ(j, a_i)] ← S[δ(j, a_i)] + S_old[j]
    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 sufijos de $a_1...a_{i-1}$ que llegan a cada estado.

CB: Antes de empezar a leer letras de w, se da que el estado inicial de $\mathcal{A}$ no tiene sufijos y no se ha pasado por ningún otro estado. Esto se ve representado correctamente al inicializar las entradas de S como 0.

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 sufijos de $\mathcal{A}$ sobre $a_1 \cdots a_i$ que llegan al estado j.

TI: Por HI, S lleva correctamente la cantidad de sufijos 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 sufijos de j_1. Al realizar esto sobre todas las transiciones del autómata, se cumple que $S[j]$ es igual al número de sufijos de $a_1 \cdots a_{i+1}$ al ejecutar $\mathcal{A}$ que llegan a $j \in Q$.

Ya demostrado que el S lleva correctamente la cantidad de sufijos que llegan a cada estado para cualquier largo de w. La cantidad de sufijos de w que son aceptados por $\mathcal{A}$ será la suma de la cantidad de sufijos de las 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 sufijos de w que son aceptados por $\mathcal{A}$.