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

IIC2223 – Teoría de Autómatas y Lenguajes Formales – 2' 2024
IIC2224 – Autómatas y Compiladores

Examen

Pregunta 1

Sea $\mathcal{P} = (Q, \Sigma, \Gamma, \Delta, q_0, \perp, \{q_f\})$ un autómata apilador con un único estado final y tal que todas sus transiciones en Δ son de la forma (p, a, A, q, B_1B_2) o de la forma (p, a, A, q, ϵ) con $p, q \in Q, a \in \Sigma \cup \{\epsilon\}$ y $A, B_1, B_2 \in \Gamma$. Demuestre que existe un autómata apilador $\mathcal{P}'$ con UN solo estado tal que $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}')$.

Solución

La solución corresponde a la demostración vista en la clase 27. Definiremos un nuevo PDA de un solo estado q^*:

$$\mathcal{P}' = (Q', \Sigma, \Gamma', \Delta', q^*, \perp', F')$$

de tal forma que:

- $Q' = F' = \{q^*\}$ es el único estado, inicial y final.
- Σ es el alfabeto de lectura (el mismo que $\mathcal{P}$).
- $\Gamma' = Q \times \Gamma \times Q$ será el alfabeto de stack.
  "$(p, A, q) \in \Gamma'$ si desde p leyendo A en el tope de stack llegamos a q al hacer pop de A."
- $\Delta' \subseteq (Q' \times (\Sigma \cup \{\epsilon\}) \times \Gamma') \times (Q' \times \Gamma'^*)$ es la relación de transición.
- $\perp' = (q_0, \perp, q_f)$ es el símbolo inicial del stack.
  "El autómata parte en q_0 y al hacer pop de $\perp$ llegará a q_f."

Básicamente, estamos guardando la información de los estados de $\mathcal{P}$ como parte del stack de $\mathcal{P}'$, en tuplas de forma (p, A, q), donde $p \in Q$ representaría nuestro estado actual, $A \in \Gamma^*$ el stack de $\mathcal{P}$ y $q \in Q$ el estado que "adivinamos" está por debajo en el stack de $\mathcal{P}'$. Definiremos Δ' de la siguiente manera:

$$\Delta' = \overbrace{\{(q^*, a, (p, A, q_2), q^*, (q, B_1, q_1)(q_1, B_2, q_2)) \mid (p, a, A, q, B_1B_2) \in \Delta \wedge q_1, q_2 \in Q\}}^{\text{Transiciones push}} \cup \underbrace{\{(q^*, a, (p, A, q), q^*, \varepsilon) \mid (p, a, A, q, \varepsilon) \in \Delta\}}_{\text{Transiciones pop}}$$

Ahora demostramos que tanto $\mathcal{P}$ como $\mathcal{P}'$ definen el mismo lenguaje. Para esto, demostraremos la siguiente afirmación para todo $p \in Q$, $A \in \Gamma$, $w \in \Sigma^*$ y $k \geq 1$:

$$(pA, w) \vdash_{\mathcal{P}}^{k} (q, \epsilon) \qquad \text{si y solo si} \qquad (q^*(p, A, q), w) \vdash_{\mathcal{P}'}^{k} (q^*, \epsilon)$$

Si esta afirmación es verdadera tendremos como caso particular que:

$$(q_0\perp, w) \vdash_{\mathcal{P}}^{*} (q_f, \epsilon) \qquad \text{si y solo si} \qquad (q^*(q_0, \perp, q_f), w) \vdash_{\mathcal{P}'}^{*} (q^*, \epsilon)$$

lo que significa que: $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}')$. Demostraremos lo anterior por inducción en el largo de sus ejecuciones.

Caso base Para una ejecución de un solo paso, sabemos que la única transición existente será pop. Digamos que $c \in \Sigma$ una letra cualquiera:

$$\begin{aligned}
& (pA, c) \vdash_{\mathcal{P}} (q, \epsilon) \\
\Leftrightarrow\ & (p, c, A, q, \epsilon) \in \Delta \\
\Leftrightarrow\ & (q^*, c, (p, A, q), q^*, \epsilon) \in \Delta' \quad \text{por construcción} \\
\Leftrightarrow\ & (q^*(p, A, q), c) \vdash_{\mathcal{P}'} (q^*, \epsilon)
\end{aligned}$$

Hipótesis inductiva Supongamos se cumple la siguiente propiedad para $k < n$

$$(pA, w) \vdash_{\mathcal{P}}^{k} (q, \epsilon) \qquad \text{si y solo si} \qquad (q^*(p, A, q), w) \vdash_{\mathcal{P}'}^{k} (q^*, \epsilon)$$

Paso inductivo Digamos que tenemos una palabra $w \in \Sigma^*$ tal que $|w| = n$ y $w = c \cdot u \cdot v$. Luego analicemos la ejecución por partes:

$$\begin{aligned}
& (pA, c) \vdash_{\mathcal{P}}^{n} (q, \epsilon) \\
\Leftrightarrow \quad & \underbrace{(pA, c \cdot u \cdot v) \vdash_{\mathcal{P}} (p_1 B_1 B_2, u \cdot v)} \vdash_{\mathcal{P}}^{i} (p_2 B_2, v) \vdash_{\mathcal{P}}^{j} (q, \epsilon) \\
\Leftrightarrow \quad & (p, c, A, p_1, B_1 B_2) \in \Delta \\
\Leftrightarrow \quad & (q^*, c, (p, A, q_2), q^*, (p_1, B_1, q_1)(q_1, B_2, q_2)) \in \Delta' \quad \text{por construcción} \\
\Leftrightarrow \quad & (q^*(p, A, p_1), c) \vdash_{\mathcal{P}'} (q^*, \epsilon)
\end{aligned}$$

Por otra parte, tomemos una sección de tamaño $i < n$ de la ejecución,

$$\begin{aligned}
& (pA, c) \vdash_{\mathcal{P}}^{n} (q, \epsilon) \\
\Leftrightarrow \quad & (pA, c \cdot u \cdot v) \vdash_{\mathcal{P}} \underbrace{(p_1 B_1 B_2, u \cdot v) \vdash_{\mathcal{P}}^{i} (p_2 B_2, v)} \vdash_{\mathcal{P}}^{j} (q, \epsilon) \\
\Leftrightarrow \quad & (p_1 B_1, u) \vdash_{\mathcal{P}}^{i} (p_2, \epsilon) \\
\Leftrightarrow \quad & (q^*(p_1, B_1, p_2), u) \vdash_{\mathcal{P}'}^{i} (p_2, \epsilon) \quad \text{ya que } i < n, \text{ por H.I.}
\end{aligned}$$

Similarmente, tomemos una sección final de tamaño $j < n$,

$$\begin{aligned}
& (pA, c) \vdash_{\mathcal{P}}^{n} (q, \epsilon) \\
\Leftrightarrow \quad & (pA, c \cdot u \cdot v) \vdash_{\mathcal{P}} (p_1 B_1 B_2, u \cdot v) \vdash_{\mathcal{P}}^{i} \underbrace{(p_2 B_2, v) \vdash_{\mathcal{P}}^{j} (q, \epsilon)} \\
\Leftrightarrow \quad & (p_2 B_2, v) \vdash_{\mathcal{P}}^{j} (q, \epsilon) \\
\Leftrightarrow \quad & (q^*(p_2, B_2, q), v) \vdash_{\mathcal{P}'}^{j} (q, \epsilon) \quad \text{ya que } j < n, \text{ por H.I.}
\end{aligned}$$

Finalmente, al juntar estas tres conclusiones podemos ver que

$$(q^*(p, A, q), c \cdot u \cdot v) \vdash_{\mathcal{P}'}^{n} (q^*, \epsilon)$$

Finalmente, hemos demostrado por inducción que $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}')$, ya que las ejecuciones de ambos PDA son equivalentes.

Distribución de puntaje

- **(1 pto.)** Por plantear el concepto de construcción y funcionamiento del nuevo PDA $\mathcal{P}'$.
- **(1 pto.)** Por definir correctamente el alfabeto de stack Γ'.
- **(1 pto.)** Por definir correctamente el símbolo inicial del stack $\perp'$
- **(1 pto.)** Por definir en Δ' las transiciones *push* correctamente.
- **(1 pto.)** Por definir en Δ' las transiciones *pop* correctamente.
- **(1 pto.)** Por demostrar que ambos PDA definen el mismo lenguaje, $\mathcal{L}(\mathcal{P}) = \mathcal{L}(\mathcal{P}')$

Pregunta 2

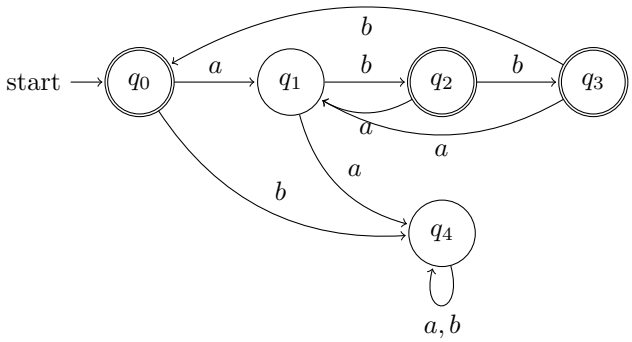
Considere la siguiente expresión regular sobre el alfabeto $\{a, b\}$:

$$R = (ab + abb + abbb)^*$$

1. Construya un autómata finito determinista $\mathcal{A}$ tal que $\mathcal{L}(\mathcal{A}) = \mathcal{L}(R)$. Explique su respuesta.
2. Demuestre que NO existe un autómata finito determinista $\mathcal{A}'$ con 4 estados tal que $\mathcal{L}(\mathcal{A}') = \mathcal{L}(R)$.

Solución

Problema 2.1. La expresión R representa cadenas en repeticiones de ab, abb y abbb. Un AFD (autómata finito determinista) que acepte esta expresión regular, debe reconocer cada uno de estos patrones. El siguiente autómata cumple con lo anterior:



Explicación: El autómata tiene cinco estados, q_0, q_1, q_2, q_3 y q_4. El estado inicial es q_0 y los estados finales son q_0, q_2 y q_3. El estado q_4 es un estado de error, que se alcanza si se lee un símbolo distinto de b o a en los estados q_0 o q_1 respectivamente. En el estado q_0 se espera leer un a para pasar al estado q_1 y luego un b para pasar al estado q_2. En el estado q_2 se espera leer un b para pasar al estado q_3 y luego un a para volver al estado q_1. En el estado q_3 se espera leer un b para volver al estado q_0 y un a para volver al estado q_1. En el estado q_4 se espera leer un a o un b para quedarse en el mismo estado.

Distribución de puntaje

- **(1 pto.)** Por aceptar ϵ en el estado inicial que además es un estado de reinicio y aceptación. Es decir, reconocer la secuencia vacía y *abbb*.
- **(1 pto.)** Por aceptar a en el estado inicial y pasar al estado q_1.
- **(1 pto.)** Por aceptar b en el estado q_1 y pasar al estado q_2. Es decir, reconocer la secuencia *ab*.
- **(1 pto.)** Por aceptar b en el estado q_2 y pasar al estado q_3. Es decir, reconocer la secuencia *abb*.
- **(1 pto.)** Por estado de rechazo en el estado q_4. Es decir, rechazar cualquier secuencia que no sea ab, abb o *abbb*.
- **(1 pto.)** Por la explicación.

Problema 2.2. El lenguaje $\mathcal{L}(R)$ tiene estructuras repetitivas que dependen de la longitud de las cadenas ab, abb, abbb. Cada uno de estos patrones requiere distinguir entre diferentes posiciones dentro de las cadenas. Con sólo 4 estados no es posible rastrear suficientes configuraciones, pues es necesario considerar las palabras que no son aceptadas como se vio anteriormente.

PD: Todas las clases de equivalencia de la relación de Myhill Nerode $\equiv_{\mathcal{L}(R)}$ son distintas, por lo que no es posible reducir el autómata.

Para esta demostración, se puede usar el concepto de clases de equivalencia de Myhill-Nerode. Este método se basa en demostrar que el lenguaje tiene más de 4 clases de equivalencia, lo que implica que no es posible reducir el autómata a menos de 5 estados.

Sabemos que dos cadenas x y y están en la misma clase de equivalencia respecto a $\mathcal{L}(R)$ si, para cualquier cadena z, la concatenación de $xz \in \mathcal{L}(R)$ implica que $yz \in \mathcal{L}(R)$ y viceversa. En otras palabras $x \equiv_{\mathcal{L}(R)} y \iff \big(\forall z \in \Sigma^*.xz \in \mathcal{L}(R) \iff yz \in \mathcal{L}(R)\big)$.

Para el lenguaje, las clases de equivalencia dependen de qué prefijos de la cadena x se alinean con los patrones ab, abb o abbb. Necesitamos distinguir cuántos estados posibles puede tener la máquina.

Los patrones posibles en $\mathcal{L}(R)$ generan 5 clases de equivalencia diferentes:

- $[\epsilon]$: Cadenas vacías o listas para empezar un patrón.
- $[a]$: Cadenas que han iniciado un patrón pero necesitan un b.
- $[ab]$: Cadenas que están en medio de un patrón válido, esperando completarse o reiniciarse.
- $[abb]$: Cadenas que casi completan un patrón válido, faltando solo un b.
- $[aa]$: Cadenas inválidas para el lenguaje porque no pueden derivar en un patrón aceptable.

Para mostrar que estas clases de equivalencia son distintas, se puede usar el siguiente argumento:

	ϵ	a	ab	abb
ϵ				
a	b			
ab	b	bbb		
abb	b	bb	bb	
aa	ab	b	b	b

En la tabla, se muestra que las clases de equivalencia son distintas, pues para cada par de cadenas, se puede encontrar una cadena z que distinga entre ellas. Por ejemplo, para las cadenas a y ab, se puede usar $z = bbb$ para distinguirlas. Para las cadenas ab y abb, se puede usar $z = bb$ para distinguirlas. Para las cadenas abb y aa, se puede usar $z = b$ para distinguirlas. Por lo tanto, como tengo por lo menos 5 clases de equivalencia de Myhill-Nerode, entonces cualquier DFA que acepte $\mathcal{L}(R)$ debe tener al menos 5 estados.

Distribución de puntaje

- **(1 pto.)** Por identificar las clases de equivalencia.
- **(1.5 ptos.)** Por comparar ab^i versus ab^j.
- **(1.5 ptos.)** Por comparar ab^i versus ϵ.
- **(1.5 ptos.)** Por comparar aa con cualquier otra cadena.
- **(0.5 ptos.)** Por la conclusión de usar la relación de Myhill-Nerode lleva a que el autómata tiene que tener más de 4 estados.

Problema 3

Una gramática libre de contexto $\mathcal{G}$ se dice *lineal* si todas sus producciones son de la forma $X \to aY$, $X \to Ya$ o $X \to a$ con $X, Y \in V$ y $a \in \Sigma$. En cambio, $\mathcal{G}$ se dice que es lineal *por la izquierda* si es lineal y NO tiene reglas de la forma $X \to aY$.

1. Demuestre que si $\mathcal{G}$ es una gramática lineal por la izquierda, entonces $\mathcal{L}(\mathcal{G})$ es regular.
2. Demuestre que existe una gramática lineal $\mathcal{G}$ tal que $\mathcal{L}(\mathcal{G})$ NO es regular.

Solución

Problema 3.1. Sea la gramática $G = (V, \Sigma, P, S)$ lineal por la izquierda. Es decir, sus producciones son de la forma $X \to Ya$ o $X \to a$. Al ser lineal por la izquierda, su árbol derivación será de la forma:

$$S \Rightarrow X_{n-1}a_n \Rightarrow X_{n-2}a_{n-1}a_n \Rightarrow ... \Rightarrow X_1a_2...a_n \Rightarrow a_1a_2...a_n$$

La idea es construir un autómata que adivinará (no determinismo) la derivación anterior desde la derecha hacia la izquierda. O sea, empezando con $X_1 \to a_1$ de ahí $X_2 \to X_1a_2$, etc, hasta llegar a $S \to X_{n-1}a_n$. Para esto, los estados del autómata serán las variables V con un estado adicional q_0 para partir.

Dado lo anterior, construimos el autómata A como sigue. Sea $A = (V \cup \{q_0\}, \Sigma, \Delta, \{q_0\}, \{S\})$, con $\Delta = \{(q_0, a, X) \mid X \to a \in P\} \cup \{(Y, a, X) \mid X \to Ya \in P\}$. Por demostrar: $\mathcal{L}(G) = \mathcal{L}(A)$.

$\mathcal{L}(G) \subseteq \mathcal{L}(A)$

Sea $w = a_1a_2...a_n \in \mathcal{L}(G)$. Con G lineal por la izquierda. La derivación de G sobre w es de la forma:

$$S \Rightarrow X_{n-1}a_n \Rightarrow X_{n-2}a_{n-1}a_n \Rightarrow ... \Rightarrow X_1a_2...a_n \Rightarrow a_1a_2...a_n$$

De esto, es claro que se puede construir una ejecución ρ de la forma:

$$\rho: \quad q_0 \xrightarrow{a_1} X_1 \xrightarrow{a_2} X_2 \longrightarrow ... \longrightarrow X_{n-1} \xrightarrow{a_n} S$$

$\mathcal{L}(A) \subseteq \mathcal{L}(G)$

La demostración es exactamente igual para este lado que el anterior, se construye la misma ejecución por construcción de Δ y se concluye la misma derivación de G.

Distribución de puntaje

- **(1 pto.)** Por la idea de cómo es el árbol de derivación de la gramática y como esto puede llevar a construir un autómata.
- **(1 pto.)** Por dar el conjunto final del autómata de manera correcta.
- **(1.5 ptos.)** Por el primer subconjunto de Δ.
- **(1.5 ptos.)** Por el segundo subconjunto de Δ.
- **(1 pto.)** Por demostrar, para ambos lados, que los lenguajes son iguales.

Problema 3.2. Sea el lenguaje $L = \{a^n b^n | n \geq 1\}$, que, como se vió en clases, es no regular.
Sea la grámatica G que lo produce tal que:

$$\begin{array}{lcl} S & \to & aX \mid aY \\ X & \to & Sb \\ Y & \to & b \end{array}$$

Como se puede notar, G es lineal y produce el lenguaje L. Por lo que queda demostrado.

Distribución de puntaje

- **(2 ptos.)** Por dar el lenguaje.
- **(2 ptos.)** Por demostrar que es no regular. Si se usa uno visto en clases, es válido no demostrarlo.
- **(2 ptos.)** Por dar la gramática que construya al lenguaje.

Problema 4

Considere el siguiente lenguaje sobre el alfabeto $\Sigma = \{a, b\}$:

$$L = \{a^n b^m a^n b^m \mid n \geq 0 \wedge m \geq 0\}.$$

Sea $L^c = \Sigma^* \backslash L$ el complemento de L.

1. ¿Es L libre de contexto? Demuestre su afirmación.
2. ¿Es L^c libre de contexto? Demuestre su afirmación.

Solución

Problema 4.1. El lenguaje **no es libre de contexto**. Para demostrar esto, usamos el lema del bombeo de lenguajes libres de contexto. Sea $N > 0$. Definimos:

$$z = a^N b^N a^N b^N.$$

Considere una partición cualquiera $uvwxy = z$ tal que: $vx \neq \varepsilon$ y $|vwx| \leq N$. Como $|vwx| \leq N$, se tiene que (1) $vwx \in \mathcal{L}(a^*b^*)$ o (2) $vwx \in \mathcal{L}(b^*a^*)$. Asimismo, tomamos $i = 2$. A continuación analizaremos uno de los casos, ya que el otro es análogo:

Sin pérdida de generalidad, suponemos que $v \in \mathcal{L}(a^*)$ y $x \in \mathcal{L}(b^*)$. Los otros casos son análogos. Entonces, se tiene:

$$\underbrace{a^j}_{u}\ \underbrace{a^k}_{v}\ \underbrace{a^l b^m}_{w}\ \underbrace{b^n}_{x}\ \underbrace{b^o a^N b^N}_{y}$$

Para algún $j, k, l, m, m, n, o \in \mathbb{N}$ tal que $j + k + l = N \wedge m + n + o = N \wedge (k \neq 0 \vee n \neq 0)$ Recordemos que escogimos $i = 2$, con lo que tenemos:

$$u\ v^2\ w\ x^2\ y = a^{j+2k+l} b^{m+2n+o} a^N b^N = a^{N+k} b^{N+n} a^N b^N$$

Como $k \neq 0$ o $n \neq 0$, entonces $N + k \neq N \vee N + n \neq N$. Por lo tanto, $uv^2wx^2y \notin L$.

Distribución de puntaje

- **(1 pto.)** Por decir que el lenguaje NO es libre de contexto.
- **(1 pto.)** Por definir z de manera correcta.
- **(1 pto.)** Por descomponer z correctamente y usar $i = 2$ o similar.
- **(1 pto.)** Por mencionar y/o analizar los dos casos, dado que $|vwx| \leq N$.
- **(1 pto.)** Por armar $uv^2wx^2y \notin L$ dado que se escogió $i = 2$ o similar.
- **(1 pto.)** Por concluir que $uv^2wx^2y \notin L$ dado que $k \neq 0$ o $n \neq 0$.

Problema 4.2. El lenguaje **es libre de contexto**. Para esto, dado que el lenguaje se forma utilizando la operación complemento, notemos que tenemos dos casos:

I) $w \neq a^*b^*a^*b^*$, es decir, no es de la forma solicitada para $\mathcal{L}$. Esto corresponde a casos como $a...ab...ba...ab...ba...$ o $b...ba...ab...ba...$, es decir, $a^*b^+a^+b^+a\Sigma$

II) $w = a^k b^l a^m b^n$ tal que $k \neq m \vee l \neq n$

Dado esto, debemos formar una gramática libre de contexto que contenga ambos casos. Un ejemplo es la siguiente gramática S, donde S_I corresponde al primer caso y S_{II} corresponde al segundo:

$$S \to S_I \mid S_{II}$$
$$S_I \to a\ S_I \mid b\ B_1$$
$$B_1 \to b\ B_1 \mid a\ A_2$$
$$A_2 \to a\ A_2 \mid b\ B_2$$
$$B_2 \to b\ B_2 \mid a\ E$$

$$E \to a\ E \mid b\ E \mid \epsilon$$

$$S_{II} \to X\ B \mid A\ Y$$

$$A \to a\ A \mid \epsilon$$

$$B \to b\ B \mid \epsilon$$

$$X \to a\ X\ a \mid a\ A\ b\ B \mid b\ B\ a\ A \mid a$$

$$Y \to b\ Y\ b \mid a\ A\ b\ B \mid b\ B\ a\ A \mid b$$

De la definición de S uno puede ver que las derivaciones de S_I serán de la forma:

$$w = a^* b^+ a^+ b^+ a \Sigma$$

y las derivaciones de S_{II} serán de la forma:

$$w = a^k b^l a^m b^n \text{ tal que } k \neq m \vee l \neq n$$

Por lo tanto, $\mathcal{L}(S) = \mathcal{L}$.

Distribución de puntaje

- **(1 pto.)** Por decir que el lenguaje es libre de contexto.
- **(0.5 ptos.)** Por el caso I.
- **(2 ptos.)** Por el caso II.
- **(0.5 ptos.)** Por las reglas de S_I.
- **(1 pto.)** Por las reglas $S_{II} \to X\ B \mid A\ Y$, es decir, por hacer un camino que permita llegar a $w = a^k b^l a^m b^n$ tal que $k \neq m$ y otro que permita llegar a $w = a^k b^l a^m b^n$ tal que $\vee l \neq n$.
- **(1 pto.)** Por el resto de las reglas de S_{II}.