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

Pauta Interrogación 2

Pregunta 1

Usando el teorema de Myhill-Nerode, demuestre que $\mathcal{L}(\mathcal{A})$ es regular para todo 2DFA $\mathcal{A}$.

Solución

Sea $\mathcal{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_f)$ un 2DFA arbitrario. Para un input $u \in \Sigma^*$ cualquiera, definiremos la función de comportamiento T_u como se hizo en clases:

$$T_u : Q \cup \{\bullet\} \longrightarrow Q \cup \{\perp\}$$

tal que

- $T_u(p) = q$ si y solo si desde $(p, |u|)$ cruza en la config. $(q, |u| + 1)$.
- $T_u(p) = \perp$ si y solo si desde $(p, |u|)$ nunca cruza de u.
- $T_u(\bullet) = q$ si y solo si desde $(q_0, 0)$ cruza por primera vez con $(q, |u| + 1)$.
- $T_u(\bullet) = \perp$ si y solo si desde $(q_0, 0)$ nunca cruza de u.

Consideremos la relación $\equiv_T$ tal que se cumpla:

$$u \equiv_T v \quad \text{ssi} \quad T_u = T_v$$

Es fácil ver que $\equiv_T$ es relación de equivalencia sobre Σ^* (es refleja, simétrica y transitiva). Por Teorema de Myhill-Nerode, demostrar que $\mathcal{L}(\mathcal{A})$ es lenguaje regular es equivalente a demostrar que $\equiv_T$ es una relación de Myhill-Nerode.

Por demostrar: $\equiv_T$ es relación de Myhill-Nerode

Debemos verificar que $\equiv_T$ cumpla las siguientes 3 propiedades:

1. $\equiv_T$ es congruencia por la derecha
2. $\equiv_T$ refine $\mathcal{L}(\mathcal{A})$
3. La cantidad de clases de equivalencia de $\equiv_T$ es finita

1. $\equiv_T$ es congruencia por la derecha

Esto implica demostrar que

$$u \equiv_T v \Longrightarrow u \cdot w \equiv_T v \cdot w \qquad \forall w \in \Sigma^*$$

Ahora, tenemos que $u \equiv_T v \Longrightarrow T_u = T_v = T$. Si $\mathcal{A}$ parte en $(p, |uw| + 1)$ sobre uw tenemos dos posibilidades:

(1) Se lee w y nunca se entra a u. En este caso $T_{uw}(p) = T_{vw}(p)$.

(2) Se lee w y entra a u en un estado q. Luego, como $T_u = T_v$ entonces el comportamiento de $\mathcal{A}$ sobre u sería igual al comportamiento sobre v. Por lo que $T_{uw}(p) = T_{vw}(p)$.

Ocurrirá algo similar en el caso de $\bullet$.

(1) Si $T_u(\bullet) = T_v(\bullet) = q$ entonces se cruza desde $(q_0, 0)$ por primera vez a la primera letra de w con estado q. Luego la lectura de uw y vw será igual en adelante, ya que su sufijo w es el mismo, por lo que $T_{uw}(\bullet) = T_{vw}(\bullet)$.

(2) Si $T_u(\bullet) = T_v(\bullet) = \perp$, entonces nunca se cruza de u ni v. Por lo tanto, tampoco cruzará de uw ni vw. Luego, $T_{uw}(\bullet) = T_{vw}(\bullet) = \perp$.

2. $\equiv_T$ refine $\mathcal{L}(\mathcal{A})$

Esto implica demostrar que

$$u \equiv_T v \implies (u \in \mathcal{L}(\mathcal{A}) \Leftrightarrow v \in \mathcal{L}(\mathcal{A}))$$

Al ser u y v indistinguibles tendremos:

(1) Si $T(\bullet) = \perp$ entonces $\mathcal{A}$ nunca sale a la derecha de u ni de v. Recordemos que para que una palabra u sea aceptada su ejecución debe terminar con $(q_f, |u| + 1)$. Por lo que tendremos que $u \notin \mathcal{L}(\mathcal{A})$ y $v \notin \mathcal{L}(\mathcal{A})$.

(2) Si $T(\bullet) = p$ entonces nos encontramos con dos casos:

(a) $p = q_f$, luego $u \in \mathcal{L}(\mathcal{A})$ y $v \in \mathcal{L}(\mathcal{A})$, ya que llegaríamos a una ejecución de aceptación.

(b) $p \neq q_f$ y $\delta(p, \dashv) = (q, \leftarrow)$, es decir, la ejecución podría continuar en la dirección opuesta. Por un lado, si $T(q) = \perp$ se repite el caso (1). Por otro, si $T(q) \in Q$, se repite el caso (2).

3. Cantidad finita de clases de equivalencia

Se cumple que

$$|\Sigma^* / \equiv_T| \leq |\{T : Q \cup \{\bullet\} \longrightarrow Q \cup \{\perp\}|\}$$
$$|\Sigma^* / \equiv_T| \leq (|Q| + 1)^{|Q|+1}$$

por lo que la cantidad de clases de equivalencia de $\equiv_T$ está acotada por un límite superior finito.
Finalmente podemos concluir que, por Teorema de Myhill-Nerode, $\mathcal{L}(\mathcal{A})$ es un lenguaje regular.

Distribución de puntajes

- **(0.5 ptos.)** Por definir correctamente la función de comportamiento T_u .
- **(2 ptos.)** Por definir correctamente la relación de equivalencia $\equiv_T$.
- **(1 pto.)** Por demostrar que $\equiv_T$ es congruencia por la derecha.
- **(1 pto.)** Por demostrar que $\equiv_T$ refine el lenguaje.
- **(1 pto.)** Por demostrar que $\equiv_T$ tiene cantidad finita de clases de equivalencia.
- **(0.5 ptos.)** Por usar el Teorema de Myhill-Nerode correctamente para demostrar lo pedido.

Pregunta 2

Demuestre una gramática libre de contexto para cada uno de los siguientes lenguajes.

1. $L_1 = \{a^i b^j c^k \mid i \neq j \vee j \neq k\}$
2. $L_2 = \{a^i b^j c^k \mid j = i + k\}$

Para cada una de las gramáticas, explique su correctitud.

Solución

1. $L_1 = \{a^i b^j c^k | i \neq j \vee j \neq k\}$
   Se define la siguiente gramática libre de contexto:

$$\begin{array}{rcl} S & \to & X_L \mid X_R \\ X_L & \to & Y_L \cdot C \\ Y_L & \to & aY_Lb \mid aA \mid bB \\ X_R & \to & A \cdot Y_R \\ Y_R & \to & bY_Rc \mid bB \mid cC \\ A & \to & aA \mid \varepsilon \\ B & \to & bB \mid \varepsilon \\ C & \to & cC \mid \varepsilon \end{array}$$

   Desglosemos la solución. Primero tenemos S que define tanto a X_L como a X_R. X_L representa cuando en la palabra se cumple que $i \neq j$, mientras que X_R cubre el otro caso $j \neq k$. Antes de ver X_L vemos los últimos tres casos A, B y C. Cada uno de estos permite generar la cantidad de veces que se quiera

la letra específica o parar de construir la palabra utilizando la palabra vacía ε. Ahora veamos Y_L. Esta parte genera aY_Lb o una cantidad de a's o b's cualquiera (usando A o B mencionadas anteriormente). Como vemos, Y_L se encarga de que cada vez que se quieran generar nuevas letras en la palabra se generen la misma cantidad de a's y de b's y solo se puede terminar la palabra si es que se escoge luego aA o bB. En esto último esta la clave, dado que con esto nos aseguramos de que por lo menos hay una a o una b más (es decir el caso $i \neq j$: $j < i$ si se elige aA y $i < j$ si se elige bB). Finalmente vemos X_L, este se encarga de generar Y_L que como sabemos es una palabra con $i \neq j$ junto a una cantidad arbitraria de c's. Debido a que X_L solo evalua el caso en que $i \neq j$ entonces la cantidad de c's que tenga es irrelevante. Analogamente tenemos lo mismo que para X_R y Y_R, que evalua el otro caso posible $j \neq k$.

- **(1 pto.)** Por establecer la diferencia entre X_L y X_R.
- **(1 pto.)** Por establecer el caso aY_Lb.
- **(1 pto.)** Por establecer el caso $aA \mid bB$ en Y_L.
- **(1 pto.)** Por establecer el caso bY_Rc.
- **(1 pto.)** Por establecer el caso $bB \mid cC$ en Y_R.
- **(1 pto.)** Por establecer los casos A, B y C.

2. $L_2 = \{a^i b^j c^k | j = i + k\}$

   Se define la siguiente gramática libre de contexto:

$$\begin{array}{rcl} S & \rightarrow & X_L \cdot X_R \\ X_L & \rightarrow & aX_Lb \mid \varepsilon \\ X_R & \rightarrow & bX_Rc \mid \varepsilon \end{array}$$

   Desglosemos la solución. Similar al inciso anterior S define a X_L y X_R, solamente que en este caso los concatena. El caso X_L se encarga de que cada vez que se genere una a tambien se genere una b. Analogamente X_R se encarga de que cada que que se genera una c también se genere una b. Finalmente, como en X_L tenemos la misma cantidad de a's que b's (la cual es i) y en X_R tenemos la misma cantidad de b's que c's (la cual es k), entonces tenemos que la concatenación S cumple que $j = i + k$.

   - **(2 ptos.)** Por establecer la concatenación entre X_L y X_R.
   - **(2 ptos.)** Por establecer el caso X_L.
   - **(2 ptos.)** Por establecer el caso X_R.

Problema 3

Sea $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ un transductor. Decimos que $\mathcal{T}$ es *síncrono* si $\Delta \subseteq Q \times \Sigma \times \Omega \times Q$. Para un lenguaje $L \subseteq \Sigma^*$, se define el lenguaje $[\![\mathcal{T}]\!](L) = \{v \in \Omega^* \mid \exists u \in L.\ (u, v) \in [\![\mathcal{T}]\!]\}$.

Demuestre que para todo transductor síncrono $\mathcal{T} = (Q, \Sigma, \Omega, \Delta, I, F)$ y para todo lenguaje regular $L \subseteq \Sigma^*$ se tiene que $[\![\mathcal{T}]\!](L)$ es regular.

Solución 1

Sabemos que $\mathcal{L}$ es un lenguaje regular, por lo que existe un autómata finito determinista que lo define, sea $\mathcal{A} = (Q', \Sigma, \delta', q_0', F')$ este autómata. Definimos $\mathcal{T}_\mathcal{A} = (Q_\mathcal{A}, \Omega, \Delta_\mathcal{A}, I_\mathcal{A}, F_\mathcal{A})$, donde:

- $Q_\mathcal{A} = Q \times Q'$.
- $\Delta_\mathcal{A} = \{((p, p'), b, (q, q')) \mid \exists a \in \Sigma^* \, [\delta'(p', a) = q' \wedge (p, a, b, q) \in \Delta]\}$.
- $I_\mathcal{A} = I \times \{q_0'\}$.
- $F_\mathcal{A} = F \times F'$.

PD. $[\![\mathcal{T}]\!](\mathcal{L}) = L(\mathcal{T}_\mathcal{A})$.

- $[\![\mathcal{T}]\!](\mathcal{L}) \subseteq L(\mathcal{T}_\mathcal{A})$

  Sea $v \in [\![\mathcal{T}]\!](\mathcal{L})$, entonces existe $u \in \mathcal{L}$ tal que $(u, v) \in \mathcal{T}$. Como $\mathcal{T}$ es síncrono, entonces $|u| = |v| = n$. Sea $u = a_1 \dots a_n$ y $v = b_1 \dots b_n$. Como $u \in \mathcal{L}$, entonces existe una ejecución ρ' de aceptación en $\mathcal{A}$ tal que:

$$\rho' : \ p'_0 \xrightarrow{a_1} p'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p'_n$$

Como $(u, v) \in \mathcal{T}$, entonces existe una ejecución ρ en $\mathcal{T}$ de aceptación tal que:

$$\rho : \ (p_0, p'_0) \xrightarrow{(a_1,b_1)} (p_1, p'_1) \xrightarrow{(a_2,b_2)} \dots \xrightarrow{(a_n,b_n)} (p_n, p'_n)$$

Por ende, la secuencia ρ'', definida como:

$$\rho'' : \ (p_0, p'_0) \xrightarrow{b_1} (p_1, p'_1) \xrightarrow{b_2} \dots \xrightarrow{b_n} (p_n, p'_n)$$

es una ejecución de aceptación en $\mathcal{T}_\mathcal{A}$, dado que $(p_i, a_i, b_i, p_{i+1}) \in \Delta$ y $\delta'(p'_i, a_i) = p'_{i+1}$ para todo i, con $0 \leq i < n$, $(p_0, p'_0) = I_\mathcal{A}$ y $(p_n, p'_n) \in F_\mathcal{A}$, por lo que $v \in L(\mathcal{T}_\mathcal{A})$.

- $L(\mathcal{T}_\mathcal{A}) \subseteq [\![\mathcal{T}]\!](\mathcal{L})$

  Sea $v = b_1 \dots b_n \in L(\mathcal{T}_\mathcal{A})$, entonces existe una ejecución ρ de aceptación en $\mathcal{T}_\mathcal{A}$ tal que:

  $$\rho : \ (p_0, p'_0) \xrightarrow{b_1} (p_1, p'_1) \xrightarrow{b_2} \dots \xrightarrow{b_n} (p_n, p'_n)$$

  Como $\mathcal{T}$ es síncrono, entonces $|u| = |v| = n$. Sea $u = a_1 \dots a_n$. Como ρ es una ejecución de aceptación en $\mathcal{T}_\mathcal{A}$, por construcción sabemos de dado que $((p_i, p'_i), b_i, (p_{i+1}, p'_{i+1})) \in \Delta_\mathcal{A}$, entonces existe $a_i \in \Sigma^*$ tal que $\delta'(p'_i, a_i) = p'_{i+1}$ y $(p_i, a_i, b_i, p_{i+1}) \in \Delta$ para todo i, con $0 \leq i < n$. Por ende, existen ejecuciones ρ' y ρ'' en $\mathcal{A}$ y $\mathcal{T}$ respectivamente, tales que:

  $$\rho' : \ p'_0 \xrightarrow{a_1} p'_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p'_n$$
  $$\rho'' : \ (p_0, p'_0) \xrightarrow{(a_1,b_1)} (p_1, p'_1) \xrightarrow{(a_2,b_2)} \dots \xrightarrow{(a_n,b_n)} (p_n, p'_n)$$

  Por ende, $(u, v) \in \mathcal{T}$ y $v \in [\![\mathcal{T}]\!](\mathcal{L})$.

Distribución de puntaje

- **(1 pto.)** Por definir $Q_\mathcal{A}$.
- **(2 ptos.)** Por definir $\Delta_\mathcal{A}$.
- **(1 pto.)** Por definir $I_\mathcal{A}$ y $F_\mathcal{A}$.
- **(1 pto.)** Por demostrar que $[\![\mathcal{T}]\!](\mathcal{L}) \subseteq L(\mathcal{T}_\mathcal{A})$.
- **(1 pto.)** Por demostrar que $L(\mathcal{T}_\mathcal{A}) \subseteq [\![\mathcal{T}]\!](\mathcal{L})$.

Solución 2

En clases se vio que si $\mathcal{T}_1$ y $\mathcal{T}_2$ son transductores, entonces $[\![\mathcal{T}_1]\!] \bigcap [\![\mathcal{T}_2]\!]$ no es necesariamente una relación racional. Sin embargo, si $\mathcal{T}_1$ y $\mathcal{T}_2$ son síncronos, entonces $[\![\mathcal{T}_1]\!] \bigcap [\![\mathcal{T}_2]\!]$ si define una relación racional.
Sea $\mathcal{T}_i = (Q_i, \Sigma, \Omega, \Delta_i, I_i, F_i)$ para $i \in \{1, 2\}$. Definimos $\mathcal{T}_1 \times \mathcal{T}_2 = (Q^\times, \Sigma, \Omega, \Delta^\times, I^\times, F^\times)$, donde:

- $Q^\times = Q_1 \times Q_2$.
- $\Delta^\times = \{((p_1, p_2), a, b, (q_1, q_2)) \mid (p_1, a, b, q_1) \in \Delta_1 \wedge (p_2, a, b, q_2) \in \Delta_2\}$.
- $I^\times = I_1 \times I_2$.
- $F^\times = F_1 \times F_2$.

La demostración de esta afirmación es análoga a la demostración del producto de autómatas, por lo que se omitirá.
A continuación, dado que $\mathcal{L}$ es regular, existe un autómata finito no determinista $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ que lo define, es decir, $L(\mathcal{A}) = \mathcal{L}$.
Definimos $\mathcal{T}_\mathcal{A} = (Q, \Sigma, \Omega, \Delta', I, F)$ como la $\mathcal{T}$-extensión de $\mathcal{A}$, donde

$$\Delta' = \{(p, a, b, q) \mid (p, a, q) \in \Delta \wedge b \in \Omega\}$$

Es claro ver que $\mathcal{T}_\mathcal{A}$ es síncrono y que $[\![\mathcal{T}_\mathcal{A}]\!] = \{(u, v) \mid u \in \mathcal{L} \wedge v \in \Omega^* \wedge |u| = |v|\}$.
Asi, como sabemos que $\mathcal{T}$ y $\mathcal{T}_\mathcal{A}$ son síncronos, podemos notar que $[\![\mathcal{T}]\!](\mathcal{L}) = \pi_2\big([\![\mathcal{T}]\!] \bigcap [\![\mathcal{T}_\mathcal{A}]\!]\big)$. Esto se debe a que $[\![\mathcal{T}]\!] \bigcap [\![\mathcal{T}_\mathcal{A}]\!]$ es una relación racional, por lo que su proyección sobre el segundo componente será un lenguaje regular.

Distribución de puntaje

- **(1 pto.)** Por plantear la intersección de dos relaciones racionales.
- **(1 pto.)** Por definir $\mathcal{T}_1 \times \mathcal{T}_2$ junto a una breve explicación.
- **(1 pto.)** Por mencionar al autómata $\mathcal{A}$.
- **(1 pto.)** Por definir $\mathcal{T}_{\mathcal{A}}$.
- **(2 ptos.)** Por demostrar que $[\![\mathcal{T}]\!](\mathcal{L})$ es regular.

Problema 4

Considere el siguiente problema:

Problema:	LONGEST-SUFFIX-NFA
Input:	Un NFA $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ y $w = a_1 \dots a_n \in \Sigma^*$
Output:	$\min \{i \in \{1, \dots, n\} \mid a_i \dots a_n \in \mathcal{L}(\mathcal{A})\}$

Esto es, el problema LONGEST-SUFFIX-NFA consiste en, dado un autómata finito no-determinista $\mathcal{A}$ y una palabra $w = a_1 \dots a_n$, retornar la posición i del sufijo más largo $a_i \dots a_n$ que es aceptado por $\mathcal{A}$. En caso que ese sufijo no exista, el algoritmo debe entregar `NULL`.

Escriba una algoritmo que resuelva LONGEST-SUFFIX-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 1

Para resolver el problema LONGEST-SUFFIX-NFA, se requiere diseñar un algoritmo que encuentre el sufijo más largo de una palabra $w = a_1, ..., a_n$ que sea aceptado por un autómata finito no determinista (NFA) $A = (Q, \Sigma, \delta, I, F)$. La idea es recorrer la palabra desde el final hacia el principio, verificando si algún sufijo es aceptado por el NFA. Para ello, se puede utilizar un algoritmo de simulación de NFA, que permita determinar si un estado es alcanzable desde otro estado, dado un símbolo de entrada. A continuación, se presenta el pseudocódigo del algoritmo:

Algorithm 1: find-longest-suffix-NFA(N, w)

Input: Un NFA $A = (Q, \Sigma, \delta, I, F)$ y una palabra $w = a_1, ..., a_n$
Output: El sufijo más largo de w que es aceptado por A

```
1  S ← F ;                                              // Conjunto de estados finales
2  m ← ∞ ;                                              // Longitud del sufijo más largo
3  for i = n to 1 do
       // Recorre la palabra desde el final
4      S_old ← S ;                                      // Copia de los estados alcanzables
5      S ← ∅ ;                                          // Conjunto de estados alcanzables
       // Notar que este ciclo tiene complejidad O(|Δ|)
6      foreach p ∈ S_old do
           // Recorre los estados alcanzables
7          S ← S ∪ {q|(q, a_i, p) ∈ Δ} ;                // Estados alcanzables con a_i
       // Notar que esta verificación se cumple en O(|Q|)
8      if I ∩ S ≠ ∅ then
           // Si hay estados alcanzables desde el inicial
9          m ← i ;                                      // Actualiza la longitud del sufijo más largo
10 if m = ∞ then
11     return NULL ;                                    // No hay sufijo aceptado
12 else
13     return i ;                                       // Retorna la posición del sufijo más largo
```

Explicación del Algoritmo

1. **Inicialización:** Se inicializa el conjunto de estados S con los estados finales F y la longitud del sufijo más largo m con infinito.
2. **Recorrido de la palabra:** Se recorre la palabra desde el final hacia el principio, verificando si algún sufijo es aceptado por el NFA.
3. **Simulación del NFA:** Para cada símbolo de la palabra, se simula el NFA para determinar los estados alcanzables desde los estados alcanzables anteriores.
4. **Verificación de estados iniciales:** Se verifica si hay algún estado alcanzable desde el estado inicial. Si es así, se actualiza la longitud del sufijo más largo.
5. **Retorno de resultados:** Si la longitud del sufijo más largo es infinito, no hay sufijo aceptado. En caso contrario, se retorna la posición del sufijo más largo.

Complejidad del Algoritmo: La complejidad del algoritmo es $O(|A| \cdot |w|)$ en el peor caso, donde $|A|$ es el tamaño del alfabeto y $|w|$ es la longitud de la palabra. La complejidad se debe a la simulación del NFA, que tiene una complejidad de $O(|\Delta|)$, donde Δ es el conjunto de transiciones del NFA. En el peor caso, se debe recorrer la palabra completa, lo que implica una complejidad de $O(|w|)$.

Justificación de la correctitud

1. **Recorrido completo de la palabra:** El bucle inverso asegura que verificamos todos los posibles sufijos de w.
2. **Simulación del NFA:** La simulación del NFA permite determinar los estados alcanzables desde los estados alcanzables anteriores, dado un símbolo de entrada.
3. **Verificación de estados iniciales:** La verificación de estados iniciales permite determinar si hay algún estado alcanzable desde el estado inicial.
4. **Retorno de resultados:** El algoritmo retorna la longitud del sufijo más largo que es aceptado por el NFA.

Distribución de puntaje

- 1 punto por explicar el algoritmo de simulación del NFA.
- 1 punto por inicializar correctamente el conjunto de estados S y la longitud del sufijo más largo m.
- 1 punto por recorrer los estados alcanzables y actualizar el conjunto de estados S.
- 1 punto por verificar si hay algún estado alcanzable desde el estado inicial y actualizar la longitud del sufijo más largo.
- 1 punto por retornar la longitud del sufijo más largo.
- 1 punto por justificar la correctitud del algoritmo y la complejidad del mismo.

Solución 2

Alternativamente, se puede resolver el problema LONGEST-SUFFIX-NFA recorriendo la palabra desde el principio hacia el final, verificando si algún sufijo es aceptado por el NFA. Para ello, se puede utilizar un algoritmo de simulación de NFA, que permita determinar si un estado es alcanzable desde otro estado, dado un símbolo de entrada. A continuación, se presenta el pseudocódigo del algoritmo:

Algorithm 2: find-longest-suffix-NFA(N, w)

Input: Un NFA $A = (Q, \Sigma, \delta, I, F)$ y una palabra $w = a_1, ..., a_n$
Output: El sufijo más largo de w que es aceptado por A

```
1  S ← {(q,1)|q ∈ I} ∪ {(q,∞)|q ∉ I} ;      // S es una tabla de hash con longitud de sufijo más largo
2  for i = 1 to n do
       // Recorre la palabra desde el principio
3      S_old ← S ;                                          // Copia de los estados alcanzables
4      S ← {(q,i)|q ∈ I} ∪ {(q,∞)|q ∉ I} ;                  // Conjunto de estados alcanzables
5      foreach p ∈ Q do
           // Notar que este ciclo tiene complejidad O(|Δ|)
6          foreach (p, a_i, q) ∈ Δ do
7              S[q] ← min(S[q], i) ;                  // Actualiza la longitud del sufijo más largo
8  m ← min S[p]|p ∈ F ;                                       // Longitud del sufijo más largo
9  if m = ∞ then
10     return NULL ;                                          // No hay sufijo aceptado
11 else
12     return m ;                                  // Retorna la posición del sufijo más largo
```

Explicación del Algoritmo

1. **Inicialización:** Se inicializa la tabla de hash S con los estados iniciales y finales del NFA.
2. **Recorrido de la palabra:** Se recorre la palabra desde el principio hacia el final, verificando si algún sufijo es aceptado por el NFA.
3. **Simulación del NFA:** Para cada símbolo de la palabra, se simula el NFA para determinar los estados alcanzables desde los estados alcanzables anteriores.
4. **Verificación de estados finales:** Se verifica si hay algún estado final alcanzable desde los estados alcanzables.
5. **Retorno de resultados:** Si la longitud del sufijo más largo es infinito, no hay sufijo aceptado. En caso contrario, se retorna la longitud del sufijo más largo.

Complejidad del Algoritmo: La complejidad del algoritmo es $O(|A| \cdot |w|)$ en el peor caso, donde $|A|$ es el tamaño del alfabeto y $|w|$ es la longitud de la palabra. La complejidad se debe a la simulación del NFA, que tiene una complejidad de $O(|\Delta|)$, donde Δ es el conjunto de transiciones del NFA. En el peor caso, se debe recorrer la palabra completa, lo que implica una complejidad de $O(|w|)$.

Justificación de la correctitud

1. **Recorrido completo de la palabra:** El bucle asegura que verificamos todos los posibles sufijos de w.
2. **Simulación del NFA:** La simulación del NFA permite determinar los estados alcanzables desde los estados alcanzables anteriores, dado un símbolo de entrada.
3. **Verificación de estados finales:** La verificación de estados finales permite determinar si hay algún estado final alcanzable desde los estados alcanzables.
4. **Retorno de resultados:** El algoritmo retorna la longitud del sufijo más largo que es aceptado por el NFA.

Distribución de puntaje

- 1 punto por explicar el algoritmo de simulación del NFA.
- 1 punto por inicializar correctamente la tabla de hash S.
- 1 punto por recorrer los estados alcanzables y actualizar la tabla de hash S.
- 1 punto por verificar si hay algún estado final alcanzable y actualizar la longitud del sufijo más largo.
- 1 punto por retornar la longitud del sufijo más largo.
- 1 punto por justificar la correctitud del algoritmo y la complejidad del mismo.