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

IIC2223 — Teoría de Autómatas y Lenguajes Formales — 2' 2018

PAUTA TAREA 3

Pregunta 1

Pregunta 1.1

Para la solución consideramos una clase de equivalencia cualquiera $[w]_{\equiv_L}$. Definimos un autómata $\mathcal{A}^{[w]}_{\equiv_L} = (Q_\equiv, \Sigma, \delta_\equiv, q_\equiv, F_\equiv)$ donde

- $Q_\equiv = \{[u]_{\equiv_L} \mid u \in \Sigma^*\}$
- $q_\equiv = [\epsilon]_{\equiv_L}$
- $F_\equiv = \{[w]_{\equiv_L}\}$
- $\delta_\equiv([u]_{\equiv_L}, a) = [u \cdot a]_{\equiv_L}$

Demostrando que $[w]_{\equiv_L} = \mathcal{L}(\mathcal{A}^{[w]}_{\equiv_L})$ se concluye que cada clase de equivalencia de $\equiv_L$ es un lenguaje regular.

Dado lo anterior, el puntaje asignado es el siguiente:

- **(4 puntos)** Por definir correctamente $\mathcal{A}^{[w]}_{\equiv_L}$ y demostrar que $[w]_{\equiv_L} = \mathcal{L}(\mathcal{A}^{[w]}_{\equiv_L})$.
- **(3 puntos)** Por no demostrar que $[w]_{\equiv_L} = \mathcal{L}(\mathcal{A}^{[w]}_{\equiv_L})$ o tener errores menores en la solución completa.
- **(0 puntos)** En otro caso.

Pregunta 1.2

Dado que L es un lenguaje regular, entonces existe un autómata $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ tal que $\mathcal{L}(\mathcal{A}) = L$. A partir de este autómata, podemos definir un 2DFA $\mathcal{A}' = (Q', \Sigma, \vdash, \dashv, q'_0, q'_f)$ tal que:

- $Q' = Q \cup (Q \times \{r\}) \cup \{q'_f\}$ con $q'_f \notin Q \cup (Q \times \{r\})$
- $q'_0 = q_0$
- Para todo estado $p \in Q$ y para toda letra $a \in \Sigma$:
  - $\delta'(p, a) = (\delta(p, a), \rightarrow)$.
  - $\delta'(p, \dashv) = (q'_f, \rightarrow)$ si $p \in F$.
  - $\delta'(p, \dashv) = ((p, r), \leftarrow)$ si $p \notin F$.

- $\delta'((p,r),a) = ((p,r),\leftarrow)$.
- $\delta'((p,r),\vdash) = (p,\rightarrow)$.

La idea es que, al ejecutar una palabra $w \in \Sigma^*$, $\mathcal{A}'$ simula la ejecución de $\mathcal{A}$ sobre w, y si al llegar al final de w se llega a un estado $p \in F$, se acepta, pero si $p \notin F$, se retrocede hasta el inicio de la palabra y a partir de p se vuelve a ejecutar w. En $\mathcal{A}'$ los estados (q,r) se encargan de retroceder en la palabra y devolver la ejecución a q cuando se llega al inicio de la palabra, mientras que q'_f es el nuevo estado (que no estaba en Q) que acepta si $w^n \in L$ para algún $n \in \mathbb{N}$.

Es importante mencionar que si $w \notin L^{\exists n}$ entonces $\mathcal{A}'$ nunca termina su ejecución y por lo tanto no aceptará w. Tomando esto en cuenta se debe demostrar que $\mathcal{L}(\mathcal{A}') = L^{\exists n}$ usando la noción de ejecución de un 2DFA.

Dado lo anterior, el puntaje asignado es el siguiente:

- **(4 puntos)** Por definir correctamente $\mathcal{A}'$ y demostrar que $\mathcal{L}(\mathcal{A}') = L^{\exists n}$.
- **(3 puntos)** Por no demostrar que $\mathcal{L}(\mathcal{A}') = L^{\exists n}$ o tener errores menores en la solución completa.
- **(0 puntos)** En otro caso.

Pregunta 2

Pregunta 2.1

Para la solución debíamos definir el orden entre los spans. Una posible definición es: sean dos spans $[i_1, j_1\rangle$ y $[i_2, j_2\rangle$ se define la relación $\preccurlyeq$ como:

$$[i_1, j_1\rangle \preccurlyeq [i_2, j_2\rangle \iff j_1 < i_2 \vee (j_1 = j_2 \wedge i_1 \leq i_2)$$

Luego se debía entregar una definición recursiva de todos los spans similar a la que fue vista en clases, para luego definir la semańtica.

- **(4 puntos)** Por definir correctamente la relación, el conjunto de todos los spans y la semántica de acuerdo a la regla definida.
- **(3 puntos)** Por olvidar detalles en la definición (por ejemplo, no mostrar que es un orden total).
- **(0 puntos)** En otro caso

Pregunta 2.2

Se debía entregar un algoritmo con eficiencia similar al que fue visto en clases. Por ejemplo, sean $\mathcal{A} = (Q, \Sigma, \Delta, q_o, F)$ y $w = a_1, \ldots, a_n$:

Algorithm 1 Algoritmo para evaluar *Leftmost-end/Longest*

```
1:  procedure LEFTMOST-END/LONGEST(A, w)
2:      P ← ∅
3:      for i ← 1 to n do
4:          P ← P ∪ (q_0, i)                          ▷ Agregar el estado inicial en cada iteración
5:          P ← updateSet(Δ, a_i, P)                  ▷ Misma función updateSet vista en clases
6:          if { p | (p, k) ∈ P } ∩ F ≠ ∅ then         ▷ Si un run llega al final se prepara para hacer output
7:              m ← min({ k | (p, k) ∈ P ∧ p ∈ F })   ▷ Buscamos el match que empezó antes (Longest)
8:              output [m, i + 1⟩
9:              P ← ∅                                 ▷ Limpiar el conjunto al encontrar un match
10:         end if
11:     end for
12: end procedure
```

- **(4 puntos)** Por definir correctamente el algoritmo y explicar la solución.
- **(3 puntos)** Por errores menores en el algoritmo.
- **(0 puntos)** En otro caso