Tarea #1 - Resolución [1b actualizada]

1. **[25 = 5 + 20 pt]** Dados un alfabeto Σ y un lenguaje $L \subseteq \Sigma^*$, diremos que L es *exigente* si satisface lo siguiente: para toda palabra $w \in L$, ninguna de sus subpalabras propias puede estar en L. Es decir, si tomo cualquier trozo de la palabra (que no sea la palabra entera), el resultado es algo que no está en L. Por ejemplo, el lenguaje $\{aa, bbb, baba\}$ es exigente, pues las posibles subpalabras de aa (o sea a), las de bbb (o sea, b y bb) y las de $baba$ (b, a, ba, ab, bab y aba) están todas fuera de él.

   (a) Dé un ejemplo de algún lenguaje infinito que sea exigente. Justifique.

   (b) De las siguientes operaciones de conjuntos, algunas preservan la propiedad de ser exigente, mientras que otras no. Para cada una determine acaso lo hace; si cree que preserva la propiedad, demuéstrelo, y si cree que no, provea un contraejemplo. Las operaciones a considerar son unión, intersección, concatenación, transposición y estrella de Kleene.

   Nota: "Preservar la propiedad" significa que si la operación se aplica a lenguajes que son exigentes, el lenguaje que resulte también lo es.

   R:

   (a) Sirven muchos. Por ejemplo, ab^*a.

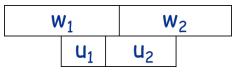
   (b) Acá, para ahorrar espacio, voy a usar la notación $u \prec w$ para expresar la relación "u es subpalabra propia de w".

   Las que *no* preservan la propiedad son:

   - Unión. Contraejemplo: $\{a\}$ y $\{ba\}$ son exigentes (de hecho *cualquier* lenguaje formado por una sola palabra lo es), pero $\{a\} \cup \{ba\} = \{a, ba\}$ no lo es, pues $a \prec ba$.
   - Estrella de Kleene. Contraejemplo: $\{a\}$. Si le aplicamos la estrella nos queda a^*, que contiene tanto a como aa, pero $a \prec aa$.

   Las que sí la preservan son:

   - Intersección. Supongamos que $w \in L_1 \cap L_2$ (lo que implicaría en particular, que $w \in L_1$), y que $u \prec w$. Si $u \in L_1 \cap L_2$, también tendría que estar en L_1, pero eso contradeciría que L_1 haya sido exigente.
   - Transposición. Sea L exigente, y supongamos que L^R no es exigente. En tal caso existirían $u \prec w$, ambas en L^R. Pero al darlas vuelta tendríamos que tanto u^R como w^R estarían en L, y como claramente $u^R \prec w^R$, se contradice que L sea exigente.
   - Concatenación. Sean L_1 y L_2 exigentes, y supongamos que L_1L_2 no lo es. En tal caso existirían $w, u \in L_1L_2$ tales que $u \prec w$. Como están en L_1L_2, cada una de ellas está formada como concatenación de un par de palabras, digamos, $w = w_1w_2$, $u = u_1u_2$, con $u_1, w_1 \in L_1$ y $u_2, w_2 \in L_2$. La situación es como sigue:

     Consideremos lo que ocurre, dependiendo de cómo queden los cortes en las dos palabras. Si el corte de u viene antes que el de w (como en el dibujo), entonces $u_1 \prec w_1$, lo que contradeciría que L_1 era exigente. Si viene después, entonces $u_2 \prec w_2$, contradiciendo la exigencia de L_2. Si coinciden en el mismo punto, contradice a ambos. **UPDATE: Nop, concatenación NO necesariamente preserva exigencia.** Hay un caso particular, que varios descubrieron al hacer la tarea, y en el cual el argumento anterior falla. Es cuando uno de los lenguajes contiene ε, y el otro contiene alguna palabra que es subpalabra de alguna del primero. Así que si los lenguajes no

incluyen ε, la propiedad se preserva, pero si la incluyen, entonces no necesariamente!! [Como en realidad es bieeen sutil la cosa, corregí como buenas las dos respuestas, variando puntaje según calidad del argumento.]

UPDATE DEL UPDATE: Pero por otro lado, ¿puede ε estar en uno de esos lenguajes, si es que son exigentes? En realidad todo depende de si acaso uno considera a ε como subpalabra legítima. Si lo es, lo es de cualquier cosa, lo que hace medio inútil que lo sea... pero cosas así ocurren, como con el conjunto $\emptyset$, que es subconjunto de cualquier conjunto.

- Si *no* consideramos ε subpalabra de otras, entonces podría ser parte de lenguajes exigentes, y la concatenación no preservaría exigencia.
- Si *sí* lo consideramos, entonces ε no puede estar en un lenguaje exigente (y el argumento original se aplica, y se preserva la exigencia). La única forma de que ε esté en un lenguaje exigente sería que el lenguaje consista sólo en eso: $\{\varepsilon\}$. Pero en ese caso al concatenarlo con el otro se obtiene el otro, así que la propiedad también se preserva.

¿Cuál es la correcta entonces? No existe un Diccionario Supremo de Definiciones, así que miré el ppt (nuestra referencia suprema en el curso)... y ahí al definir subpalabras no excluí ε. Ergo, es lo segundo, no lo primero. Especiales felicitaciones a Paula Vergara, que en su tarea hizo exactamente este mismo análisis de la disyuntiva, y su dependencia respecto a la definición.

2. **[15 = 3 × 5 pt]** Para cada uno de los siguientes lenguajes, dé un ejemplo de una palabra que esté dentro del lenguaje, y un ejemplo de una palabra que no lo esté. Si cree que alguna de esas no existe (porque el lenguaje es vacío, o porque es completo), justifique eso.

   (a) $\{w \in \{a,b\}^* : w = a^n b^m, \text{ con } n < m < 2n\}$

   (b) $\{w \in \{a,b\}^* : |w|_{ab} + |w|_{ba} = |w|_{aa} + |w|_{bb}\}$

   (c) El lenguaje de la expresión regular $a^*[(b+\emptyset)(\varepsilon+a)]^*b^*$.

   R:

   (a) La palabra más corta en el lenguaje es con $n = 2$, $m = 3$: $aabbb$. Una que no está: ε.

   (b) ε pertenece (o abb, si quieren una no nula), aa no.

   (c) La expresión es equivalente a $a^*(b+ba)^*b^*$. La palabra a pertenece, la palabra baa no (pues una vez que aparece una b, no pueden aparecer dos a consecutivas).

3. **[15 pt]** Escriba una expresión regular para el lenguaje formado por todas las palabras w con alfabeto $\{a,b,c\}$ tales que $|w|_{ac} = |w|_{ba} = |w|_{cb} = |w|_{ca} = 0$.

   R:

   ¿Cómo son las palabras acá? La condición $|w|_{cb} = |w|_{ca} = 0$ nos indica que una vez que aparece una c, de ahí en adelante sólo pueden aparecer c. Por otro lado, el $|w|_{ba} = 0$ nos dice que si aparece una b, después sólo pueden aparecer otras b, o tal vez alguna c (y, por lo que dijimos primero, de ahí sólo vendrían c). Finalmente, el $|w|_{ac}$ nos dice que después de a, sólo pueden venir otras a, o posiblemente b (y si hay alguna c más adelante, *tiene* que haber al menos una b entremedio). En resumen: si la palabra parte por a, después pueden venir más a, y luego posiblemente algunas b. Si queremos permitir además algunas c al final, tenemos que garantizar la b entremedio. Podemos lograr ambas cosas con $a^*(b^* + bb^*c^*)$. Si no hay a's al comienzo, entonces son sólo b's seguidas por c's, lo que está casi cubierto en lo anterior, salvo por el caso en que no hay ninguna b. De modo que si agregamos c^*, estamos listos:

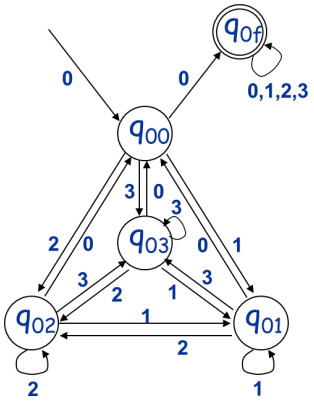
$$a^*(b^* + bb^*c^*) + c^*$$

4. **[25 pt]** Consideremos el alfabeto $\Sigma = \{0, 1, 2, 3\}$, y el lenguaje formado por todas las palabras $w \in \Sigma^+$ que verifican lo siguiente: si el primer símbolo de la palabra es el dígito d, entonces en algún punto de la palabra debe haber dos dígitos contiguos que sumen d. Por ejemplo, 001, 123102 y 31120 cumplen la condición, pero 011, 12312 y 311020 no lo hacen. Construya un autómata finito determinista que reconozca este lenguaje.

   R:

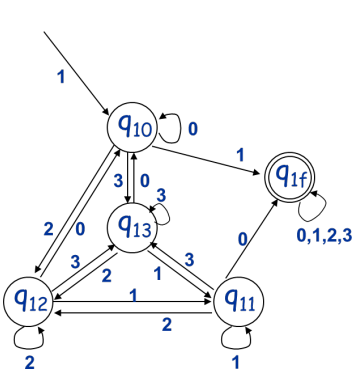
   Básicamente necesitamos recordar con qué dígito partimos, y cual es el más reciente; de esa forma, con el dígito nuevo que llegue podemos chequear si la suma coincide con el inicial (y pasar a un estado de aceptación). Podríamos poner las 16 combinaciones, y empezar a conectarlas. Para simplificar un poco, podemos separar casos, según el dígito inicial (pues son básicamente casos separados). Si partimos con un 0, debe haber un 00. Si partimos con un 1, debe haber un 01 o bien un 10. Si partimos con un 2, debe haber un 02, un 20, o un 11. Y si partimos con 3, tendrá que haber un 03, un 30, un 12 o un 21.

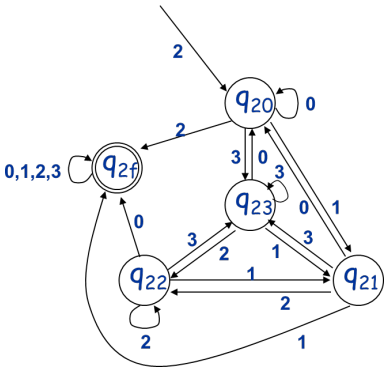
   En cada caso debemos llevar registro del dígito más reciente, y capturar los dígitos que darían la suma. Si partiésemos con 0, necesitamos lo siguiente:

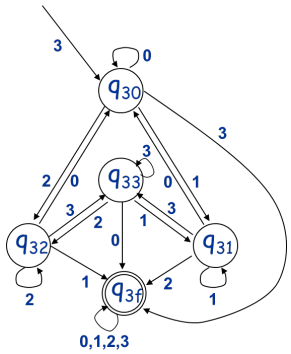


   Aquí los nombres de los estados están puesto de modo que q_{ij} es el caso en que el primer dígito que se leyó fue i, y el más reciente es j. La única transición que necesitamos capturar es el 0 leído en q_{00} (pues es el caso en que los dos dígitos más recientes suman 0, que fue el inicial).

   Los otros casos son similares. Para el 1, 2 y 3, tenemos

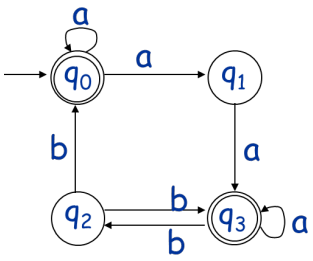






   Para tener el AFD completo, sólo necesitamos reunirlos todos y agregar un estado de inicio, desde el cual saldrán esas cuatro flechas con 0, 1, 2 y 3 que nos llevan a cada caso. (Naturalmente, todos los estados de aceptación podrían ser uno sólo.)

5. **[20 = 15 + 5 pt]** A partir del siguiente AFND+ε,

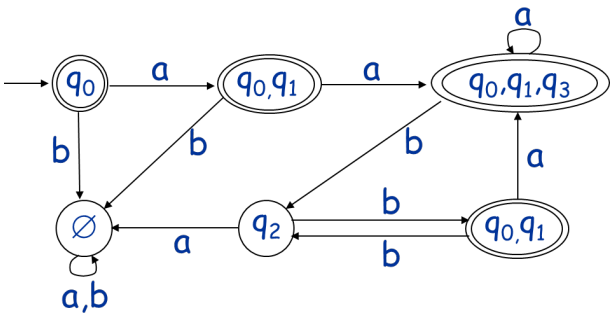


(a) Obtenga un AFD, usando el algoritmo que vimos en clases. Identifique claramente a qué subconjunto de estados del AFND+ε corresponde cada estado del AFD.

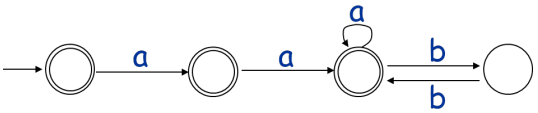
(b) Describa el lenguaje en e.q.s.a.e.[1]

R:

(a) Queda



(b) No es difícil ver que los dos estados de la derecha son equivalentes: ambos son de aceptación, y ambos llevan al mismo destino tanto si se lee una a como una b. Los podemos fundir en uno sólo entonces, y si además quitamos el basurero, básicamente el autómata es



que correspondería a la expresión regular $\varepsilon + a + aa(a + bb)^*$. Por lo tanto el lenguaje está formado por ε, la palabra a, y las palabras que comiencen con dos a y en las cuales cualquier grupo de b's tiene que ser de largo par.

[1] Español que su abuelita entendería ©