Computación Natural: Ejercicios

Jorge Juan González

Mayo 2022

Índice

1. **Ejercicio 1** **2**

2. **Ejercicio 2** **2**

3. **Ejercicio 3** **3**

4. **Ejercicio 4** **3**

5. **Ejercicio 5** **5**

6. **Ejercicio 6** **5**

7. **Ejercicio 7** **6**

1. Ejercicio 1

Un posible algoritmo de tubos basado en la técnica de Lipton que resuelve la instancia del SAT propuesta sería el que se muestra en el Algoritmo 1.

1 $input(N0)$
2 $Na1 = S(N0, 1, 1)$
3 $Na1' = S^{-}(N0, 1, 1)$
4 $Na2 = S(Na1', 2, 0)$
5 $Na = merge(Na1, Na2)$
6 $Nb1 = S(Na, 1, 0)$
7 $Nb1' = S^{-}(Na, 1, 0)$
8 $Nb2 = S(Nb1', 3, 0)$
9 $Nb = merge(Nb1, Nb2)$
10 $Nc1 = S(Nb, 2, 0)$
11 $Nc1' = S^{-}(Nb, 2, 0)$
12 $Nc2 = S(Nc1', 3, 1)$
13 $Nc = merge(Nc1, Nc2)$
14 $detect(Nc)$

Algorithm 1: Algoritmo de tubos

Las variables que aparecen en el algoritmo propuesto han sido nombradas de acuerdo con el término de la instancia del SAT al que representan:

$$(a_1 \vee a_2) \wedge (b_1 \vee b_2) \wedge (c_1 \vee c_2)$$

2. Ejercicio 2

Un posible sistema de stickers ρ que genera el lenguaje propuesto mediante computaciones arbitrarias ($L_n(\rho)$) sería el que se muestra en la Figura 1.

Figura 1: Sistema de stickers

$$\rho = (V, \gamma, A, D)$$

$$V = \{a, b\}\ \gamma = \{(a, a), (b, b)\}\ A = \begin{bmatrix} aa \\ aa \end{bmatrix}$$

$$D = \{\left(\begin{pmatrix} a \\ a \end{pmatrix}, \begin{pmatrix} a \\ a \end{pmatrix}\right), \left(\begin{pmatrix} b \\ b \end{pmatrix}, \begin{pmatrix} b \\ b \end{pmatrix}\right)\}$$

El lenguaje de moléculas no restringido de ρ ($LM_n(\rho)$) sería:

$$LM_n(\rho) = \{\begin{bmatrix} w \\ w \end{bmatrix} \begin{bmatrix} w \\ w \end{bmatrix}^r : w \in \{a, b\}^*\}$$

El lenguaje propuesto $L_n(\rho)$ es, en función de la forma de sus dominoes:

- **No unilateral:** Existe al menos un dominó que hace crecer la molécula por ambos lados.
- **No regular:** Existe al menos un dominó que hace crecer la molécula por la izquierda.
- **No simple:** No todos los dominoes hacen crecer la misma hebra por el mismo lado.

Dado que con D es imposible generar moléculas no completas, el lenguaje de sus computaciones primitivas $L_p(\rho)$ será equivalente al generado usando computaciones arbitrarias $L_n(\rho)$, por lo tanto:

$$L_p(\rho) = L_n(\rho)$$

De nuevo, para el D elegido, ninguna computación o serie de operaciones de sticking resulta en un retardo mayor que 0, por lo tanto:

$$\forall d \leq 0, L_d(\rho) = L_n(\rho)$$

3. Ejercicio 3

De acuerdo con el enunciado del ejercicio, el lenguaje de computaciones arbitrarias generado por ρ sería:

$$L_n(\rho) = \{a^{n+1}b^nc^n : n \geq 0\}$$

Un posible sistema de stickers ρ que genera el lenguaje propuesto mediante computaciones arbitrarias ($L_n(\rho)$) sería el que se muestra en la Figura 2.

Figura 2: Sistema de stickers

$$\rho = (V, \gamma, A, D)$$

$$V = \{a, b, c\}\ \gamma = \{(a, a), (b, b), (c, c)\}\ A = \begin{bmatrix} a \\ a \end{bmatrix}$$

$$D = \{\left(\begin{pmatrix} a \\ \lambda \end{pmatrix}, \begin{pmatrix} b \\ \lambda \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} c \\ \lambda \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ b \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ a \end{pmatrix}, \begin{pmatrix} \lambda \\ c \end{pmatrix}\right)\}$$

El lenguaje de moléculas no restringido de ρ ($LM_n(\rho)$) sería:

$$LM_n(\rho) = \{\begin{bmatrix} a \\ a \end{bmatrix}^{n+1} \begin{bmatrix} b \\ b \end{bmatrix}^n \begin{bmatrix} c \\ c \end{bmatrix}^n : w \in \{a, b\}^*\}$$

Podríamos caracterizar, de acuerdo a la forma de sus dominoes, el lenguaje propuesto $L_n(\rho)$, generado por ρ mediante computaciones arbitrarias, de la misma manera que en el ejercicio anterior:

- **No unilateral:** Existe al menos un dominó que hace crecer la molécula por ambos lados.
- **No regular:** Existe al menos un dominó que hace crecer la molécula por la izquierda.
- **No simple:** No todos los dominoes hacen crecer la misma hebra por el mismo lado.

En este caso, $L_n(\rho)$ sí que incluye moléculas no completas, de modo que el lenguaje de sus computaciones primitivas $L_p(\rho)$ será un subconjunto del generado usando computaciones arbitrarias $L_n(\rho)$, por lo tanto:

$$L_p(\rho) \subseteq L_n(\rho)$$

Para el sistema de stickers ρ propuesto, cualquier operación de sticking aplicada sobre el axioma tendrá un retardo mínimo de 1. Sin embargo, es posible llegar a cualquier molécula del lenguaje con un retardo acotado a 1. De este modo podemos concluir que:

$$\forall 1 \leq d \leq n, L_d(\rho) = L_n(\rho)$$

4. Ejercicio 4

Asumiendo que sólo se procesarán moléculas completas y que la relación de complementariedad es la identidad, un posible AFWK para ese lenguaje sería el que se muestra en la Figura 3, cuyas transiciones y diagrama de transiciones se muestran en las Figuras 4 y 5 respectivamente.

Este AFWK consumiría el primer w de la hebra superior hasta que puede consumir una c. A partir de ese momento consume los mismos caracteres en ambas hebras, lo cual sólo será posible cuando la segunda w de la hebra superior sea igual a la primera w de la hebra inferior. Después, una vez consumida la c restante, empezará a consumir las restantes a y b de la hebra inferior, las cuales compondrán la segunda w leída de la hebra superior debido a que la relación de complementariedad es la identidad.

Si existieran más símbolos a la derecha de la c que a la izquierda, el autómata llegaría a un punto en el que quedasen caracteres en la hebra superior que nunca pueden ser consumidos dado

que después de leer la c de la hebra inferior sólo consumimos caracteres de la hebra inferior.

Si existieran más símbolos a la izquierda de la c que a la derecha, el autómata llegaría a un punto en el que sería imposible alcanzar la c de la hebra inferior dado que, al consumirse los mismos símbolos de la hebra superior que de la inferior, los símbolos de la hebra superior se acabarían antes de alcanzar la c de la hebra inferior.

Hemos contemplado al posibilidad de que w sea una cadena vacía y, por tanto, si el primer símbolo que se puede leer es una c en ambas hebras, entonces estamos en ese escenario y la molécula debería terminar en ese punto.

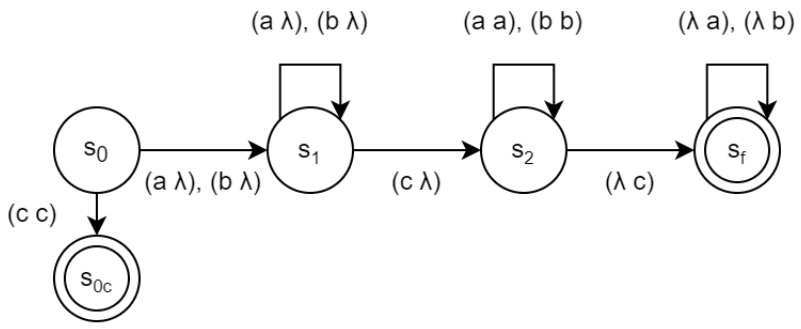
Figura 3: Descripción del AFWK

$$M = (V, \gamma, K, s_0, F, \delta)$$
$$V = \{a, b, c\}\ \gamma = \{(a, a), (b, b)\}$$
$$K = \{s_0, s_{0c}, s_1, s_2, s_f\}\ F = \{s_{0c}, s_f\}$$

Figura 4: Transiciones del AFWK

$$\delta\left(s_0, \begin{pmatrix} a \\ \lambda \end{pmatrix}\right) = \{s_1\}\ \delta\left(s_0, \begin{pmatrix} b \\ \lambda \end{pmatrix}\right) = \{s_1\}\ \delta\left(s_0, \begin{pmatrix} c \\ c \end{pmatrix}\right) = \{s_{0c}\}$$
$$\delta\left(s_1, \begin{pmatrix} a \\ \lambda \end{pmatrix}\right) = \{s_1\}\ \delta\left(s_1, \begin{pmatrix} b \\ \lambda \end{pmatrix}\right) = \{s_1\}\ \delta\left(s_1, \begin{pmatrix} c \\ \lambda \end{pmatrix}\right) = \{s_2\}$$
$$\delta\left(s_2, \begin{pmatrix} a \\ a \end{pmatrix}\right) = \{s_2\}\ \delta\left(s_2, \begin{pmatrix} b \\ b \end{pmatrix}\right) = \{s_2\}\ \delta\left(s_2, \begin{pmatrix} \lambda \\ c \end{pmatrix}\right) = \{s_f\}$$
$$\delta\left(s_f, \begin{pmatrix} \lambda \\ a \end{pmatrix}\right) = \{s_f\}\ \delta\left(s_f, \begin{pmatrix} \lambda \\ b \end{pmatrix}\right) = \{s_f\}$$

Figura 5: Diagrama de transiciones del AFWK.



El autómata tiene más de un estado final y no todos sus estados son finales por lo que no es stateless ni all-final. Tampoco sería simple porque existen transiciones que consumen símbolos de ambas hebras. Sin embargo, el autómata propuesto sí que sería 1-limitado dado que en cada transición se consumiría como mucho un símbolo de cada hebra. Por tanto, la familia de lenguajes se denotaría como $1WK(\alpha)$.

5. Ejercicio 5

La mejor variante de AFWK a utilizar para este problema sería el AFWK reverso. Un puntero empezaría en el inicio de la hebra superior y otro puntero al final de la hebra inferior, avanzando ambos en direcciones opuestas. Él autómata consumiría los mismos símbolos por ambos punteros y pararía en el momento en el que los punteros se encuentren. Las Figuras 6 y 7 describen el AFWK reverso propuesto y sus transiciones respectivamente. La Figura 8 representa las transiciones del autómata.

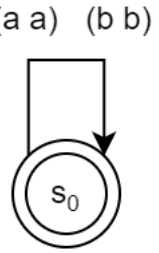
Figura 6: Descripción del AFWK reverso

$$M = (V, \gamma, K, s_0, F, \delta)$$
$$V = \{a, b\}\ \gamma = \{(a, a), (b, b)\}$$
$$K = \{s_0\}\ F = \{s_0\}$$

Figura 7: Transiciones del AFWK reverso

$$\delta\left(\ s_0, \begin{pmatrix} a \\ a \end{pmatrix}\ \right) = \{s_0\}\ \delta\left(\ s_0, \begin{pmatrix} b \\ b \end{pmatrix}\ \right) = \{s_0\}$$

Figura 8: Diagrama de transiciones del AFWK reverso.



En este caso, optar por un AFWK reverso simplifica mucho el autómata hasta el punto de estar compuesto de un único estado, inicial y final, haciendo que sea de tipo stateless.

El autómata consta de transiciones que consumen un símbolo por hebra, por lo que también será de tipo 1-limitado.

De acuerdo con lo expuesto, el autómata sería stateless + 1-limitado, por lo que la familia de lenguajes se denotaría como $N1WK(\alpha)$.

6. Ejercicio 6

La cadena $ababbabbaaabba \in D^*(L)$ porque podemos llegar a ella realizando operaciones de duplicación a partir de una cadena de L. Las operaciones de duplicación que podríamos hacer para llegar a ella serían las siguientes:

$$abba \rightarrow \underline{ab}|\underline{abba} \rightarrow ab\underline{abba}|\underline{abba} \rightarrow ababba\underline{a}|\underline{abba} \rightarrow ab\underline{abba}|\underline{bbaaabba}$$

7. Ejercicio 7

La regla $caba\#a\$cab\#a$ nos indica que si nos encontramos ante una molécula cuya hebra superior empieza por $cabaa$ y su hebra inferior empieza por $caba$, deberemos cortar las hebras donde nos indican los símbolos # e intercambiar los dos segmentos derechos resultantes, quedándonos sólo con la hebra superior al tratarse de operaciones 1-splicing.

Si realizamos la siguiente operación de splicing utilizando hebras presentes en L obtenemos la hebra $cabaaab$:

$$(\underline{caba}b, cab\underline{aab}) \vdash_r \underline{cabaaab}$$
$$cabaaab \in \sigma_1^1(L)$$

Después podemos realizar la siguiente operación de splicing utilizando la mayor hebra obtenida hasta ahora para obtener la hebra $cabaaaab$:

$$(\underline{caba}ab, cab\underline{aaab}) \vdash_r \underline{cabaaaab}$$
$$cabaaaab \in \sigma_1^2(L)$$

Podemos repetir esta última operación infinitamente obteniendo hebras de cada vez mayor longitud:

$$(\underline{caba}ab, cab\underline{aaaab}) \vdash_r \underline{cabaaaaab} \in \sigma_1^3(L)$$
$$(\underline{caba}ab, cab\underline{aaaaab}) \vdash_r \underline{cabaaaaaab} \in \sigma_1^4(L)$$

El resto de operaciones de splicing que podemos hacer nos darán hebras que pertenecen a L o a $\sigma_1^{n-1}(L)$.

De este modo podemos concluir que:

$$\sigma_1^*(L) = \{caba^nb, n \geq 0\}$$

Véase que n puede ser 0. Esto se debe a que la hebra $cabb \in L$ y sabemos que $\sigma_1^0(L) = L$, por lo que $cabb \in \sigma_1^*(L)$.