Computación Natural: Trabajo

Jorge Juan González

13 de mayo de 2022

Índice

1. Introducción

En este trabajo diseñaremos un módulo de análisis de moléculas completas (m) para sistemas de stickers regulares (ρ). En concreto, implementaremos una función en Python que, dado un sistema de stickers regulares y una molécula completa nos devolverá si ese sistema puede generar esa molécula y, de ser así, la computación o serie de operaciones de sticking necesaria para generarla.

2. Implementación

El código implementado comprueba que la molécula introducida es completa, comprueba que todas las letras de la molécula pertenecen al vocabulario del sistema y si la molécula contiene el axioma.

Si se cumplen estas condiciones, pasamos a comprobar si con el set de dominoes D del sistema podemos llegar a obtener la molécula a partir del axioma A del sistema. Para ello, realizamos una búsqueda en profundidad en la que revertimos un dominó en cada nodo del árbol de búsqueda.

Entendemos por reversión la operación inversa a la operación de sticking, es decir, dada una molécula original y un dominó se obtiene la molécula sobre la cual aplicar ese dominó resultaría en la molécula original.

Si esa operación es posible para la molécula y el dominó elegido, guardamos el dominó elegido en una pila y volvemos a elegir el primer dominó ($D_\rho[0]$). Después, repetimos la comprobación usando la molécula resultante y ese primer dominó.

Si la operación no es posible, elegimos el siguiente dominó y volvemos a intentar. Si no hay un dominó siguiente o si hemos llegado a un axioma incorrecto, retrocedemos al paso anterior a la última reversión exitosa, elegimos el siguiente dominó y volvemos a intentar.

El pseudocódigo de este algoritmo ha sido dividido en las 3 funciones que se presentan a continuación.

Function `CanGenerate`($\rho(V, \gamma, A, D)$, m_0)**:**
- **if** $m_0 \notin WK_\gamma(V)$ **then**
  - **return** $False$
- **end**
- **if** $m_0 \notin V^*$ **then**
  - **return** $False$
- **end**
- **if** $A \not\subseteq m_0$ **then**
  - **return** $False$
- **end**
- $D_{used} \leftarrow [\,]$
- **return** $TryRevert(A, D, m_0, 0, D_{used})$

Algorithm 1: Función principal

```
Function TryRevert(A, D, m_0, i, D_used):
    m, successful ← Revert(m_0, D[i])
    if successful then
        Append(D_used, i)
        i ← 0
        if m = A then
            return True, D_used                                  /* Axioma correcto */
        else
            if length(m) ≤ length(A) then
                return OnFail(A, D, m, i, D_used)                /* Axioma incorrecto */
            else
                return TryRevert(A, D, m, i, D_used)             /* Continuamos revirtiendo */
            end
        end
    else
        return OnFail(D, m_0, i, D_used)
    end
```

Algorithm 2: Función encargada de reducir la molécula hasta el axioma

```
Function OnFail(A, D, m_0, i, D_used):
    i ← i + 1                                                    /* Siguiente dominó */
    if i ≥ length(D) then
        if length(D_used) = 0 then
            return False, D_used      /* No quedan más combinaciones de dominoes */
        else
            i ← Pop(D_used)                                      /* Último dominó válido */
            m ← Apply(m_0, D[i])                                 /* Deshacemos la reversión */
            return OnFail(A, D, m, i, D_used)                    /* Volvemos a cambiar de dominó */
        end
    else
        return TryRevert(A, D, m_0, i, D_used)                   /* Continuamos con el nuevo dominó
         */
    end
```

Algorithm 3: Función encargada de cambiar de dominó o deshacer la última reversión exitosa

El algoritmo elegido permite comprobar si una molécula cualquiera forma parte del lenguaje para cualquier sistema de stickers. Sin embargo, el coste de este algoritmo crece linealmente con la longitud de la molécula y exponencialmente con el número de dominoes del sistema de stickers. De este modo el coste del algoritmo propuesto sería $\mathcal{O}(n^n)$.

Si conociésemos el Autómata Finito de Watson-Crick que comprueba que una molécula completa pertenece al lenguaje generado por un sistema de stickers regular determinado, podríamos utilizar ese AFWK para resolver la tarea con un coste lineal $\mathcal{O}(n)$.

3. Experimentación y resultados

Para comprobar que nuestro algoritmo funciona correctamente evaluaremos la salida obtenida para cada posible entrada usando los sistemas de stickers regulares y las moléculas descritos en las Figuras 1, 2 y 3.

Figura 1: Sistema de stickers regular 1

$$\rho_1 = (V_1, \gamma_1, A_1, D_1)$$

$$V_1 = \{a, b\}\ \gamma_1 = \{(a, a), (b, b)\}\ A_1 = \begin{bmatrix} a \\ a \end{bmatrix}$$

$$D_1 = \{\left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} a \\ a \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} b \\ b \end{pmatrix}\right)\}$$

Figura 2: Sistema de stickers regular 2

$$\rho_2 = (V_2, \gamma_2, A_2, D_2)$$

$$V_2 = \{a, b, c\}\ \gamma_2 = \{(a, a), (b, b)\}\ A_2 = \begin{bmatrix} a \\ a \end{bmatrix}$$

$$D_2 = \{\left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} b \\ \lambda \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} c \\ \lambda \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ b \end{pmatrix}\right), \left(\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}, \begin{pmatrix} \lambda \\ c \end{pmatrix}\right)\}$$

Figura 3: Moléculas completas empleadas

$$\boldsymbol{m_1} = \begin{bmatrix} a & a & a & b & b & b \\ a & a & a & b & b & b \end{bmatrix} \boldsymbol{m_2} = \begin{bmatrix} a & b & b & c & c & c \\ a & b & b & c & c & c \end{bmatrix}$$

$$\boldsymbol{m_3} = \begin{bmatrix} a & b & b & b & b & b \\ a & b & b & b & b & b \end{bmatrix} \boldsymbol{m_4} = \begin{bmatrix} b & a & a & a & a & a \\ b & a & a & a & a & a \end{bmatrix}$$

La Tabla 1 muestra cuáles de las siguientes moléculas completas (m_{1-4}) pueden ser generadas por cada uno de los sistemas de stickers regulares descritos previamente (ρ_{1-2}) de acuerdo con la salida del algoritmo implementado.

Cuadro 1: Resultados obtenidos.

	$\boldsymbol{m_1}$	$\boldsymbol{m_2}$	$\boldsymbol{m_3}$	$\boldsymbol{m_4}$
$\boldsymbol{\rho_1}$	Sí	No	Sí	No
$\boldsymbol{\rho_2}$	No	Sí	Sí	No

Cabe destacar que el algoritmo también ha sido probado con otros 3 sistemas de stickers no regulares y 3 moléculas distintas para validar que el algoritmo funciona con sistemas de stickers no regulares.

4. Conclusiones

Hemos diseñado e implementado un algoritmo capaz de, dado un sistema de stickers y una molécula, comprueba que la molécula es completa y que el sistema puede generarla. Hemos puesto a prueba este algoritmo combinando varios sistemas de stickers regulares y moléculas completas obteniendo los resultados correctos en todos los casos.