NOMBRE: Benjamín Farías Valdés

N.ALUMNO: 17642531

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

IIC2223 — Teoría de Autómatas y Lenguajes Formales — 2' 2021

Tarea 1

Pregunta 1

Dado un lenguaje regular cualquiera L, el lenguaje $L-1$ corresponde al de todas las palabras de L pero **sin su último símbolo.** Con esto en mente, la intuición indica que el autómata que acepta al lenguaje $L-1$ será muy similar al que acepta al lenguaje L, pero **los estados que venían justo antes que los de aceptación serán los de aceptación en este nuevo autómata,** de forma que acepte las palabras que terminan en 1 letra antes que las de L.

Dado el autómata $\mathcal{A}$ tal que $\mathcal{L}(\mathcal{A}) = L$ (que sabemos que existe ya que L es regular), podemos formalizar la definición del autómata $\mathcal{B}$, que será el que acepta al lenguaje $L-1$. Sea $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ el DFA que acepta al lenguaje L, entonces se define $\mathcal{B}$ a continuación:

$$\mathcal{B} = (Q, \Sigma, \delta, q_0, F')$$

- El conjunto de estados Q, estado inicial q_0, alfabeto Σ y función de transición δ **son los mismos que $\mathcal{A}$.**
- El conjunto de estados finales cambia a F', que se define a continuación:

$$F' := \{q \in Q | \exists a \in \Sigma . \delta(q, a) \in F\}$$

- Este nuevo conjunto de estados finales F' corresponde a **todos aquellos estados que tenían una transición hacia un estado final en el autómata $\mathcal{A}$.**

Ahora debemos demostrar que $\mathcal{L}(\mathcal{B}) = L-1$, para lo que **demostraremos la contención hacia ambos lados:**

- $L-1 \subseteq \mathcal{L}(\mathcal{B})$**:** Sea $w = a_1 a_2 ... a_n \in L-1$. Existe entonces una palabra $u = w \cdot a_{n+1} = a_1 a_2 ... a_n a_{n+1} \in L$, debido a la definición del conjunto $L-1$. Para el autómata $\mathcal{A}$, tenemos entonces la siguiente ejecución de aceptación sobre la palabra u:

$$\rho_{\mathcal{A},u} : q_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} ... \xrightarrow{a_n} p_n \xrightarrow{a_{n+1}} p_{n+1} \wedge p_{n+1} \in F$$

  Como la palabra w es igual a u pero **sin su último símbolo,** podemos también ejecutarla en el autómata $\mathcal{A}$, aunque la ejecución **NO será necesariamente de aceptación:**

$$\rho_{\mathcal{A},w} : q_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} ... \xrightarrow{a_n} p_n$$

Ahora bien, sabemos por construcción del autómata $\mathcal{B}$ que cualquier estado que tenga una transición hacia un estado final en $\mathcal{A}$ será final en $\mathcal{B}$:

$$F' := \{q \in Q | \exists a \in \Sigma . \delta(q, a) \in F\}$$

En particular, tenemos que esto se cumple para el estado p_n al ver la primera ejecución $\rho_{\mathcal{A},u}$:

$$q = p_n$$
$$a = a_{n+1}$$
$$\delta(p_n, a_{n+1}) = p_{n+1} \in F$$

Por lo tanto, se desprende que el estado p_n de estas ejecuciones será un **estado de aceptación en el autómata $\mathcal{B}$,** es decir, $p_n \in F'$. Ahora, como sabemos que el conjunto de estados, estado inicial y transiciones son los mismos para ambos autómatas, podemos ejecutar la palabra w en $\mathcal{B}$:

$$\rho_{\mathcal{B},w} : q_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} ... \xrightarrow{a_n} p_n \in F'$$

Esta última ejecución es idéntica a $\rho_{\mathcal{A},w}$, pero como sabemos que $p_n \in F'$, **necesariamente es una ejecución de aceptación para $\mathcal{B}$.**

Dada cualquier palabra $w \in L - 1$, es posible obtener una palabra $u \in L$ (por definición) tal que se cumpla lo demostrado arriba, por lo que concluímos finalmente que **el autómata $\mathcal{B}$ acepta al lenguaje $L - 1$,** es decir, $L - 1 \subseteq \mathcal{L}(\mathcal{B})$.

- $\mathcal{L}(\mathcal{B}) \subseteq L - 1$**:** Sea $v \in \mathcal{L}(\mathcal{B})$. Sabemos que existirá un ejecución de aceptación de $\mathcal{B}$ sobre v de la forma:

$$\rho_{\mathcal{B},v} : q_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} ... \xrightarrow{a_k} p_k \wedge p_k \in F'$$

Tenemos que p_k es un estado de aceptación en $\mathcal{B}$, lo que por construcción de $\mathcal{B}$ significa que satisface:

$$\exists a \in \Sigma . \delta(p_k, a) \in F$$

Esto implica que existe un símbolo $a \in \Sigma$ tal que existe la transición $\delta(p_k, a) = p_{k+1}$, donde $p_{k+1} \in F$, lo que quiere decir que en el autómata $\mathcal{A}$, p_{k+1} **es un estado de aceptación.** Con esta información, formamos la palabra $w = v \cdot a$, y la ejecutamos sobre el autómata $\mathcal{A}$, sabiendo que la ejecución tendrá la misma estructura que la de $\mathcal{B}$ por construcción, pero realizando una lectura más debido a que añadimos al símbolo a:

$$\rho_{\mathcal{A},w} : q_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} ... \xrightarrow{a_k} p_k \xrightarrow{a_{k+1}} p_{k+1} \wedge p_{k+1} \in F$$

Dado que la palabra $w = v \cdot a$ fue aceptada por el autómata $\mathcal{A}$, se desprende que $w \in L$. Además, si elegimos al símbolo a, entonces $v \in L - 1$ **por definición del conjunto,** puesto que satisface:

$$\exists a \in \Sigma . v \cdot a \in L$$

Dada cualquier palabra $v \in \mathcal{L}(\mathcal{B})$, es posible encontrar un símbolo $a \in \Sigma$ tal que se cumpla lo demostrado arriba, por lo que concluímos finalmente que **toda palabra aceptada por el autómata $\mathcal{B}$ pertenece al lenguaje $L - 1$,** es decir, $\mathcal{L}(\mathcal{B}) \subseteq L - 1$.

Finalmente, como se cumple la contención hacia ambos lados, concluímos que $\mathcal{L}(\mathcal{B}) = L - 1$, por lo que el lenguaje $L - 1$ es **regular.**

NOMBRE: Benjamín Farías Valdés

N.ALUMNO: 17642531

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

IIC2223 — Teoría de Autómatas y Lenguajes Formales — 2' 2021

Tarea 1

Pregunta 2

a)

Antes de construir el autómata necesario, es útil notar que lo que se busca es aceptar todas las secuencias de vectores tales que al sumarlos se obtiene el vector nulo (el que tiene 0 en todas sus componentes). Podemos fijarnos primero en la suma de cualquiera de las componentes de los vectores, la que corresponde a una suma de 1s y 0s. De la definición de la suma que aparece en el enunciado, es posible ver que al sumar cualquier componente de estos vectores, el valor acumulado de la suma irá **alternando** entre 0 y 1, y de hecho este valor cambiará cada vez que se sume un 1. Esto ocurre **por definición,** ya que al sumar un 0 con un 1 se obtiene 1, y cualquier 0 que se le sume posteriormente seguirá entregando 1, hasta que se sume otro 1 y entonces quede de nuevo en 0. De esta observación se puede desprender que **la paridad de la cantidad de 1s determina finalmente el resultado de la suma completa:**

1. **Cantidad Par de 1s:** Como la cantidad de 1s es múltiplo de 2, cada número 1 se terminará cancelando con algún otro al momento de sumar, por lo que al final sólo quedará la suma de los 0s que es simplemente 0.
2. **Cantidad Impar de 1s:** Como la cantidad de 1s NO es múltiplo de 2, al realizar la suma y cancelar los pares de 1s, quedará un 1 sólo, el que al sumarlo con los 0s entregará como resultado 1.

Entonces, haciendo uso de esta propiedad, podemos comenzar por diseñar un DFA que revise si la cantidad de 1s dentro de una de las componentes de la secuencia de vectores es **par,** aceptando en dicho caso y rechazando si es impar, puesto que sólo cuando es par la suma de esa componente resultará en 0. A continuación se construye este autómata para la 1era componente, el que denominaremos como $\mathcal{S}_1$:

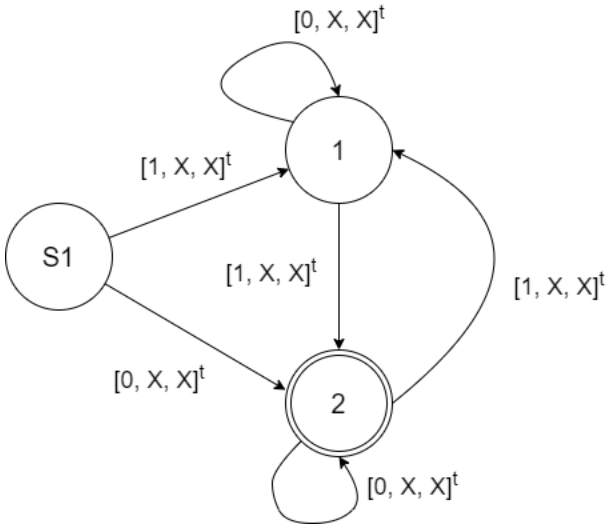

Figura 1: Autómata $\mathcal{S}_1$, que analiza la 1era componente de los vectores

Este DFA posee un estado inicial $S1$ (para evitar aceptar la secuencia vacía) y 2 estados que son los que determinan la paridad de la cantidad de 1s. El estado 1 indica que hasta el momento se ha leído una cantidad impar de 1s, mientras que el estado 2 indica que se ha leído una cantidad par. Al momento de leer un vector y encontrar un 1 en su 1era componente, el autómata $\mathcal{S}_1$ alterna entre los estados 1 y 2, y al terminar de leer la secuencia de vectores aceptará en el estado 2 (que indica que la cantidad de 1s es par) y rechazará en el estado 1. Esto significa que este autómata aceptará si es que la suma de los vectores de la secuencia de entrada es 0 en la 1era componente, según la propiedad encontrada más arriba sobre la paridad de la cantidad de 1s presentes.

De forma **análoga,** podemos construir un autómata $\mathcal{S}_2$ que realiza lo mismo que $\mathcal{S}_1$, pero revisando la 2da componente de cada vector:

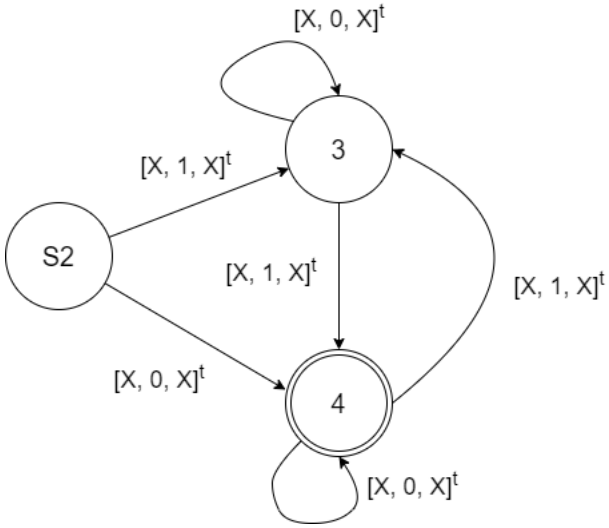

Figura 2: Autómata $\mathcal{S}_2$, que analiza la 2da componente de los vectores

Por último, construimos un tercer autómata para analizar la 3era componente:

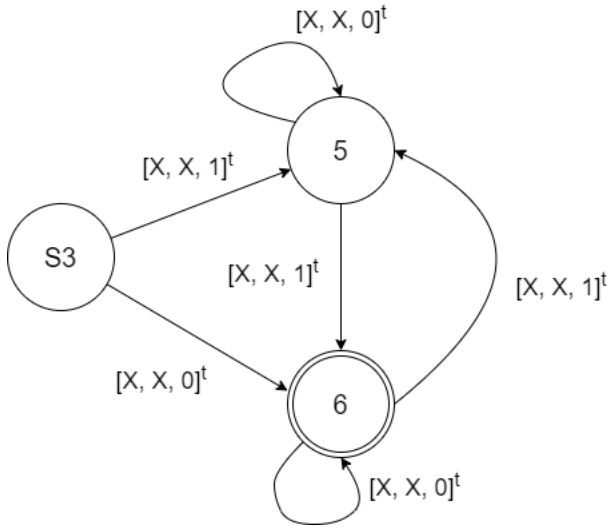

Figura 3: Autómata $\mathcal{S}_3$, que analiza la 3era componente de los vectores

Como necesitamos que **todas las componentes sumen** 0**,** podemos construir un DFA $\mathcal{S}$ que corresponde al **producto entre estos 3,** ejecutándolos en paralelo y aceptando cuando los 3 aceptan la secuencia de entrada:

$$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{S}_3$$

El autómata $\mathcal{S}$ es justamente lo que buscamos, ya que revisa todas las componentes en la secuencia de vectores, y acepta sólo cuando la suma en cada una de ellas es 0, lo que implica que se cumple la siguiente propiedad:

$$\sum_{i=1}^{n} \vec{v_i} = [0, 0, 0]^t$$

Por lo tanto, tenemos que $\mathcal{L}(\mathcal{S}) = L_1$, y como se encontró un DFA que acepta al lenguaje, se tiene que corresponde a un **lenguaje regular** sobre el alfabeto $\{0, 1\}^3$.

b)

El lenguaje L_2 corresponde al lenguaje de todas las secuencias de vectores tal que existe al menos un par de ellos cuyo producto punto es 0, según las definiciones de suma y multiplicación del enunciado. A diferencia del ejercicio anterior, en el que sólo importaba la suma acumulada de todos los vectores, ahora nuestro autómata **debe ser capaz de revisar todos los pares de vectores posibles dentro de la secuencia de entrada, y aceptar si es que alguno de estos pares cumple con la propiedad del producto punto nulo.**

Para construir este autómata, dividiremos el problema en 2 partes:

1. Encontrar un autómata $\mathcal{P}$ que sea capaz de revisar si un par de vectores cumplen con tener su producto punto nulo.

2. Encontrar un autómata $\mathcal{A}$ que sea capaz de revisar todas las combinaciones posibles de 2 vectores dentro de la secuencia de entrada, y que funcione en conjunto con $\mathcal{P}$ para poder resolver el problema.

Primero diseñaremos el autómata $\mathcal{P}$. Comenzamos analizando la operación del producto punto entre 2 vectores v_1 y v_2:

1. Se realiza la **multiplicación componente a componente** entre ambos vectores, resultando en un nuevo vector: $v_t = v_1 \times v_2$.
2. Se **suman** los valores de las 3 componentes de v_t, obteniendo el valor del producto punto: $p = v_t[0] + v_t[1] + v_t[2]$

Si nos fijamos en el paso 2, podemos ver que **aplica la misma propiedad de paridad de la cantidad de 1s** utilizada para el ejercicio $2a$. Esto es, si la cantidad de 1s en las componentes de v_t es par, entonces la suma resultará en 0. Como los valores de las componentes de v_t vienen del paso 1 (multiplicación), es necesario darse cuenta de que la única forma de que alguna componente de v_t sea 1 es que esa misma componente sea 1 tanto en v_1 como en v_2 (dado que según enunciado, 1 sólo se obtiene de multiplicar 1×1). Por lo tanto, para poder determinar si el producto punto entre 2 vectores es 0, necesitamos un autómata que **revise cada componente, verifique si los 2 vectores tienen valor 1 en dicha componente, y luego revise si la cantidad de veces que esto se cumple es par** (ya que representa a los 1s que luego serán sumados en el paso 2 del producto punto).

De forma similar al ejercicio $2a$, definimos el siguiente DFA, denominado $\mathcal{C}_1$, que revisa la 1era componente de los próximos 2 vectores y acepta en caso de que ambos tengan valor 1. En cualquier otro caso, quedará atrapado en el estado R y eventualmente rechazará:

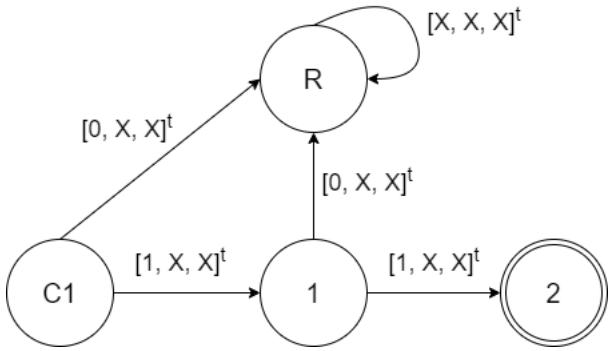

Figura 4: Autómata $\mathcal{C}_1$, que analiza la 1era componente de un par de vectores, revisando si ambos valen 1

Análogamente, construimos el DFA $\mathcal{C}_2$, que hace lo mismo que $\mathcal{C}_1$ pero para la 2da componente de los vectores:

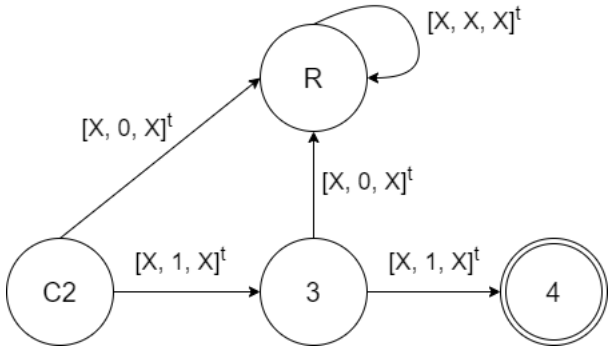

Figura 5: Autómata $\mathcal{C}_2$, que analiza la 2da componente de un par de vectores, revisando si ambos valen 1

Se hace nuevamente lo mismo para la 3era componente:

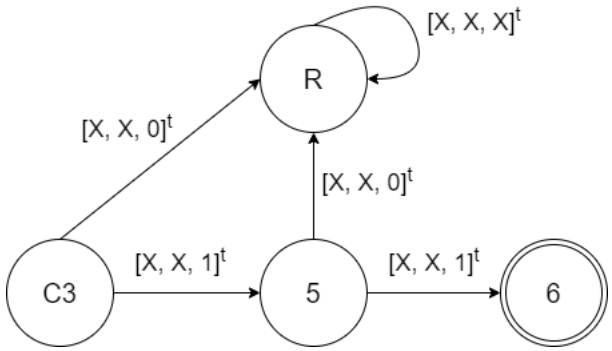

Figura 6: Autómata $\mathcal{C}_3$, que analiza la 3era componente de un par de vectores, revisando si ambos valen 1

Teniendo estos 3 autómatas, hay que notar que para que haya una cantidad par de 1s (y por lo tanto el producto punto sea 0), deben aceptar **2 de los 3 autómatas** (es decir uno de ellos debe rechazar, ya que sino serían una cantidad impar), o bien, **los 3 deben rechazar.** Para cada uno de estos casos, definimos un autómata a continuación:

- $\mathcal{C}_1$ **y** $\mathcal{C}_2$ **aceptan**: Se define $\mathcal{C}_{1,2} = \mathcal{C}_1 \times \mathcal{C}_2 \times \mathcal{C}_3^C$, que acepta cuando $\mathcal{C}_1$ y $\mathcal{C}_2$ aceptan, y $\mathcal{C}_3$ rechaza (usando el complemento, dado que este acepta cuando el original rechaza).
- $\mathcal{C}_1$ **y** $\mathcal{C}_3$ **aceptan**: Se define $\mathcal{C}_{1,3} = \mathcal{C}_1 \times \mathcal{C}_3 \times \mathcal{C}_2^C$, que acepta cuando $\mathcal{C}_1$ y $\mathcal{C}_3$ aceptan, y $\mathcal{C}_2$ rechaza.
- $\mathcal{C}_2$ **y** $\mathcal{C}_3$ **aceptan**: Se define $\mathcal{C}_{2,3} = \mathcal{C}_2 \times \mathcal{C}_3 \times \mathcal{C}_1^C$, que acepta cuando $\mathcal{C}_2$ y $\mathcal{C}_3$ aceptan, y $\mathcal{C}_1$ rechaza.
- **Todos rechazan**: Se define $\mathcal{C}_{null} = \mathcal{C}_1^C \times \mathcal{C}_2^C \times \mathcal{C}_3^C$, que acepta cuando los 3 autómatas rechazan.

Con estos 4 nuevos autómatas, que entregan **todas las posibilidades de que el producto punto sea** 0**,** construimos finalmente el autómata $\mathcal{P}$, que **acepta cuando cualquiera de estos 4 acepta, determinando así si es que el producto punto de los próximos 2 vectores en la secuencia será** 0**:**

$$\mathcal{P} = (\mathcal{C}_{1,2}^C \times \mathcal{C}_{1,3}^C \times \mathcal{C}_{2,3}^C \times \mathcal{C}_{null}^C)^C$$

El DFA $\mathcal{P}$ fue definido utilizando las propiedades de autómatas y sus complementos y productos, para obtener la unión de los lenguajes aceptados por cada uno.

Ahora, hay que encontrar el **otro autómata** $\mathcal{A}$**,** que se encargará de revisar todos los pares de vectores y hará uso de $\mathcal{P}$ para determinar si algún par cumple con la propiedad del producto punto nulo. Para construirlo haremos uso del **NO determinismo,** que permite explorar todas las combinaciones de forma mucho más compacta que si fuera determinista. A continuación se ilustra el NFA que denominamos $\mathcal{A}$:

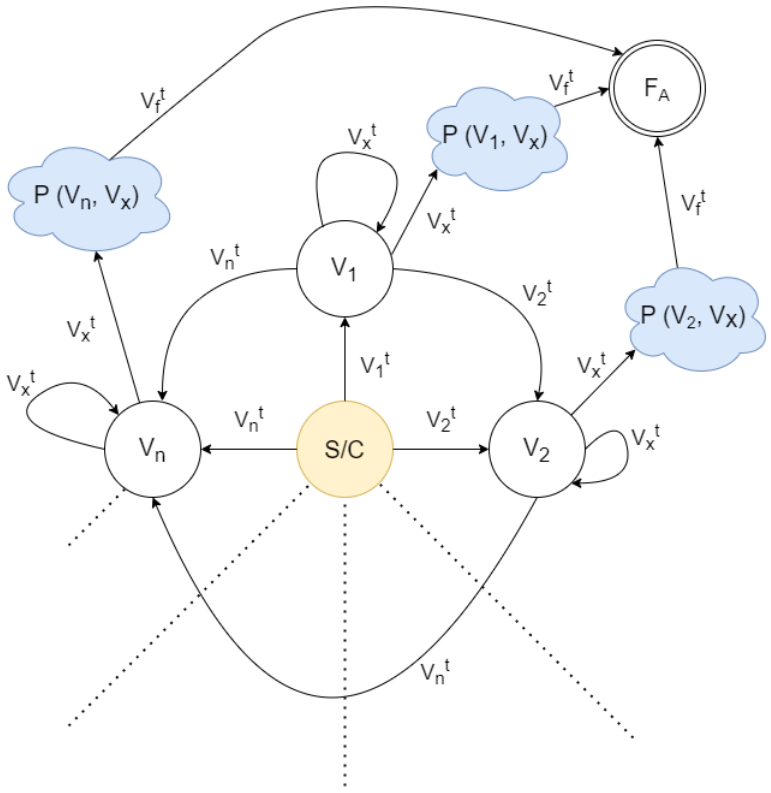

Figura 7: Autómata $\mathcal{A}$, que revisa todos los posibles pares de vectores y busca alguno que cumpla con la propiedad del producto punto nulo

Aquí se explica la simbología utilizada:

- S/C: Corresponde al **estado inicial,** antes de leer cualquier vector. Es importante notar que este autómata funciona en paralelo con $\mathcal{P}$, por lo que a partir de este estado ya es posible que decida seguir por la rama de ejecución que revisa el primer par de vectores en la secuencia.
- $V_{1...n}^t$: Corresponden a los n vectores que conforman el alfabeto. Se asume un orden a priori arbitrario de estos, que se mantiene en todo momento para efectos de este autómata.
- $V_{1...n}$: Corresponden a los estados asociados a cada vector del vocabulario. Encontrarse en el estado V_n **indica que el vector que se quiere comparar con el próximo de la secuencia de entrada es el vector V_n^t.** De esta forma, se puede **memorizar el vector actual** a comparar, mientras que el otro que conforma el par simplemente se obtiene al leer la secuencia de entrada.
- V_x^t: Corresponde a cualquier vector dentro del vocabulario, es decir, representa n transiciones posibles que llevan a un mismo estado.
- F_A: Corresponde al único **estado de aceptación** de este autómata.
- $\mathcal{P}(V_n, V_x)$: Corresponde a una **abstracción del diagrama,** donde el autómata decide seguir la rama de ejecución que compara el vector memorizado actualmente (V_n) con el vector que acaba de leer (V_x), **haciendo uso del autómata** $\mathcal{P}$ que está corriendo en paralelo a $\mathcal{A}$. En caso de que $\mathcal{P}$ acepte, significa que **se encontró un par de vectores que cumplen con lo pedido** (este caso exitoso se representa con el vector ficticio V_f^t dentro del dibujo), por lo que transiciona hacia el único estado de aceptación F_A, **aceptando finalmente la secuencia de entrada.** En caso de que $\mathcal{P}$ rechace, se

quedará atrapado en el estado R que fue descrito más arriba en la construcción de $\mathcal{P}$, por lo que dicha ejecución **será de rechazo.**

La gracia del autómata $\mathcal{A}$ es que al ser un NFA, al momento de estar parado en cualquiera de los estados que representan a los vectores (V_n), **tiene 3 opciones para continuar su ejecución:**

1. **Revisar la propiedad del producto punto:** Se continúa por la rama de ejecución de $\mathcal{P}$ (que estaba corriendo en paralelo), lo que permite determinar si para el vector memorizado actualmente (V_n) y el siguiente vector a leer (V_x) se cumple que su producto punto es 0. Esta opción **le da el poder a $\mathcal{A}$ para chequear si $\vec{v_i}^t \cdot \vec{v_j}$ es verdadero.**
2. **Quedarse en el mismo estado:** Esto prácticamente ignora al vector que se acaba de leer, manteniendo el mismo vector que se tenía en memoria **para poder generar otros pares que lo incluyan.**
3. **Avanzar al estado asociado al vector que se acaba de leer:** Esto ignora al par actual, y permite **buscar otros pares que incluyan al nuevo vector** leído desde la secuencia de entrada. Esto es posible ya que cada estado asociado a un vector, al leer un vector nuevo desde la secuencia, posee una transición hacia el estado asociado a dicho vector (en el dibujo sólo se ilustra para los vectores V_1, V_2 y V_n por temas de simplicidad).

Las opciones 2 y 3 en conjunto **permiten generar todos los posibles pares de vectores de la secuencia de entrada,** gracias a que el NO determinismo permitirá mantener un vector actual en memoria y asignar como pareja todos los otros que aparezcan en el input, o bien, moverse a otro vector de la secuencia y asignarle a él todos los otros que aparezcan más adelante, logrando computar las $\frac{n!}{(n-2)!\cdot 2!}$ combinaciones.

La opción 1 permite que, dado cualquier par de vectores asignado, se pueda **revisar si efectivamente cumplen con la propiedad pedida,** por lo que se podrá aceptar la secuencia de entrada en dicho caso. Además, como el NO determinismo permite explorar todas las ejecuciones antes de aceptar o rechazar, **siempre será posible explorar todas las combinaciones y encontrar alguna que acepte, de haberla, por lo que el NFA $\mathcal{A}$ logra aceptar al lenguaje L_2.**

Finalmente, tenemos que $\mathcal{L}(\mathcal{A}) = L_2$, y como se encontró un NFA que acepta al lenguaje, se tiene que es posible **determinizarlo para obtener un DFA** que también cumple con dicha expresión, por lo que L_2 corresponde a un **lenguaje regular** sobre el alfabeto $\{0,1\}^3$.