>>> IF013 - Fundamentos Teóricos de Informática
>>> Licenciatura de Sistemas - UNPSJB - Sede Trelew

Name: Celia Cintas[†], Pablo Navarro[‡], Samuel Almonacid[§]
Date: August 1, 2017

[†]cintas@cenpat-conicet.gob.ar, cintas.celia@gmail.com, @RTFMCelia
[‡]pnavarro@cenpat-conicet.gob.ar, pablo1n7@gmail.com
[§]almonacid@cenpat-conicet.gob.ar, almonacid.samuel.tw@gmail.com

>>> Unidad 1

1. Autómatas finitos. Reconocedores. Traductores. Diagrama de estados. Autómatas finitos no deterministas.
2. Equivalencia entre autómatas finitos deterministas y no deterministas. Morfismos sobre autómatas. Autómata Cociente.
3. Propiedades de lenguajes aceptados por Autómatas Finitos. Expresiones y lenguajes regulares.
4. Propiedades algebraicas de los lenguajes regulares. Equivalencia entre autómatas finitos y lenguajes regulares.
5. Teorema de Kleene. Gramáticas regulares. Relación entre gramáticas regulares y autómatas finitos.
6. Usos y aplicaciones de los autómatas finitos y lenguajes regulares.

>>> Lenguajes Regulares

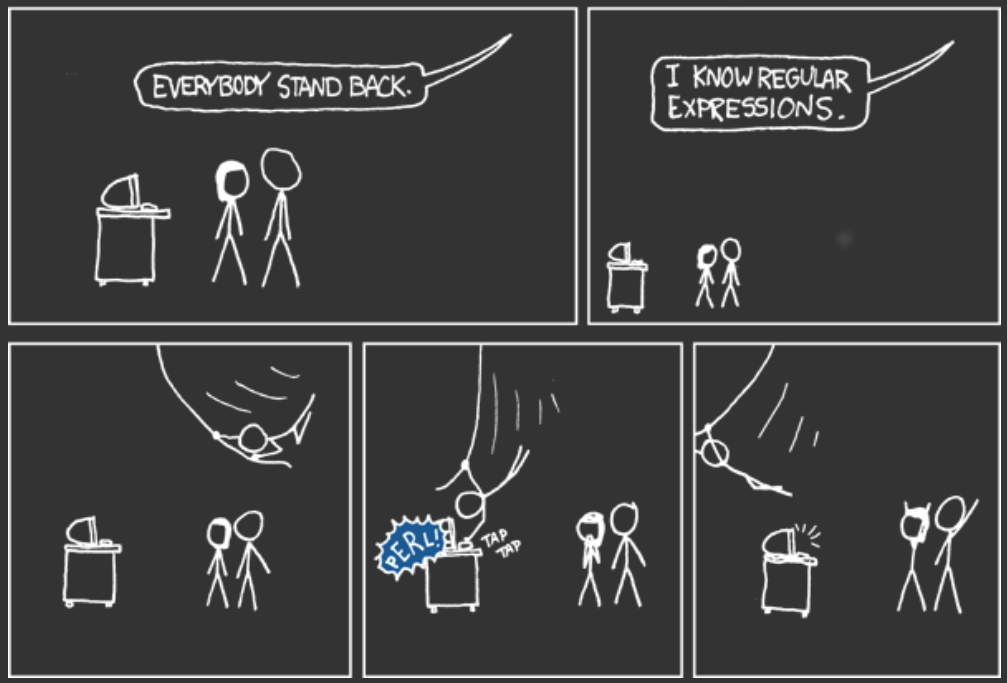
Son interesantes por su simplicidad, y a la vez porque incluyen muchos lenguajes relevantes en la práctica. Los mecanismos de búsqueda provistos por varios editores de texto (vim, emacs), así como por el shell de Unix y todas las herramientas asociadas para procesamiento de texto (sed, awk, perl), se basan en lenguajes regulares. Validación, *web scraping*, tokenización. Gran cantidad de lenguajes de programación cuentan con Expresiones Regulares nativas.Los lenguajes regulares también se usan en biología computacional para búsqueda en secuencias de ADN o proteínas.

>>> Lenguajes Regulares (Cont.)

Los lenguajes regulares se pueden describir usando tres mecanismos distintos:

* Expresiones Regulares (ERs).
* Autómatas Finitos Determinísticos (AFDs).
* Autómatas Finitos no determinísticos (AFNDs).

Algunos de los mecanismos son buenos para describir lenguajes, y otros para implementar reconocedores eficientes.



Definición

Una expresión regular (ER) sobre un alfabeto finito Σ se define recursivamente como sigue:

1. $\forall c \in \Sigma$, c es una ER.
2. $\emptyset$ es una ER.
3. λ es ER.
4. Si E_1 y E_2 son ERs. $E_1|E_2$ es una ER.
5. Si E_1 y E_2 son ERs. $E_1.E_2$ es una ER.
6. Si E_1 es ER. E_1^* es una ER.
7. Si E_1 es ER. (E_1) es una ER.

Definición

El lenguaje descrito por una ER E, $L(E)$, se define recursivamente como sigue:

1. Si $c \in \Sigma, L(c) = \{c\}$. Esto es un conjunto de una sola cadena de una sola letra.
2. $L(\emptyset) = \emptyset$.
3. $L(E_1|E_2) = L(E_1) \cup L(E_2)$.
4. $L(E_1.E_2) = L(E_1).L(E_2)$.
5. $L(E_1^*) = L(E_1)^*$.
6. $L((E_1)) = L(E_1)$.

Definición

Un lenguaje L es regular si existe una ER E tal que $L = L(E)$.

>>> Expresiones Regulares (Cont.)

Algunos ejemplos, sea $\Sigma = \{a, b\}$:

1. $a|b$ designa $\{a, b\}$
2. $(a|b)(a|b)$ designa $\{aa, ab, bb, ba\}$
3. a^* designa $\{\lambda, a, aa, aaa, aaaa, \cdots\}$
4. $(a|b)^*$ designa 0 o mas casos de una a o b.
5. $a|a^*b$

>>> Expresiones Regulares (Cont.)

Algunos ejercicios:

* Sea $\Sigma = \{a, b\}$. ¿Cómo se podría escribir una ER para las cadenas de a's y b's que contuvieran una cantidad impar de b's?
* ¿Cómo se podría escribir una ER para las cadenas de a's y b's que nunca contuvieran tres b's seguidas?

>>> Expresiones Regulares (Cont.)

Posibles Soluciones:

* Sea $\Sigma = \{a, b\}$. ¿Cómo se podría escribir una ER para las cadenas de a's y b's que contuvieran una cantidad impar de b's?
  $a^*(ba^*ba^*)^*ba^*$
* ¿Cómo se podría escribir una ER para las cadenas de a's y b's que nunca contuvieran tres b's seguidas?
  $(a|ba|bba)^*(\lambda|b|bb)$

Definición

Si dos Expresiones Regulares E_1 y E_2 representan el mismo lenguaje $(L(E_1) = L(E_2)$, se dice que son equivalentes $(E_1 \equiv E_2)$.

Leyes algebraicas para Expresiones Regulares, sean α, β y γ ER sobre el alfabeto Σ:

1. Asociatividad de la Union: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
2. Conmutatividad de la unión: $\alpha + \beta = \beta + \alpha$
3. Asociatividad de la concatenación: $(\alpha\beta)\gamma = \alpha(\beta\gamma)$
4. Elemento neutro de la concatenación: $\alpha\lambda = \lambda\alpha = \alpha$
5. Elemento neutro de la unión: $\alpha + \emptyset = \emptyset + \alpha = \alpha$
6. Distributividad de la concatenación respecto de la unión: $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$
7. $\lambda^* = \lambda$
8. $\emptyset^* = \lambda$
9. $\alpha^*\alpha^* = \alpha^*$

Ver lista de propiedades completas en el documento.

>>> Demo Time!

Expresiones Regulares en Python - Ver ipynb

>>> Autómatas Finitos

Un AFD es otro mecanismo para describir lenguajes. En vez de pensar en generar las cadenas (como las ERs), un AFD describe un lenguaje mediante reconocer las cadenas del lenguaje, y ninguna otra.

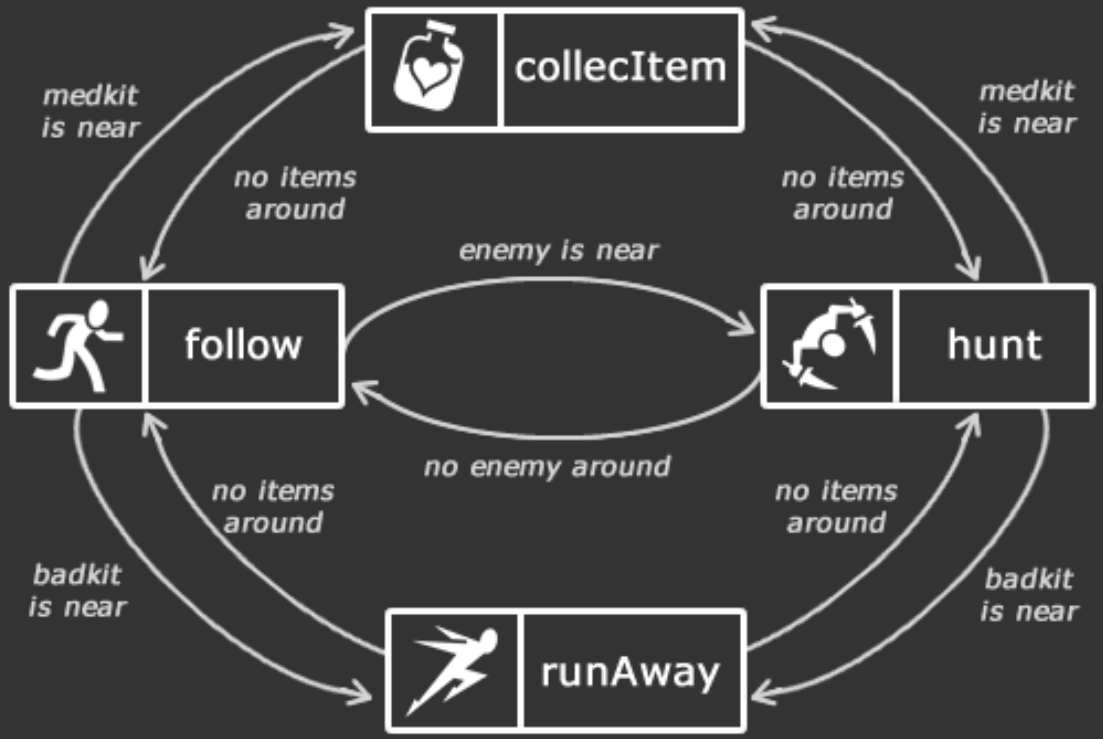


Figure: https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-squad-pattern-using-steering-behaviors--gamedev-13638

Definición

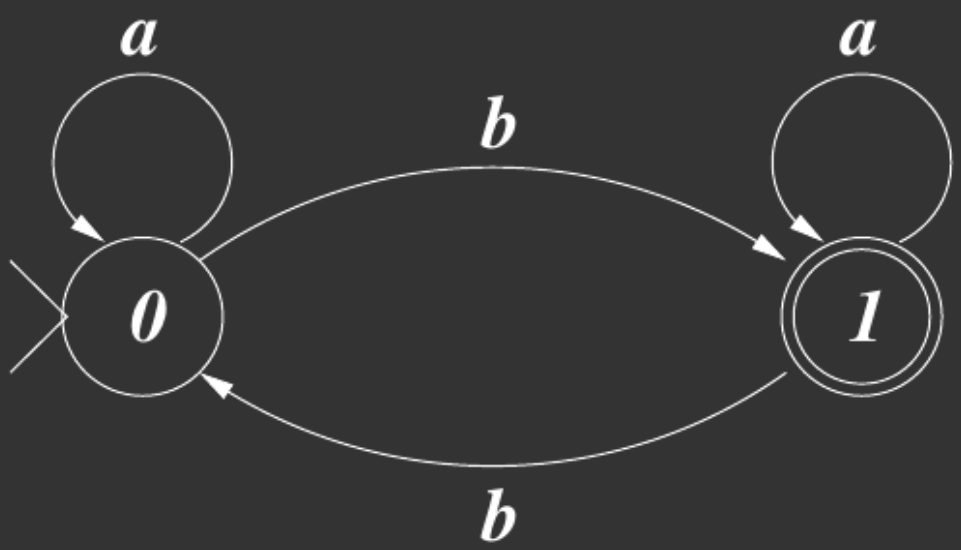
Un autómata finito determinístico (AFD) es una tupla $M = (K, \Sigma, \delta, S, F)$, tal que

* K es un conjunto finito de estados.
* Σ es un alfabeto finito.
* $S \in K$ es el estado inicial.
* $F \subseteq K$ son los estados finales.
* $\delta : K \times \Sigma \to K$ es la función de transición.

Qué reconoce este AFD ?

$M = (K, \Sigma, \delta, S, F)$, donde $K = \{0, 1\}$, $\Sigma = \{a, b\}$, $S = 0$, $F = \{1\}$ y la función δ se define cómo:

δ	0	1
a	0	1
b	1	0



Definición

La función de transición puede extenderse para que acepte como 2do argumento cadenas en Σ, o sea $\hat{\delta} : K \times \Sigma^* \to K$, definiendola de la siguiente manera, como func. generalizada:

$$\hat{\delta}(q, \lambda) = q$$
$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a) \text{ con } x \in \Sigma^* \text{ y } a \in \Sigma$$

Notar que bajo esta definición:

$$\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \lambda), a) = \delta(q, a)$$

δ	0	1
a	0	1
b	1	0

$\delta(0, a) = 0; \delta(0, b) = 1, \delta(1, a) = 1; \delta(1, b) = 0$

Definición

Se dice que una cadena x es aceptada por un AFD $M = (K, \Sigma, \delta, S, F)$ si y solo si $\hat{\delta}(S, x) \in F$

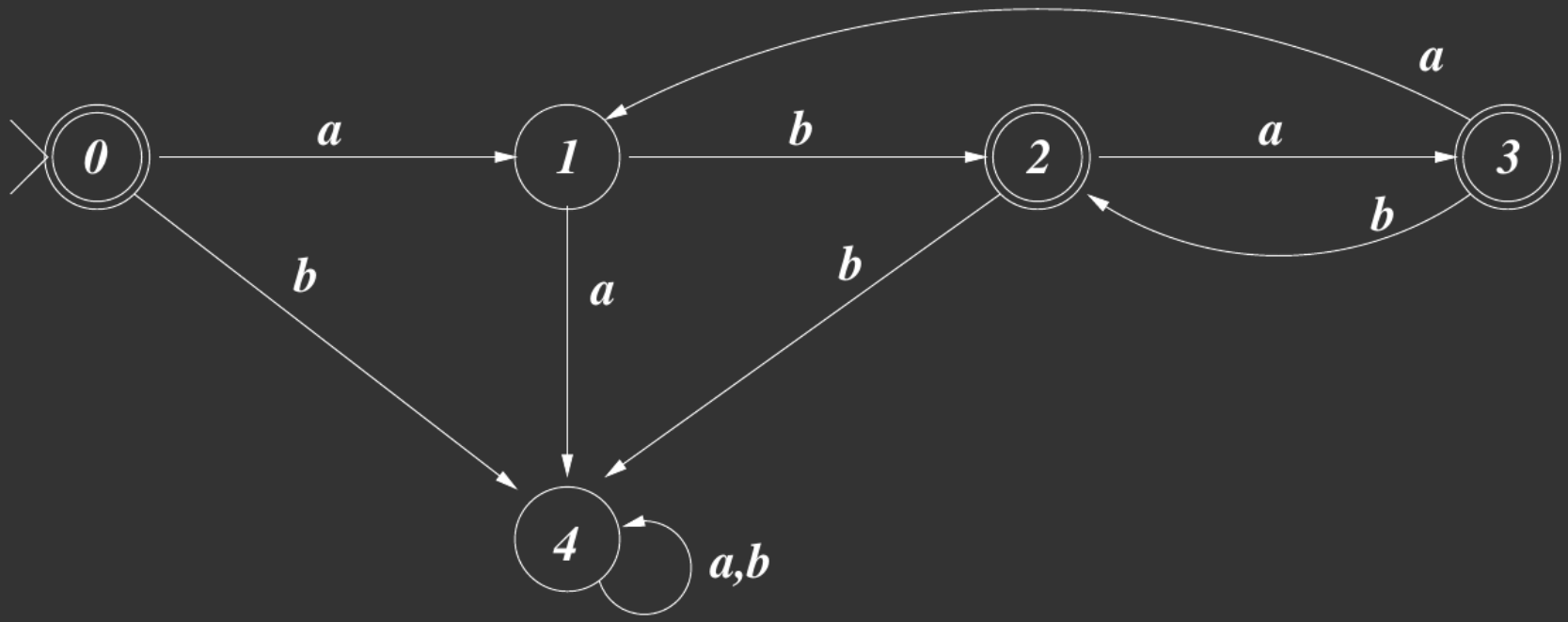
Definición

Dado un AFD $M = (K, \Sigma, \delta, S, F)$, el lenguaje aceptado por M, el cuál se denota $L(M)$, es el conjunto de cadenas aceptadas por M y se define cómo:

$$L(M) = \{x | \delta(S, x) \in F\}$$

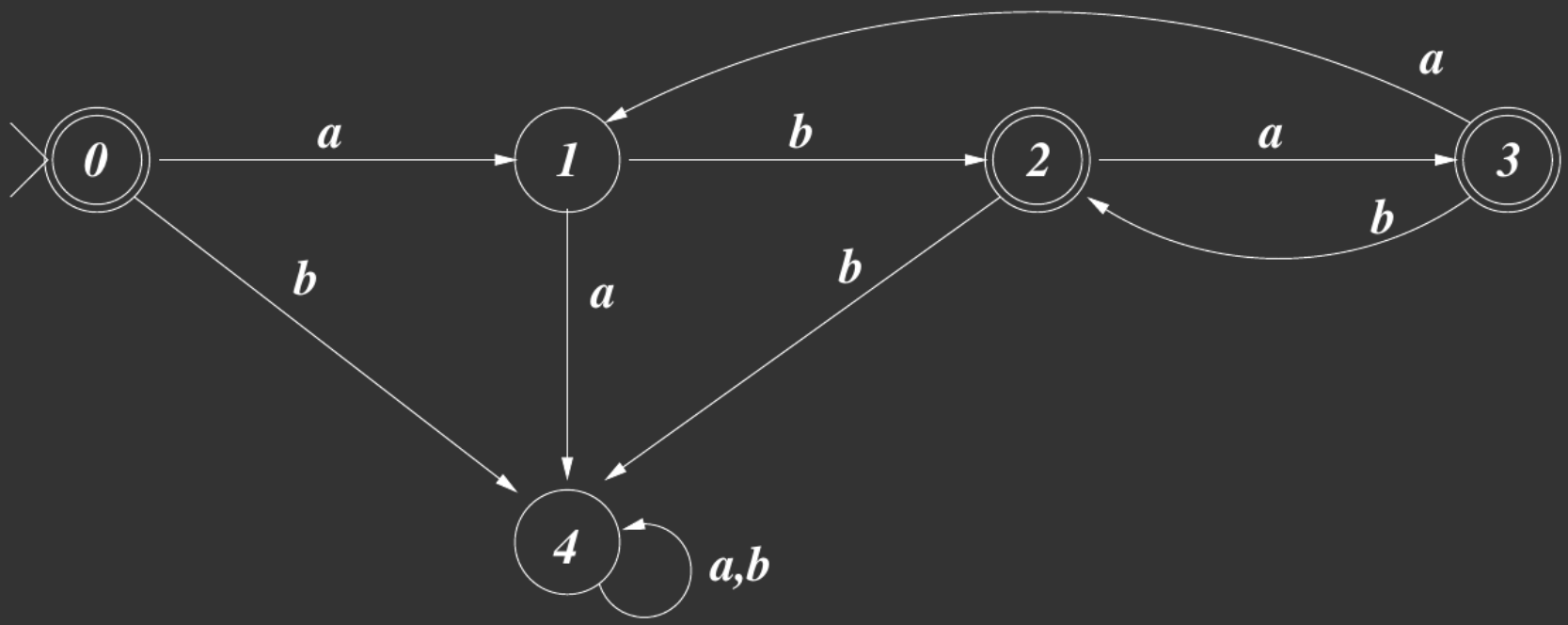
>>> Autómatas Finitos (Cont.)

Qué procesa este autómata? Qué función tiene el estado 4?



>>> Autómatas Finitos (Cont.)

Qué procesa este autómata? Qué función tiene el estado 4?

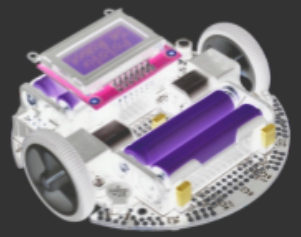


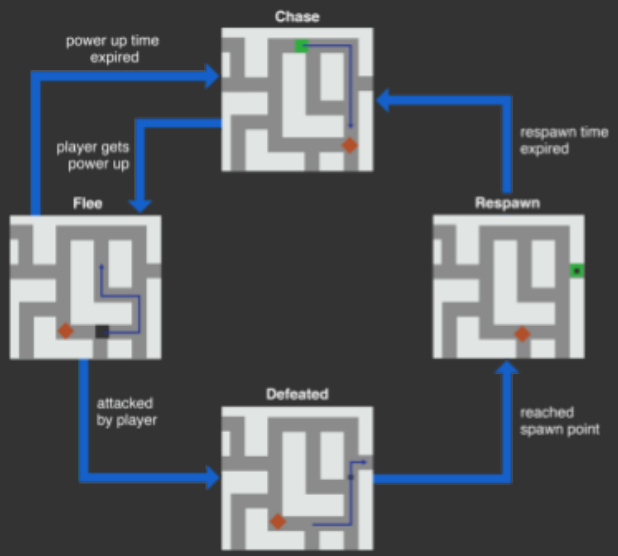
$$(ab|aba)^*$$

>>> Demo Time!

Autómatas Finitos en Python - Ver ipynb

>>> Autómatas Finitos (Cont.)





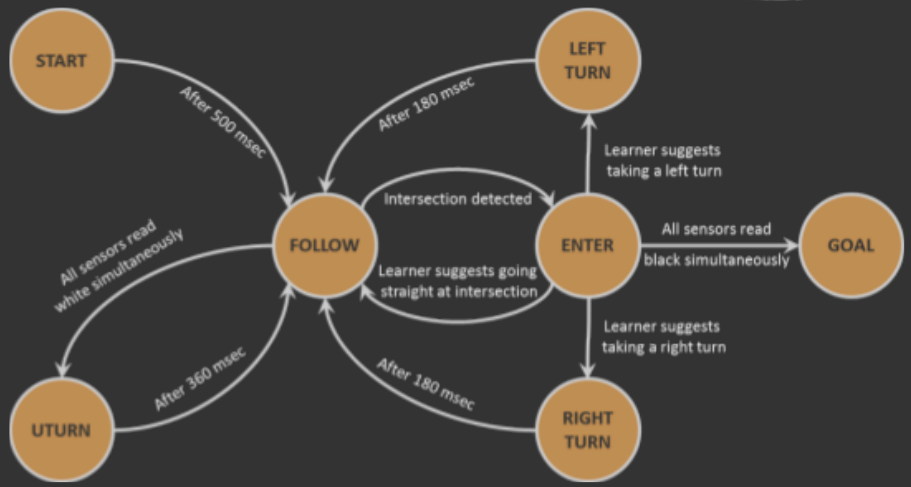


Figure: https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-squad-pattern-using-steering-behaviors--gamedev-13638, https://svaish3cs3630.wordpress.com/2013/01/21/assignment-1-robot-kinematics-and-fsm/, https://developer.apple.com/library/content/documentation/General/Conceptual/GameplayKit_Guide/StateMachine.html#//apple_ref/doc/uid/TP40015172-CH7-SW3

>>> Autómatas Finitos Traductores (AFT)

Hasta ahora hemos tratado los autómatas finitos como reconocedores de cadenas que pertenecen a un lenguaje donde su aceptación o rechazo está dado por el estado en que termina la computación. Ahora vamos a ver AF que para computar funciones, donde la función de salida es una función compleja que mapea o traduce cadenas de entrada en cadenas de salida.

Definición

Un autómata finito traductor de Mealy es una sextupla $M = (K, \Sigma, \delta, Z, f_0)$ donde $f_0 : k \times \Sigma \to Z$. La función de salida se indicará en la tabla de estados con una columna extra:

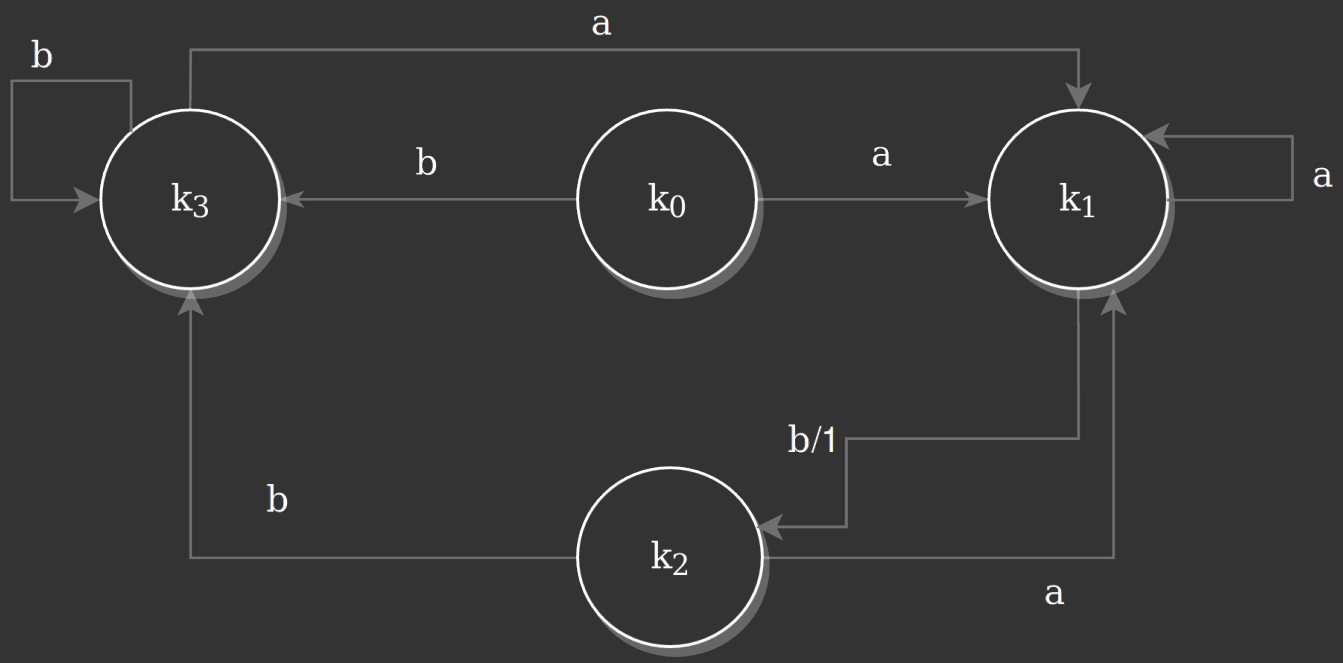
estados	$entrada_0$	$entrada_1$
k_0	k_i/z_j	k_{i+1}/z_j
$\dots$	$\dots$	$\dots$

>>> Maquinas de Mealy (AFT Mealy) (Cont.)

Cómo contamos las apariciones de la subcadena ab??.

$$M = (K, \Sigma, \delta, Z, f_0)$$
$$K = \{k_0, k_1, k_2, k_3\}$$
$$\Sigma = \{a, b\}$$
$$Z = \{1\}$$

δ	a	b
k_0	$k_1/-$	$k_3/-$
k_1	$k_1/-$	$k_2/1$
k_2	$k_1/-$	$k_3/-$
k_3	$k_1/-$	$k_3/-$



Definición

Un autómata finito traductor de Moore es una sextupla $M = (K, \Sigma, \delta, Z, f_0)$. Donde $f_0 : K \to Z$. Aquí Z es el alfabeto de salida y puede variar de acuerdo al problema planteado.
La función de salida se indicará en la tabla de estados con una columna extra:

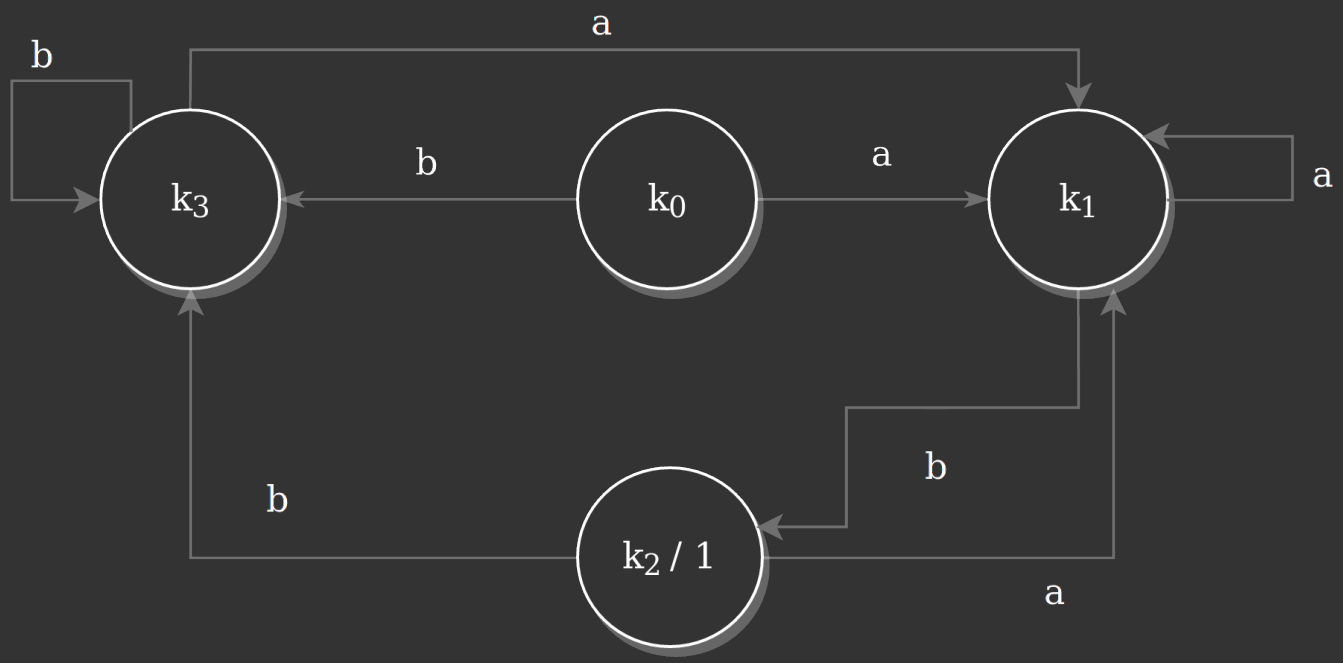
estados	entradas	salida
k_0	$\cdots$	z_0
k_1	$\cdots$	z_1
$\cdots$	$\cdots$	$\cdots$

>>> Maquinas de Moore (AFT Moore) (Cont.)

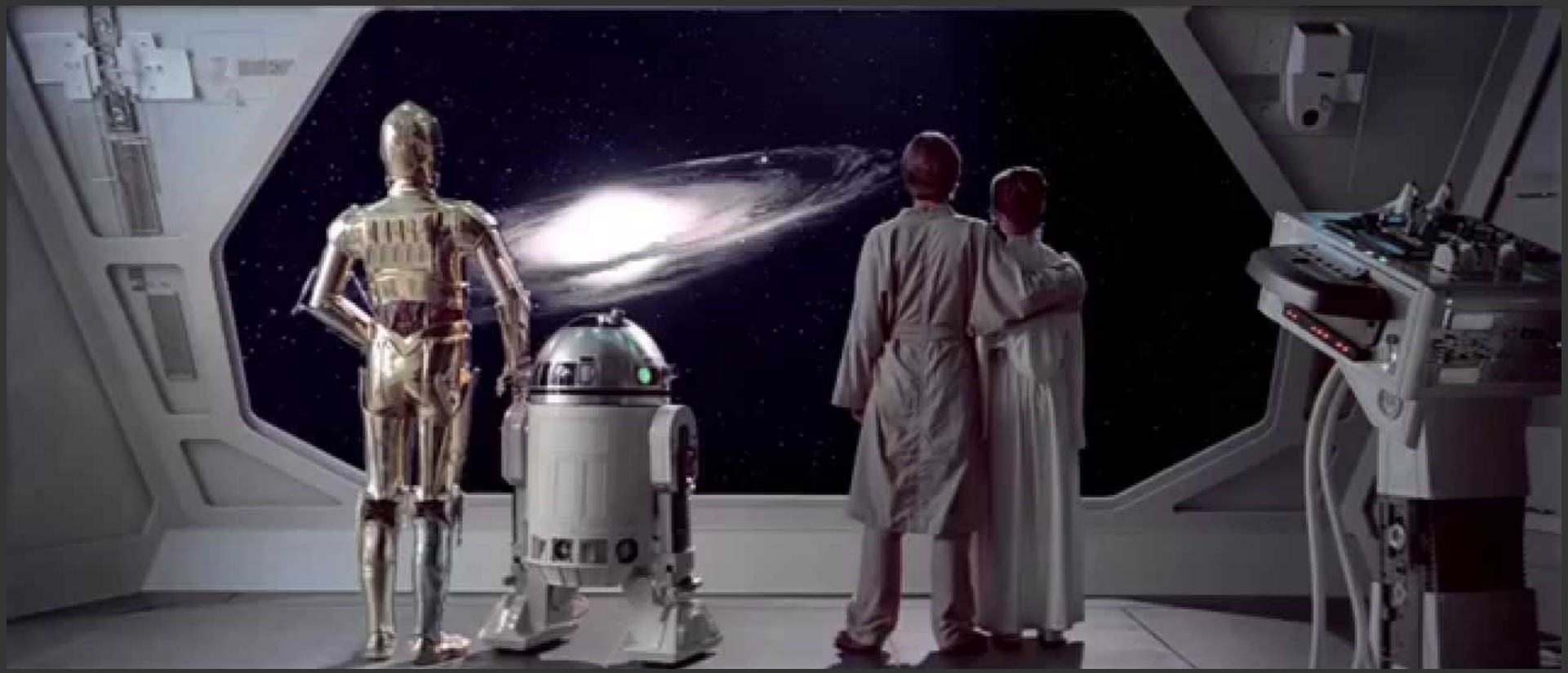
Mismo ejemplo que el anterior.

$$M = (K, \Sigma, \delta, Z, f_0)$$
$$K = \{k_0, k_1, k_2, k_3\}$$
$$\Sigma = \{a, b\}$$
$$Z = \{1\}$$

δ	a	b	f_0
k_0	k_1	k_3	-
k_1	k_1	k_2	-
k_2	k_1	k_3	1
k_3	k_1	k_3	-



>>> Gracias!



Bibliografía

1. Introduction to Automata Theory, Languages, and Computation - Hopcroft et. al 2007 (3er ed.)
2. Teoría de la Computación - Gonzalo Navarro 2011.
3. Fundamentos de Cs. de la Computación - Juan Carlos Augusto 1995.