AUTÓMATAS

Regex convertido en robot

Ing. Max Cerna

Tips de software

- Mantenlo simple
- No optimice prematuramente
- Diseñe sistemas que puedan probarse
- Es más fácil modificar un sistema que funciona que hacer que un sistema funcione

Agenda

1. Especificación léxica
2. Autómata finito
3. Expresiones hacia autómatas
4. NFA hacia DFA
5. Implementación de un autómata finito

Objetivo

Convertir expresiones regulares en autómatas finitos

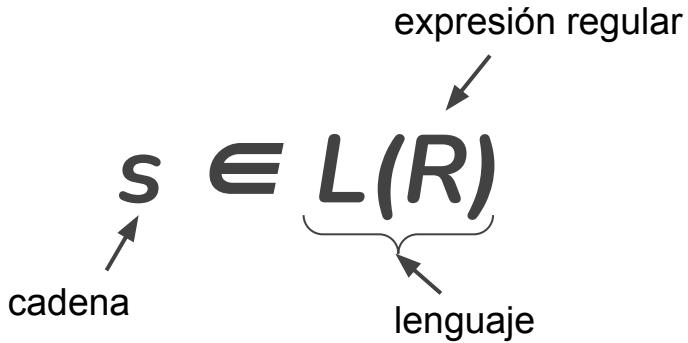
Especificación Léxica

Especificación Léxica

Al menos uno	AA*	A+
Unión	A+B	A|B
Opción	A+ε	A?
Rango	‘a’+’b’+…+’z’	[a − z]
Rango (exclusión) - Complemento de [a − z]		[^a − z]

Especificación Léxica

¿Como saber si?



Especificación Léxica

¿Como saber si?

$$s \in L(R)$$

Una respuesta de sí o no, no es suficiente

- dividir la entrada en tokens
- Adaptar las expresiones regulares a este objetivo.

Especificación Léxica

1) Escribir una regex para los lexemas de cada clase (categoría) de token
   - Número = digit+
   - Palabra clave = ‘if’ + ‘else’ + ...
   - Identificador= letter (letter + digit)*
   - OpenPar = ‘(‘
   - ...

Especificación Léxica

2) Construir R, coincidiendo con todos los lexemas para todos los tokens

R = Keyword + Identifier + Number + ...

= R1 + R2 + ...

R debe ser capaz de coincidir con cualquier secuencia de caracteres que forme un lexema válido para cualquier tipo de token que estemos interesados en reconocer.

Combinamos expresiones regulares más pequeñas que representan los diferentes tipos de tokens.

Especificación Léxica

3) Considerando la entrada x1...xn

   Para $1 \leq i \leq n$ verificar

   $x1...xi \in L(R)$

4) Si éxito, entonces sabemos que:

   $x1...xn \in L(Rj)$ para algún j

5) Eliminar x1...xi de la entrada e ir a (3)

Especificación Léxica

¿Es utilizada toda la entrada?

Especificación Léxica

¿Cual token se debe utilizar?

Especificación Léxica

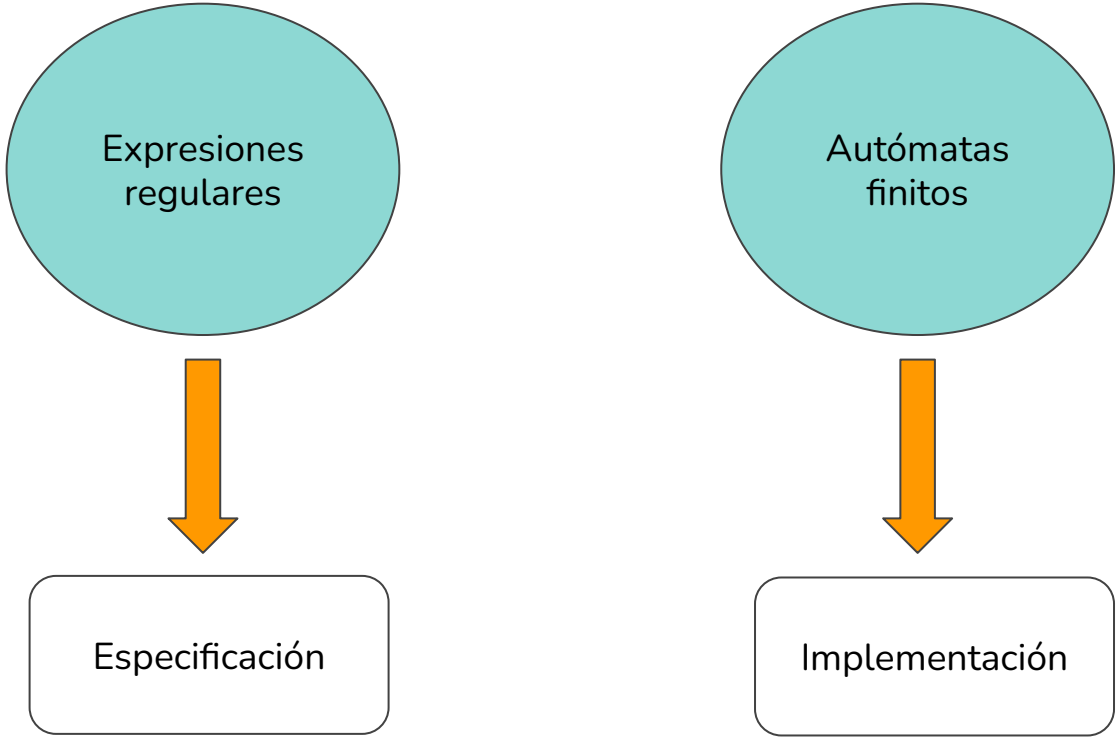
¿Y si la cadena no coincide?

Resumen

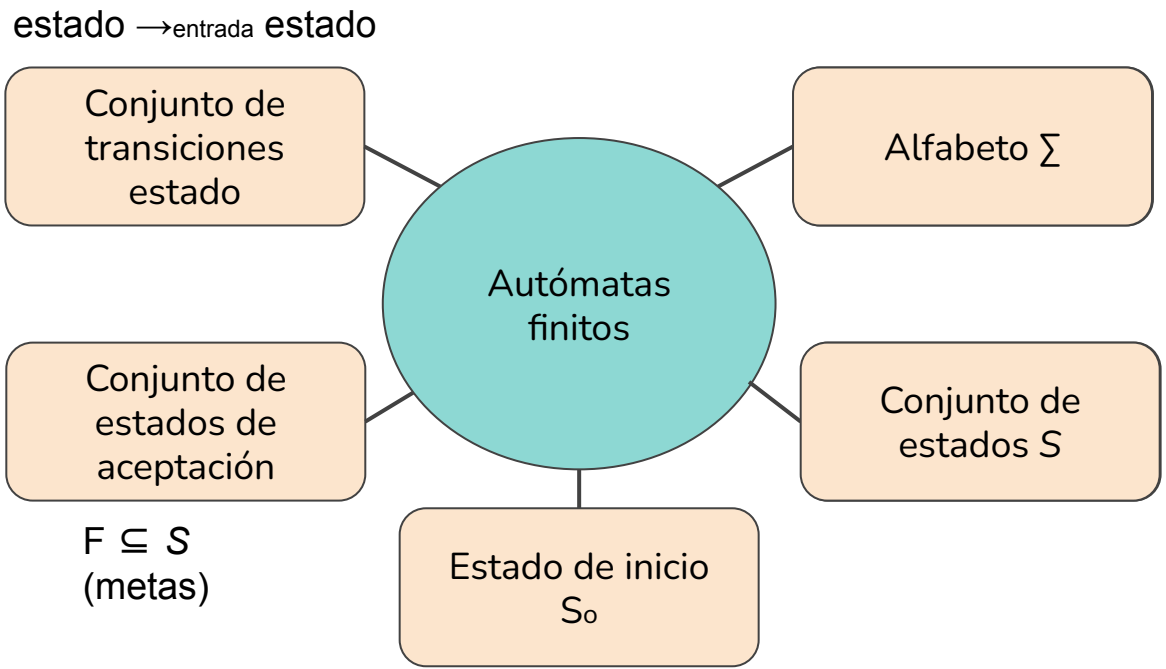
- Las expresiones regulares son una notación concisa para patrones de cadenas.
- El uso en análisis léxico requiere pequeñas extensiones.
  - Para resolver ambigüedades
  - Para manejar errores
- Buenos algoritmos conocidos
  - Requerir solo una pasada sobre la entrada
  - Pocas operaciones por carácter (búsqueda de tabla)

Autómata finito

Autómata finito



Autómata finito



Autómata finito

Transición

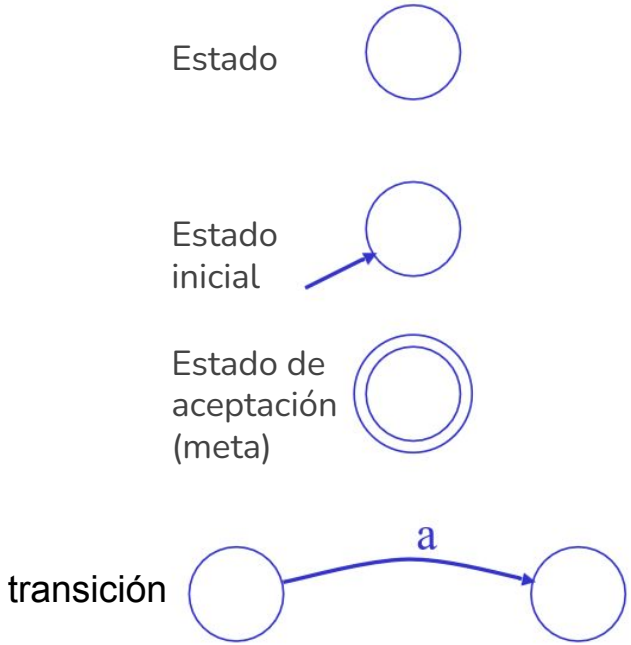
$$s_1 \rightarrow^a s_2$$

Se interpreta como:

"En el estado s1 con la entrada a dirigirse hacia s2"

- Si finaliza la entrada y está en estado de aceptación entonces ***aceptar***
- De lo contrario entonces ***rechazar***

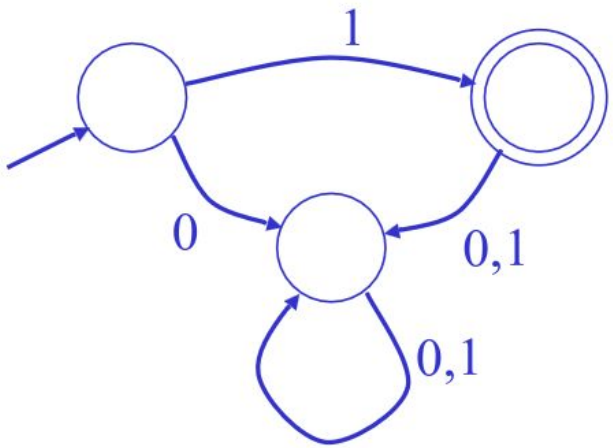
Autómata finito - Notación



Autómata finito - Ejemplo

Un autómata finito que solo acepta “1“

Autómata finito - Ejemplo

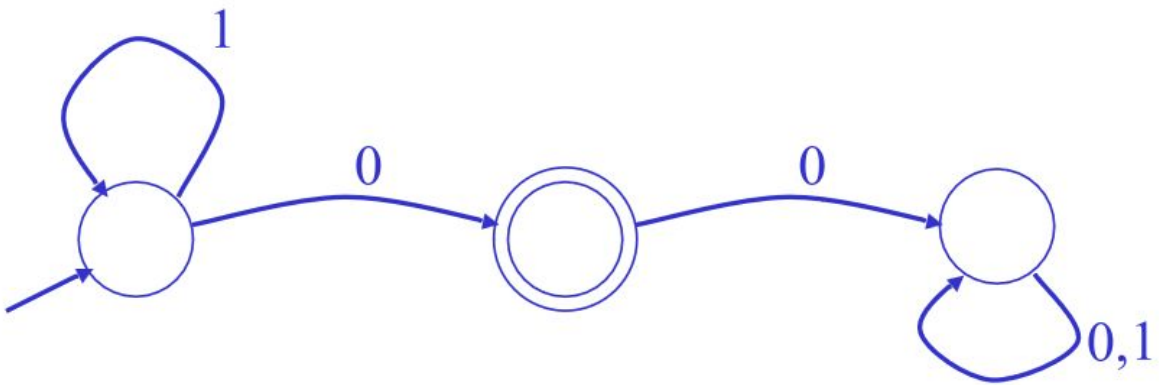


Autómata finito - Ejemplo

Un autómata finito que acepta cualquier cantidad de ”1” seguido de un 0 simple.

Alfabeto: 0,1

Autómata finito - Ejemplo



Autómata finito - Ejemplo

Seleccione el lenguaje regular que denota el mismo lenguaje que este autómata finito:

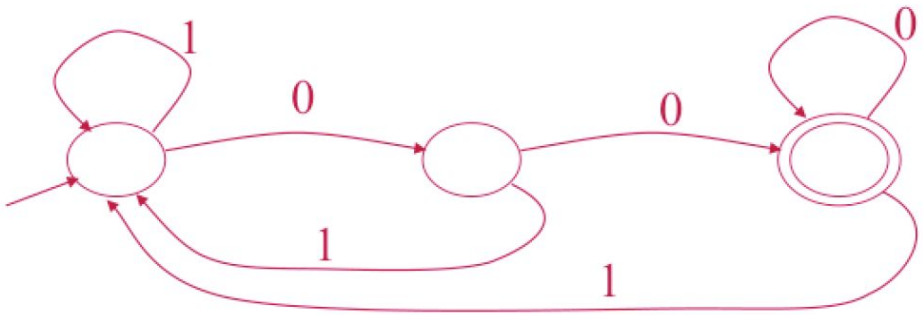
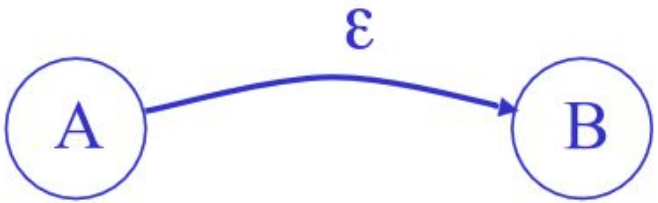


Figura: Automata Finito

- (0 + 1)∗
- (1 ∗ +0)(1 + 0)
- 1 ∗ +(01) ∗ +(001) ∗ +(000 ∗ 1)∗
- (0 + 1) ∗ 00

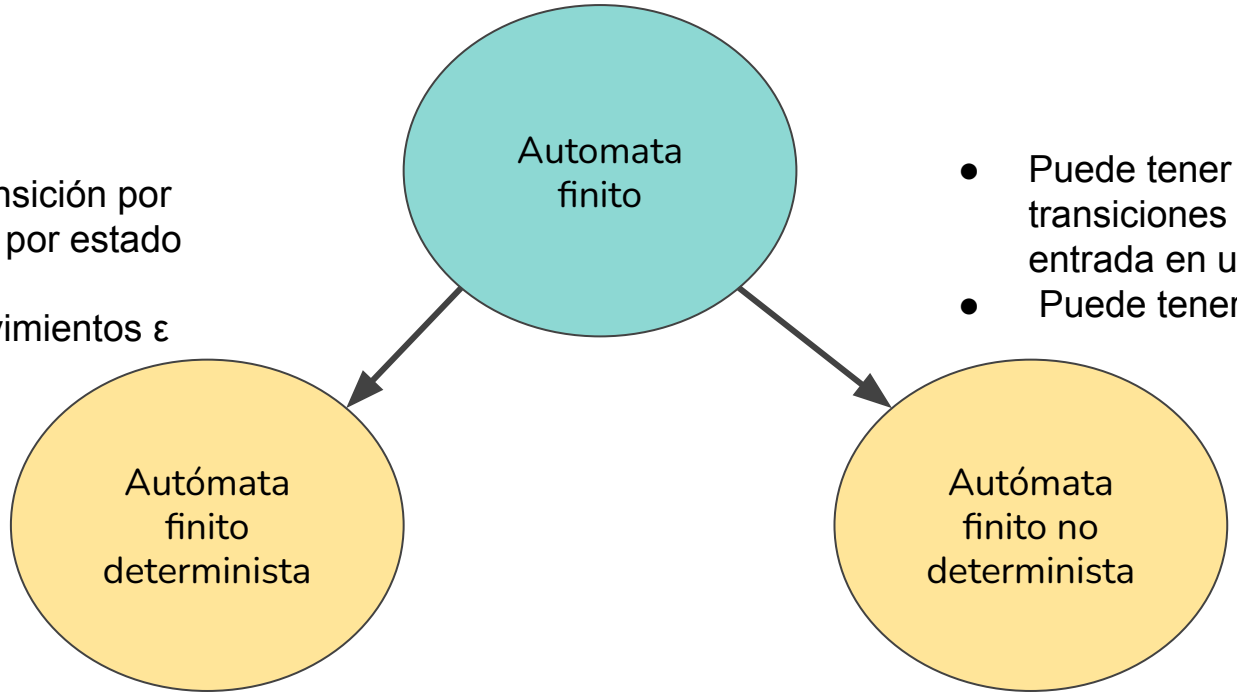
Autómata finito - Epsilon



La máquina puede pasar del estado A al estado B, sin leer la entrada

Sólo existen en NFA

Autómata finito



- Una transición por entrada por estado
- Sin movimientos ε

- Puede tener múltiples transiciones para una entrada en un estado dado
- Puede tener movimientos ε

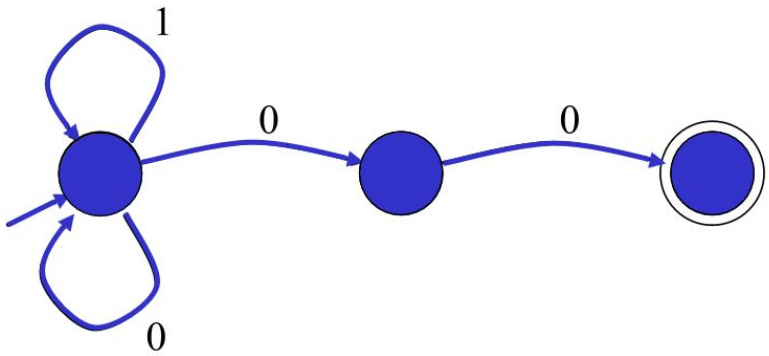
Autómata finito

- Un DFA toma solo un camino a través del gráfico de estado

- Un NFA puede elegir

Autómata finito

- Un NFA puede desplazarse hacia varios estados



NFA vs DFA

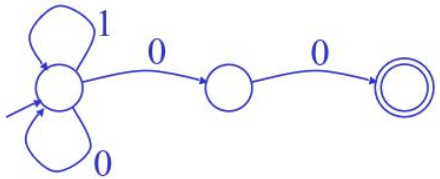
- NFA y DFA reconocen el mismo conjunto de idiomas regulares

- Los DFA son más rápidos de ejecutar, no hay opciones a considerar.

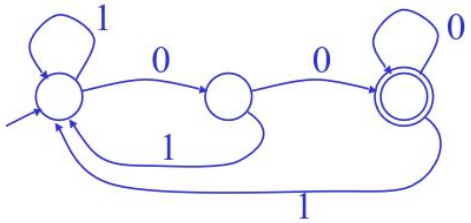
NFA vs DFA

- Para un idioma determinado, NFA puede ser más simple que DFA

NFA



DFA



Expresiones hacia Autómatas

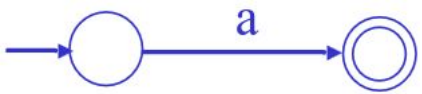
Para cada regex, definir un NFA

Notación: NFA para regex M

Expresiones hacia Autómatas

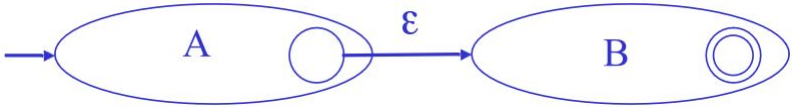
Para ε

Para la entrada a

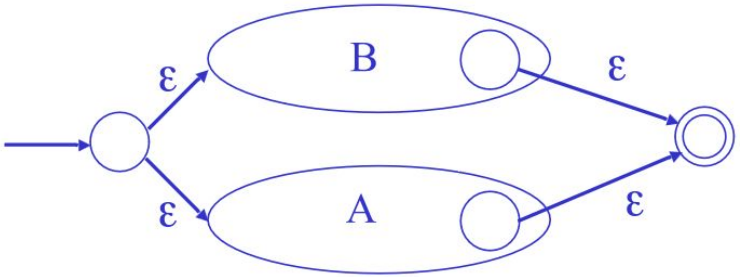


Expresiones hacia Autómatas

Para AB

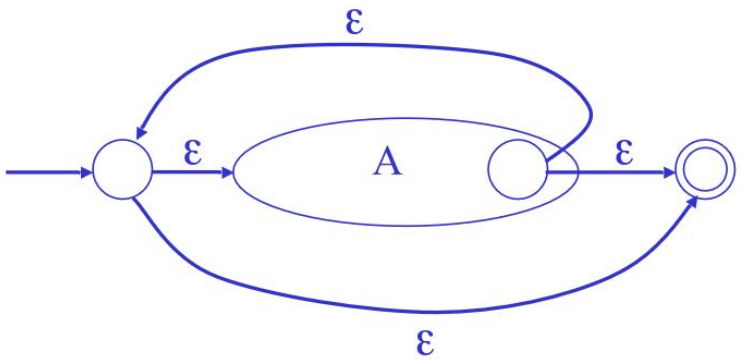


Para A+B



Expresiones hacia Autómatas

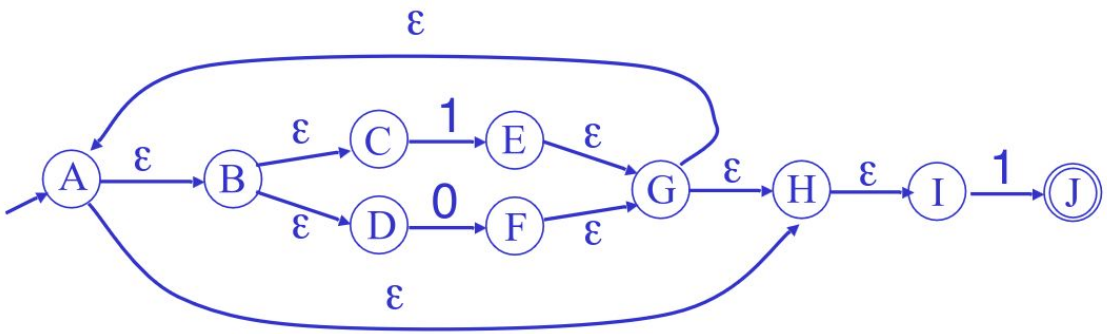
Para A*



Expresiones hacia Autómatas

Considere la expresión regular (1+0)*1

Expresiones hacia Autómatas



NFA hacia DFA

NFA a DFA: El Truco

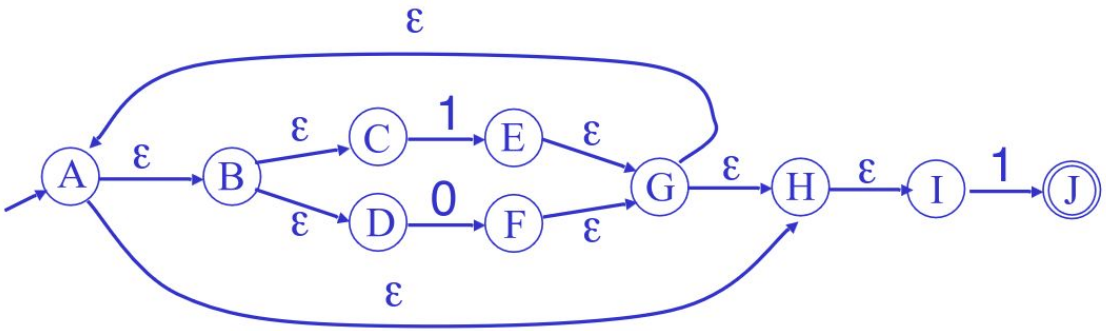
- Construcción del NFA
- Cada estado del DFA = un subconjunto no vacío de estados del NFA
- Estado inicial = el conjunto de estados del NFA accesibles a través de movimientos ε desde el estado inicial del NFA

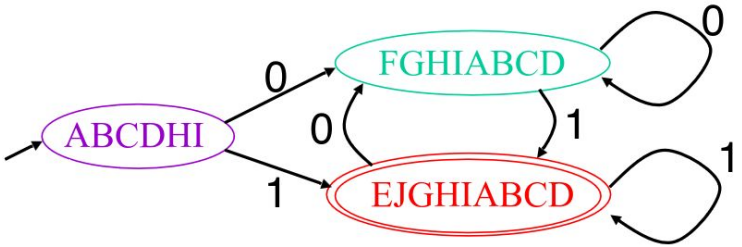
NFA HACIA DFA

- Un NFA puede estar en varios estados en cualquier momento

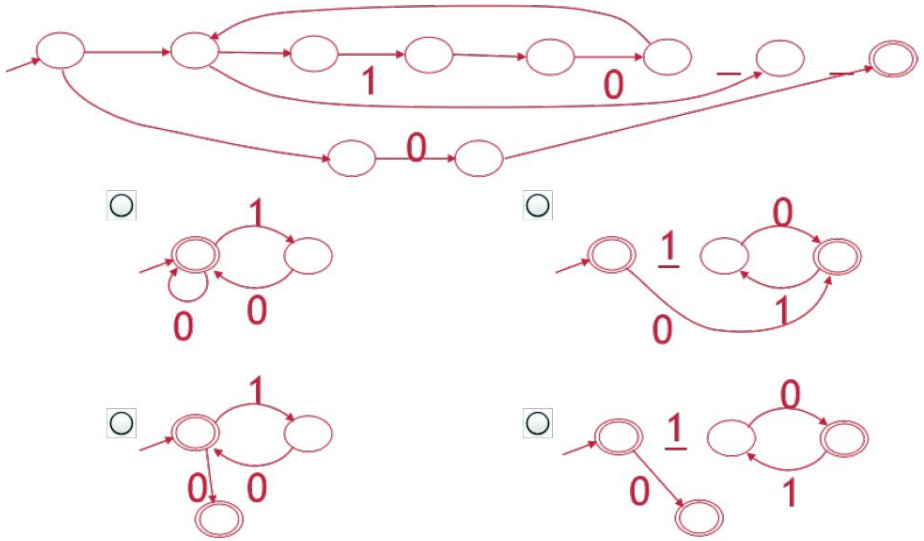
- ¿Cuantos?

NFA HACIA DFA





NFA HACIA DFA



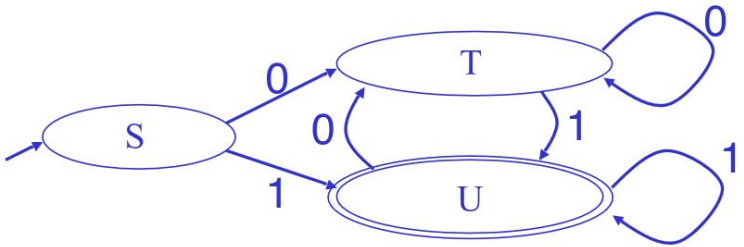
Implementación de un autómata finito

Implementación de un autómata finito

Un DFA puede ser implementado mediante una tabla “T” 2D

- Una dimensión son los "estados"
- Otra dimensión es el símbolo de entrada
- Para cada transición $Si \rightarrow a\ Sk$ define $T[i, a] = k$

Implementación de un autómata finito



	0	1
S	T	U
T	T	U
U	T	U

Implementación de un autómata finito

La conversión de NFA a DFA es el núcleo de herramientas como flex.

Sin embargo, los DFA pueden ser muy grandes.

En la práctica, las herramientas tipo flex sacrifican velocidad por espacio en la elección de las representaciones de NFA y DFA.