

Teoría de la Computación

No determinismo

NFA, transiciones ε, construcción de subconjuntos y propiedades de cierre

Gerson Garrido

Semana 2

UNIVERSIDAD NACIONAL DE INGENIERÍA

Hoy

Sesión A

1. Un diseño que con un DFA cuesta
2. Qué cambia en un NFA
3. Tres maneras de pensar el no determinismo
4. La definición formal
5. Transiciones ε y clausura ε
6. El conjunto de estados activos

Sesión B

7. La construcción de subconjuntos
8. Por qué funciona: la equivalencia
9. El costo: hasta 2^n estados
10. Unión, concatenación, estrella, reverso
11. El complemento no se hace en el NFA

Lectura

Apuntes de la semana 2. Apoyo: Sipser, sec. 1.2; Erickson, *Models of Computation*, nota 4.

SECCIÓN 01

Autómatas finitos no deterministas

01

El tercer símbolo desde el final es 1

$$T = \{\, w \in \{0,1\}^* : \text{el tercer símbolo de } w \text{ desde el final es } 1 \,\}$$

- Están: `100`, `0110`, `1111`. No están: `011`, `0010`, ni las de longitud menor que 3.

El tercer símbolo desde el final es 1

$$T = \{\, w \in \{0,1\}^* : \text{el tercer símbolo de } w \text{ desde el final es } 1 \,\}$$

- Están: `100`, `0110`, `1111`. No están: `011`, `0010`, ni las de longitud menor que 3.
- Método de la semana 1: ¿qué hay que recordar?

El tercer símbolo desde el final es 1

$$T = \{\, w \in \{0,1\}^* : \text{el tercer símbolo de } w \text{ desde el final es } 1 \,\}$$

- Están: `100`, `0110`, `1111`. No están: `011`, `0010`, ni las de longitud menor que 3.
- Método de la semana 1: ¿qué hay que recordar?
- Cuando llega un `1`, la máquina no sabe si será el tercero desde el final: no ve cuánto falta.

El tercer símbolo desde el final es 1

$$T = \{\, w \in \{0,1\}^* : \text{el tercer símbolo de } w \text{ desde el final es } 1 \,\}$$

- Están: `100`, `0110`, `1111`. No están: `011`, `0010`, ni las de longitud menor que 3.
- Método de la semana 1: ¿qué hay que recordar?
- Cuando llega un `1`, la máquina no sabe si será el tercero desde el final: no ve cuánto falta.
- Tiene que recordar los **tres últimos símbolos**, completos: $2^3 = 8$ estados, 16 transiciones.

El tercer símbolo desde el final es 1

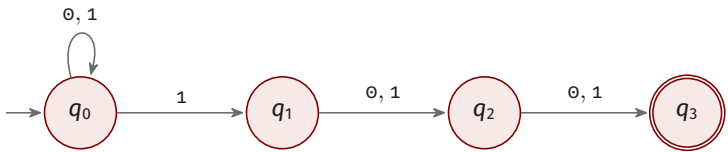
$$T = \{\, w \in \{0,1\}^* : \text{el tercer símbolo de } w \text{ desde el final es } 1 \,\}$$

- Están: `100`, `0110`, `1111`. No están: `011`, `0010`, ni las de longitud menor que 3.
- Método de la semana 1: ¿qué hay que recordar?
- Cuando llega un `1`, la máquina no sabe si será el tercero desde el final: no ve cuánto falta.
- Tiene que recordar los **tres últimos símbolos**, completos: $2^3 = 8$ estados, 16 transiciones.

Y si fuera el décimo desde el final

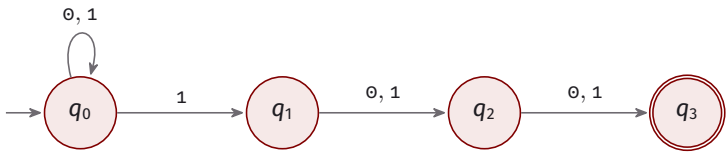
$2^{10} = 1024$ estados. El diseño no se parece a la descripción del lenguaje.

Si la máquina pudiera adivinar



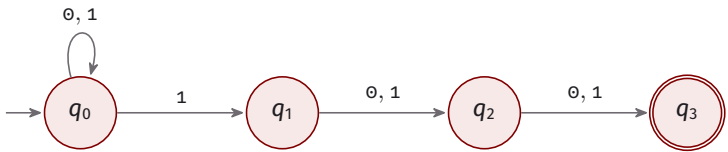
- q_0: todavía no apostó. Al leer un `1` puede **apostar**: “este es el tercero desde el final”.

Si la máquina pudiera adivinar



- q_0: todavía no apostó. Al leer un `1` puede **apostar**: “este es el tercero desde el final”.
- q_1, q_2, q_3: comprueba la apuesta leyendo dos símbolos más.

Si la máquina pudiera adivinar



- q_0: todavía no apostó. Al leer un `1` puede **apostar**: “este es el tercero desde el final”.
- q_1, q_2, q_3: comprueba la apuesta leyendo dos símbolos más.
- Cuatro estados. Pero no es un DFA: de q_0 salen dos flechas con `1`, y de q_3 no sale ninguna.

Qué cambia en un NFA

Se dibuja igual que un DFA. Cambian tres reglas sobre las flechas:

1. **Varias flechas con el mismo símbolo.** La máquina puede seguir cualquiera.

Qué cambia en un NFA

Se dibuja igual que un DFA. Cambian tres reglas sobre las flechas:

1. **Varias flechas con el mismo símbolo.** La máquina puede seguir cualquiera.
2. **Ninguna flecha.** Ese intento se acaba: el camino **muere**. No hace falta estado trampa.

Qué cambia en un NFA

Se dibuja igual que un DFA. Cambian tres reglas sobre las flechas:

1. **Varias flechas con el mismo símbolo.** La máquina puede seguir cualquiera.
2. **Ninguna flecha.** Ese intento se acaba: el camino **muere**. No hace falta estado trampa.
3. **Flechas con ε.** Se siguen sin leer nada de la entrada.

Qué cambia en un NFA

Se dibuja igual que un DFA. Cambian tres reglas sobre las flechas:

1. **Varias flechas con el mismo símbolo.** La máquina puede seguir cualquiera.
2. **Ninguna flecha.** Ese intento se acaba: el camino **muere**. No hace falta estado trampa.
3. **Flechas con ε.** Se siguen sin leer nada de la entrada.

> **Idea clave: qué significa aceptar**
>
> Un NFA acepta una cadena si **existe al menos un** camino que la lee completa y termina en un estado de aceptación. La rechaza solo si **ningún** camino lo logra.

El archivo y la tabla

machines/third_from_end.nfa

```
states: q0 q1 q2 q3
alphabet: 0 1
start: q0
accept: q3
delta:
q0 0 q0
q0 1 q0 q1
q1 0 q2
q1 1 q2
q2 0 q3
q2 1 q3
```

terminal

```
$ build/show_nfa machines/third_from_end.nfa
Q  = {q0, q1, q2, q3}
Σ  = {0, 1}
q0 = q0
F  = {q3}

     δ   0      1        ε
->   q0  {q0}   {q0,q1}  {}
     q1  {q2}   {q2}     {}
     q2  {q3}   {q3}     {}
   * q3  {}     {}       {}
```

- Una línea puede tener **varios destinos**. Un par puede no tener línea: q_3.
- Cada casilla es un **conjunto** de estados. {} es $\emptyset$.

Tres maneras de pensar el no determinismo

1. **Adivinar bien.** Si alguna elección lleva a aceptar, la máquina la toma. Sirve para *diseñar*: “adivina dónde está el tercero desde el final, y comprueba”.

Tres maneras de pensar el no determinismo

1. **Adivinar bien.** Si alguna elección lleva a aceptar, la máquina la toma. Sirve para *diseñar*: “adivina dónde está el tercero desde el final, y comprueba”.
2. **Explorar todos los caminos a la vez.** La máquina se copia en cada elección. Una copia sin flecha desaparece. Importa *en qué estados* hay alguna copia.

Tres maneras de pensar el no determinismo

1. **Adivinar bien.** Si alguna elección lleva a aceptar, la máquina la toma. Sirve para *diseñar*: “adivina dónde está el tercero desde el final, y comprueba”.
2. **Explorar todos los caminos a la vez.** La máquina se copia en cada elección. Una copia sin flecha desaparece. Importa *en qué estados* hay alguna copia.
3. **Un árbol de ejecuciones.** Un hijo por cada flecha posible. El DFA es el caso de una sola rama.

Tres maneras de pensar el no determinismo

1. **Adivinar bien.** Si alguna elección lleva a aceptar, la máquina la toma. Sirve para *diseñar*: “adivina dónde está el tercero desde el final, y comprueba”.
2. **Explorar todos los caminos a la vez.** La máquina se copia en cada elección. Una copia sin flecha desaparece. Importa *en qué estados* hay alguna copia.
3. **Un árbol de ejecuciones.** Un hijo por cada flecha posible. El DFA es el caso de una sola rama.

Las tres dicen lo mismo

Acepta si **algún** camino termina en un estado de aceptación al acabarse la cadena. Una apuesta equivocada no puede terminar en aceptación.

El árbol, rama por rama

terminal

```
$ build/paths_nfa machines/third_from_end.nfa 0110
w = 0110
  q0 -0-> q0 -1-> q0 -1-> q0 -0-> q0    termina en q0, que no está en F
  q0 -0-> q0 -1-> q0 -1-> q1 -0-> q2    termina en q2, que no está en F
  q0 -0-> q0 -1-> q1 -1-> q2 -0-> q3    termina en q3, que está en F
caminos que leen toda la cadena: 3    que mueren antes: 0    que terminan en F: 1
algún camino termina en F: acepta
```

- Tres caminos: no apostar nunca, apostar en el segundo `1`, apostar en el primero.

El árbol, rama por rama

terminal

```
$ build/paths_nfa machines/third_from_end.nfa 0110
w = 0110
  q0 -0-> q0 -1-> q0 -1-> q0 -0-> q0    termina en q0, que no está en F
  q0 -0-> q0 -1-> q0 -1-> q1 -0-> q2    termina en q2, que no está en F
  q0 -0-> q0 -1-> q1 -1-> q2 -0-> q3    termina en q3, que está en F
caminos que leen toda la cadena: 3    que mueren antes: 0    que terminan en F: 1
algún camino termina en F: acepta
```

- Tres caminos: no apostar nunca, apostar en el segundo 1, apostar en el primero.
- El segundo apostó tarde: la cadena se acabó antes de llegar a q_3.

El árbol, rama por rama

terminal

```
$ build/paths_nfa machines/third_from_end.nfa 0110
w = 0110
  q0 -0-> q0 -1-> q0 -1-> q0 -0-> q0    termina en q0, que no está en F
  q0 -0-> q0 -1-> q0 -1-> q1 -0-> q2    termina en q2, que no está en F
  q0 -0-> q0 -1-> q1 -1-> q2 -0-> q3    termina en q3, que está en F
caminos que leen toda la cadena: 3    que mueren antes: 0    que terminan en F: 1
algún camino termina en F: acepta
```

- Tres caminos: no apostar nunca, apostar en el segundo `1`, apostar en el primero.
- El segundo apostó tarde: la cadena se acabó antes de llegar a q_3.
- Uno acepta, y con uno basta: `0110` $\in T$.

Apostar demasiado pronto

terminal

```
$ build/paths_nfa machines/third_from_end.nfa 1100
w = 1100
  q0 -1-> q0 -1-> q0 -0-> q0 -0-> q0    termina en q0, que no está en F
  q0 -1-> q0 -1-> q1 -0-> q2 -0-> q3    termina en q3, que está en F
  q0 -1-> q1 -1-> q2 -0-> q3    muere: q3 no tiene transición con 0
caminos que leen toda la cadena: 2    que mueren antes: 1    que terminan en F: 1
algún camino termina en F: acepta
```

- El tercer camino llegó a q_3 con un símbolo por leer, y murió.

Apostar demasiado pronto

terminal

```
$ build/paths_nfa machines/third_from_end.nfa 1100
w = 1100
  q0 -1-> q0 -1-> q0 -0-> q0 -0-> q0    termina en q0, que no está en F
  q0 -1-> q0 -1-> q1 -0-> q2 -0-> q3    termina en q3, que está en F
  q0 -1-> q1 -1-> q2 -0-> q3    muere: q3 no tiene transición con 0
caminos que leen toda la cadena: 2    que mueren antes: 1    que terminan en F: 1
algún camino termina en F: acepta
```

- El tercer camino llegó a q_3 con un símbolo por leer, y murió.
- Pasó por un estado de aceptación y **no cuenta**: solo importa dónde se está al acabarse la cadena.

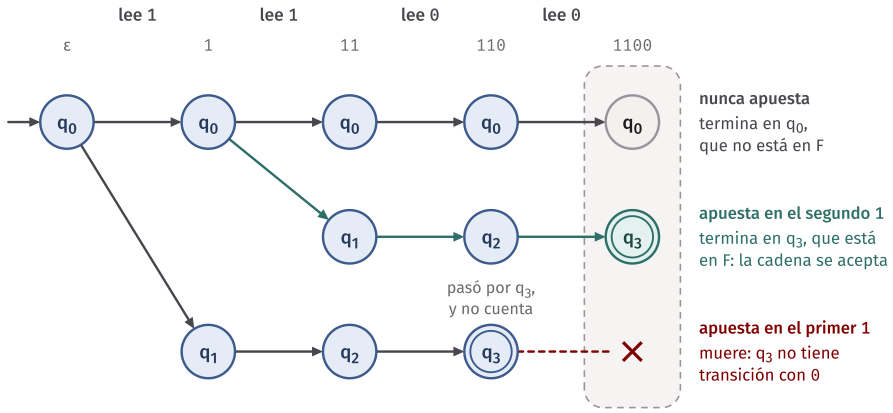
Apostar demasiado pronto

terminal

```
$ build/paths_nfa machines/third_from_end.nfa 1100
w = 1100
  q0 -1-> q0 -1-> q0 -0-> q0 -0-> q0    termina en q0, que no está en F
  q0 -1-> q0 -1-> q1 -0-> q2 -0-> q3    termina en q3, que está en F
  q0 -1-> q1 -1-> q2 -0-> q3    muere: q3 no tiene transición con 0
caminos que leen toda la cadena: 2    que mueren antes: 1    que terminan en F: 1
algún camino termina en F: acepta
```

- El tercer camino llegó a q_3 con un símbolo por leer, y murió.
- Pasó por un estado de aceptación y **no cuenta**: solo importa dónde se está al acabarse la cadena.
- Por eso q_3 no tiene flechas de salida: obliga a que la apuesta sea exacta.

El árbol de ejecuciones de `1100`



Tres errores sobre aceptar

Errores comunes

“Acepta si *todos* los caminos aceptan.” Es al revés: basta uno. Con `0110`, dos de los tres caminos no aceptan, y la cadena se acepta.

Tres errores sobre aceptar

Errores comunes

“Acepta si *todos* los caminos aceptan.” Es al revés: basta uno. Con `0110`, dos de los tres caminos no aceptan, y la cadena se acepta.

“Elige al azar: a veces acepta y a veces no.” No hay probabilidades. Que w esté en el lenguaje depende de si *existe* un camino.

Tres errores sobre aceptar

Errores comunes

“Acepta si *todos* los caminos aceptan.” Es al revés: basta uno. Con `0110`, dos de los tres caminos no aceptan, y la cadena se acepta.

“Elige al azar: a veces acepta y a veces no.” No hay probabilidades. Que w esté en el lenguaje depende de si *existe* un camino.

“Un camino que muere rechaza la cadena.” Un camino que muere solo deja de contar. Se rechaza cuando no queda *ninguno* que acepte.

La definición formal

Autómata finito no determinista, NFA ***(nondeterministic finite automaton)***

Una quíntupla $N = (Q, \Sigma, \delta, q_0, F)$, donde:

1. Q es un conjunto finito, el de los **estados**;
2. Σ es un alfabeto;
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ es la **función de transición**;
4. $q_0 \in Q$ es el **estado inicial**;
5. $F \subseteq Q$ es el conjunto de **estados de aceptación**.

La definición formal

Autómata finito no determinista, NFA ***(nondeterministic finite automaton)***

Una quíntupla $N = (Q, \Sigma, \delta, q_0, F)$, donde:

1. Q es un conjunto finito, el de los **estados**;
2. Σ es un alfabeto;
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ es la **función de transición**;
4. $q_0 \in Q$ es el **estado inicial**;
5. $F \subseteq Q$ es el conjunto de **estados de aceptación**.

- Devuelve un **conjunto**: $\delta(q_0, 1) = \{q_0, q_1\}$. Puede ser vacío: $\delta(q_3, 0) = \emptyset$.

La definición formal

Autómata finito no determinista, NFA ***(nondeterministic finite automaton)***

Una quíntupla $N = (Q, \Sigma, \delta, q_0, F)$, donde:

1. Q es un conjunto finito, el de los **estados**;
2. Σ es un alfabeto;
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ es la **función de transición**;
4. $q_0 \in Q$ es el **estado inicial**;
5. $F \subseteq Q$ es el conjunto de **estados de aceptación**.

- Devuelve un **conjunto**: $\delta(q_0, 1) = \{q_0, q_1\}$. Puede ser vacío: $\delta(q_3, 0) = \emptyset$.
- Recibe un símbolo **o** ε: $\delta(q, \varepsilon)$ son los estados a los que se pasa sin leer.

La definición formal

Autómata finito no determinista, NFA ***(nondeterministic finite automaton)***

Una quíntupla $N = (Q, \Sigma, \delta, q_0, F)$, donde:

1. Q es un conjunto finito, el de los **estados**;
2. Σ es un alfabeto;
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ es la **función de transición**;
4. $q_0 \in Q$ es el **estado inicial**;
5. $F \subseteq Q$ es el conjunto de **estados de aceptación**.

- Devuelve un **conjunto**: $\delta(q_0, 1) = \{q_0, q_1\}$. Puede ser vacío: $\delta(q_3, 0) = \emptyset$.
- Recibe un símbolo **o** ε: $\delta(q, \varepsilon)$ son los estados a los que se pasa sin leer.
- ε no es un símbolo del alfabeto, y nunca aparece en la entrada.

Caminos y aceptación

Camino de p a r con etiqueta w

Estados $r_0, r_1, \ldots, r_m$ y etiquetas $y_1, \ldots, y_m$ en $\Sigma \cup \{\varepsilon\}$, con $m \geq 0$, tales que:

1. $r_0 = p$ y $r_m = r$;
2. $r_i \in \delta(r_{i-1}, y_i)$ para cada i: cada paso sigue una flecha;
3. $y_1 y_2 \cdots y_m = w$: las etiquetas, concatenadas, forman w.

Caminos y aceptación

Camino de p a r con etiqueta w

Estados $r_0, r_1, \ldots, r_m$ y etiquetas $y_1, \ldots, y_m$ en $\Sigma \cup \{\varepsilon\}$, con $m \geq 0$, tales que:

1. $r_0 = p$ y $r_m = r$;
2. $r_i \in \delta(r_{i-1}, y_i)$ para cada i: cada paso sigue una flecha;
3. $y_1 y_2 \cdots y_m = w$: las etiquetas, concatenadas, forman w.

- Una etiqueta ε no aporta nada: el camino puede tener más pasos que $|w|$.

Caminos y aceptación

Camino de p a r con etiqueta w

Estados $r_0, r_1, \ldots, r_m$ y etiquetas $y_1, \ldots, y_m$ en $\Sigma \cup \{\varepsilon\}$, con $m \geq 0$, tales que:

1. $r_0 = p$ y $r_m = r$;
2. $r_i \in \delta(r_{i-1}, y_i)$ para cada i: cada paso sigue una flecha;
3. $y_1 y_2 \cdots y_m = w$: las etiquetas, concatenadas, forman w.

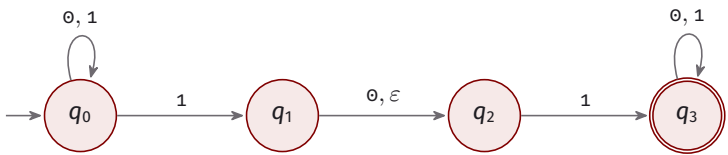
- Una etiqueta ε no aporta nada: el camino puede tener más pasos que $|w|$.
- N **acepta** w si hay un camino de q_0 a algún estado de F con etiqueta w.
- $L(N)$ es el conjunto de las cadenas que N acepta.

SECCIÓN 02

Transiciones ε

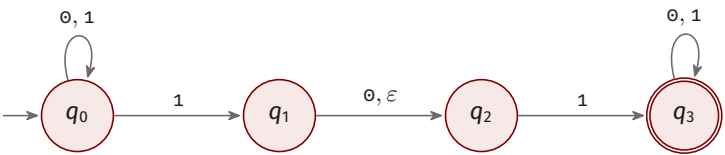
02

Transiciones ε: contiene 101 o 11



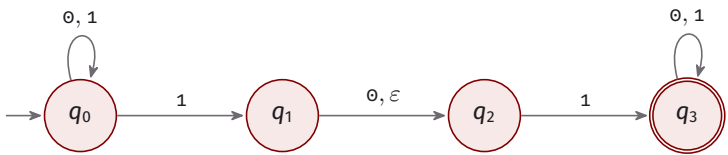
- La máquina **adivina dónde empieza** el patrón y lo comprueba símbolo por símbolo.

Transiciones ε: contiene 101 o 11



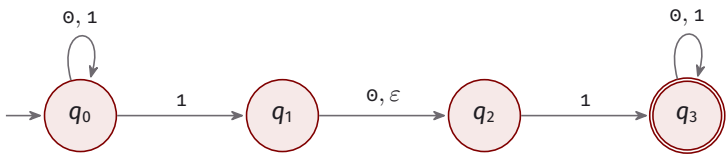
- La máquina **adivina dónde empieza** el patrón y lo comprueba símbolo por símbolo.
- De q_1 a q_2: leyendo el `0` del medio de `101`, o **sin leer nada**, para `11`.

Transiciones ε: contiene 101 o 11



- La máquina **adivina dónde empieza** el patrón y lo comprueba símbolo por símbolo.
- De q_1 a q_2: leyendo el `0` del medio de `101`, o **sin leer nada**, para `11`.
- Si se equivocó, ese camino muere y no pasa nada. No hay que pensar "cuánto se conserva", como en el DFA de *Contiene 101*.

Transiciones ε: contiene 101 o 11



- La máquina **adivina dónde empieza** el patrón y lo comprueba símbolo por símbolo.
- De q_1 a q_2: leyendo el `0` del medio de `101`, o **sin leer nada**, para `11`.
- Si se equivocó, ese camino muere y no pasa nada. No hay que pensar "cuánto se conserva", como en el DFA de *Contiene 101*.
- En el archivo: `q1 eps q2`.

Los caminos con una transición ε

terminal

```
$ build/paths_nfa machines/contains_101_or_11.nfa 0101
w = 0101
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q0    termina en q0, que no está en F
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q1    termina en q1, que no está en F
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q1 -ε-> q2    termina en q2, que no está en F
  q0 -0-> q0 -1-> q1 -0-> q2 -1-> q3    termina en q3, que está en F
  q0 -0-> q0 -1-> q1 -ε-> q2    muere: q2 no tiene transición con 0
caminos que leen toda la cadena: 4    que mueren antes: 1    que terminan en F: 1
algún camino termina en F: acepta
```

- El cuarto apostó por el primer `1` y leyó `101`: acepta.

Los caminos con una transición ε

terminal

```
$ build/paths_nfa machines/contains_101_or_11.nfa 0101
w = 0101
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q0    termina en q0, que no está en F
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q1    termina en q1, que no está en F
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q1 -ε-> q2    termina en q2, que no está en F
  q0 -0-> q0 -1-> q1 -0-> q2 -1-> q3    termina en q3, que está en F
  q0 -0-> q0 -1-> q1 -ε-> q2    muere: q2 no tiene transición con 0
caminos que leen toda la cadena: 4    que mueren antes: 1    que terminan en F: 1
algún camino termina en F: acepta
```

- El cuarto apostó por el primer `1` y leyó `101`: acepta.
- El quinto tomó la transición ε esperando un `11`: llegó un `0` y murió.

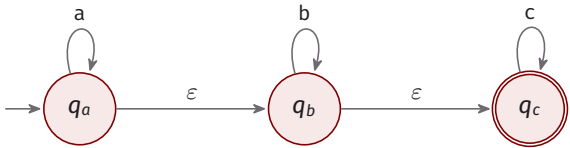
Los caminos con una transición ε

terminal

```
$ build/paths_nfa machines/contains_101_or_11.nfa 0101
w = 0101
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q0    termina en q0, que no está en F
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q1    termina en q1, que no está en F
  q0 -0-> q0 -1-> q0 -0-> q0 -1-> q1 -ε-> q2    termina en q2, que no está en F
  q0 -0-> q0 -1-> q1 -0-> q2 -1-> q3    termina en q3, que está en F
  q0 -0-> q0 -1-> q1 -ε-> q2    muere: q2 no tiene transición con 0
caminos que leen toda la cadena: 4    que mueren antes: 1    que terminan en F: 1
algún camino termina en F: acepta
```

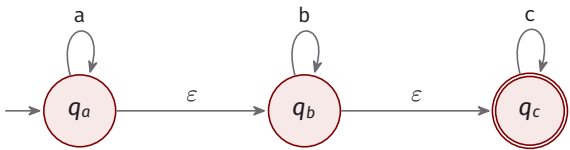
- El cuarto apostó por el primer `1` y leyó `101`: acepta.
- El quinto tomó la transición ε esperando un `11`: llegó un `0` y murió.
- Un paso ε no consume símbolos: el tercer camino tiene 5 pasos y lee 4 símbolos.

Transiciones ε en cadena



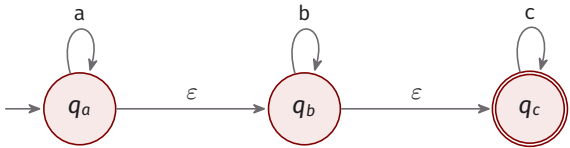
- Algunas a, después algunas b, después algunas c. Cada bloque puede ser vacío.

Transiciones ε en cadena



- Algunas a, después algunas b, después algunas c. Cada bloque puede ser vacío.
- La máquina adivina cuándo se acabó cada bloque.

Transiciones ε en cadena



- Algunas a, después algunas b, después algunas c. Cada bloque puede ser vacío.
- La máquina adivina cuándo se acabó cada bloque.
- Acepta ε aunque $q_a \notin F$: el camino $q_a \to q_b \to q_c$ tiene etiqueta ε.

La clausura ε

Clausura ε de un conjunto de estados S (ε-*closure*)

$E(S)$ es el conjunto de los estados a los que llega, desde algún estado de S, un camino con etiqueta ε: cero, una o varias transiciones ε seguidas.

La clausura ε

Clausura ε de un conjunto de estados S (ε-closure)

$E(S)$ es el conjunto de los estados a los que llega, desde algún estado de S, un camino con etiqueta ε: cero, una o varias transiciones ε seguidas.

- $S \subseteq E(S)$: el camino de cero pasos.
- $E(E(S)) = E(S)$, y si $S \subseteq S'$ entonces $E(S) \subseteq E(S')$.

La clausura ε

Clausura ε de un conjunto de estados S (ε-closure)

$E(S)$ es el conjunto de los estados a los que llega, desde algún estado de S, un camino con etiqueta ε: cero, una o varias transiciones ε seguidas.

- $S \subseteq E(S)$: el camino de cero pasos.
- $E(E(S)) = E(S)$, y si $S \subseteq S'$ entonces $E(S) \subseteq E(S')$.
- En los bloques: $E(\{q_a\}) = \{q_a, q_b, q_c\}$, $E(\{q_b\}) = \{q_b, q_c\}$.
- En *Contiene 101 o 11*: $E(\{q_1\}) = \{q_1, q_2\}$.

La clausura ε

Clausura ε de un conjunto de estados S (ε-closure)

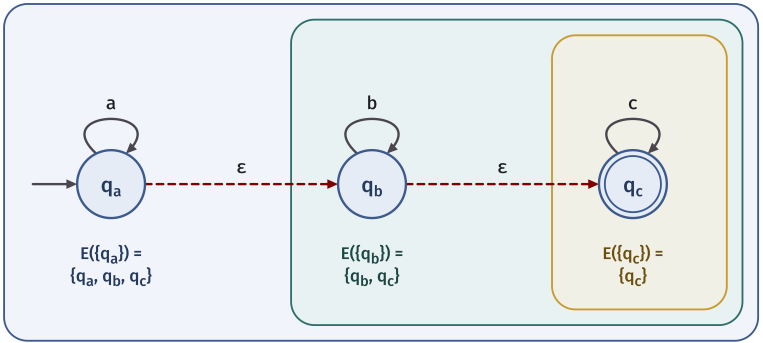
$E(S)$ es el conjunto de los estados a los que llega, desde algún estado de S, un camino con etiqueta ε: cero, una o varias transiciones ε seguidas.

- $S \subseteq E(S)$: el camino de cero pasos.
- $E(E(S)) = E(S)$, y si $S \subseteq S'$ entonces $E(S) \subseteq E(S')$.
- En los bloques: $E(\{q_a\}) = \{q_a, q_b, q_c\}$, $E(\{q_b\}) = \{q_b, q_c\}$.
- En *Contiene 101 o 11*: $E(\{q_1\}) = \{q_1, q_2\}$.

Error común

Seguir una sola transición ε y detenerse. $q_c \in E(\{q_a\})$ aunque no haya flecha directa.

Las clausuras de los bloques, una dentro de otra



Desde q_a: cero pasos ε dan q_a, un paso da q_b y dos pasos dan q_c.

q_c está en E({q_a}) aunque no hay flecha directa de q_a a q_c: la clausura sigue todas las cadenas de flechas ε.

Calcular $E(S)$: un recorrido de un grafo

nfa.h: eps_closure(n, s)

```
1   StateSet closure = s;
2   std::queue<std::string> pending;  // con los de s
3   while (!pending.empty()) {
4       std::string q = pending.front();
5       pending.pop();
6       for (const std::string& r : delta_of(n, q, EPS)) {
7           if (closure.insert(r).second) {
8               pending.push(r);
9           }
10      }
11  }
12  return closure;
```

- Se dejan solo las flechas ε: queda un grafo dirigido.
- $E(S)$ son los vértices alcanzables desde S.

Calcular $E(S)$: un recorrido de un grafo

nfa.h: eps_closure(n, s)

```
1   StateSet closure = s;
2   std::queue<std::string> pending;  // con los de s
3   while (!pending.empty()) {
4       std::string q = pending.front();
5       pending.pop();
6       for (const std::string& r : delta_of(n, q, EPS)) {
7           if (closure.insert(r).second) {
8               pending.push(r);
9           }
10      }
11  }
12  return closure;
```

- Se dejan solo las flechas ε: queda un grafo dirigido.
- $E(S)$ son los vértices alcanzables desde S.
- Recorrido en anchura, como en Algoritmos.

Calcular $E(S)$: un recorrido de un grafo

nfa.h: eps_closure(n, s)

```
1  StateSet closure = s;
2  std::queue<std::string> pending;  // con los de s
3  while (!pending.empty()) {
4      std::string q = pending.front();
5      pending.pop();
6      for (const std::string& r : delta_of(n, q, EPS)) {
7          if (closure.insert(r).second) {
8              pending.push(r);
9          }
10     }
11 }
12 return closure;
```

- Se dejan solo las flechas ε: queda un grafo dirigido.
- $E(S)$ son los vértices alcanzables desde S.
- Recorrido en anchura, como en Algoritmos.
- Cada estado entra una vez a la cola: termina aunque haya ciclos de ε.

SECCIÓN 03

El conjunto de estados activos

03

Seguir un NFA: los estados activos

- Listar caminos no sirve para calcular: pueden ser una cantidad exponencial.
- No interesa cuántas copias hay, sino **en qué estados hay alguna**: los **estados activos**.

Seguir un NFA: los estados activos

- Listar caminos no sirve para calcular: pueden ser una cantidad exponencial.
- No interesa cuántas copias hay, sino **en qué estados hay alguna**: los **estados activos**.

Función de transición extendida de un NFA, sobre conjuntos

$$\delta^*(S, \varepsilon) = E(S)$$
$$\delta^*(S, xa) = E\Big(\bigcup_{q \in \delta^*(S,x)} \delta(q, a) \Big)$$

Seguir un NFA: los estados activos

- Listar caminos no sirve para calcular: pueden ser una cantidad exponencial.
- No interesa cuántas copias hay, sino **en qué estados hay alguna**: los **estados activos**.

Función de transición extendida de un NFA, sobre conjuntos

$$\delta^*(S, \varepsilon) = E(S)$$
$$\delta^*(S, xa) = E\Big(\bigcup_{q \in \delta^*(S,x)} \delta(q, a) \Big)$$

- Desde cada estado activo se siguen las flechas con a, se juntan los destinos y se cierra con ε.

Seguir un NFA: los estados activos

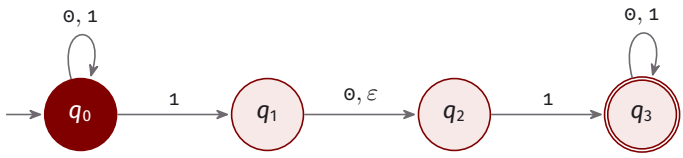
- Listar caminos no sirve para calcular: pueden ser una cantidad exponencial.
- No interesa cuántas copias hay, sino **en qué estados hay alguna**: los **estados activos**.

Función de transición extendida de un NFA, sobre conjuntos

$$\delta^*(S, \varepsilon) = E(S)$$
$$\delta^*(S, xa) = E\Big(\bigcup_{q \in \delta^*(S,x)} \delta(q, a) \Big)$$

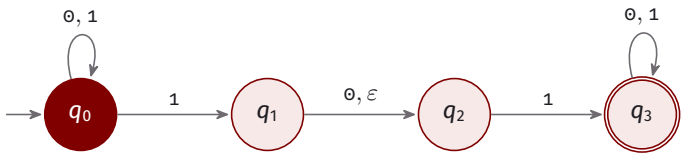
- Desde cada estado activo se siguen las flechas con a, se juntan los destinos y se cierra con ε.
- Estados activos después de leer w: $\delta^*(\{q_0\}, w)$.

A mano: `01011` en *Contiene 101 o 11*



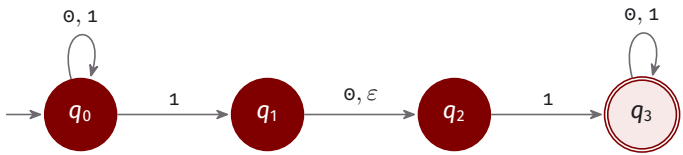
Leído	Símbolo	Destinos de las flechas	Estados activos
ε			$E(\{q_0\}) = \{q_0\}$

A mano: `01011` en *Contiene 101 o 11*



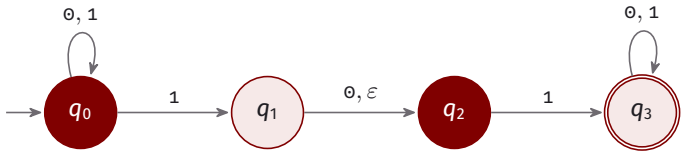
Leído	Símbolo	Destinos de las flechas	Estados activos
ε			$E(\{q_0\}) = \{q_0\}$
0	0	$\{q_0\}$	$\{q_0\}$

A mano: `01011` en *Contiene 101 o 11*



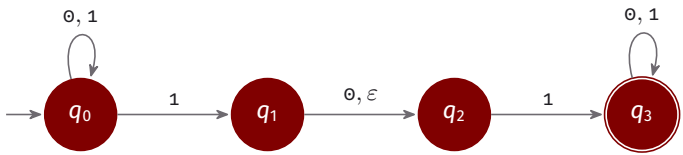
Leído	Símbolo	Destinos de las flechas	Estados activos
ε			$E(\{q_0\}) = \{q_0\}$
`0`	`0`	$\{q_0\}$	$\{q_0\}$
`01`	`1`	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$

A mano: `01011` en *Contiene 101 o 11*



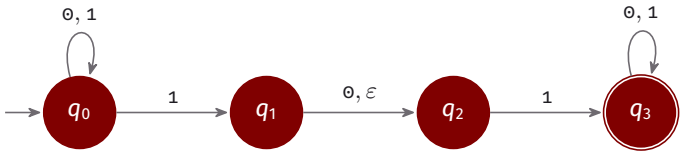
Leído	Símbolo	Destinos de las flechas	Estados activos
ε			$E(\{q_0\}) = \{q_0\}$
`0`	`0`	$\{q_0\}$	$\{q_0\}$
`01`	`1`	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
`010`	`0`	$\{q_0\} \cup \{q_2\} \cup \emptyset$	$\{q_0, q_2\}$

A mano: `01011` en *Contiene 101 o 11*



Leído	Símbolo	Destinos de las flechas	Estados activos
ε			$E(\{q_0\}) = \{q_0\}$
`0`	`0`	$\{q_0\}$	$\{q_0\}$
`01`	`1`	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
`010`	`0`	$\{q_0\} \cup \{q_2\} \cup \emptyset$	$\{q_0, q_2\}$
`0101`	`1`	$\{q_0, q_1\} \cup \{q_3\}$	$\{q_0, q_1, q_2, q_3\}$

A mano: `01011` en *Contiene 101 o 11*



Leído	Símbolo	Destinos de las flechas	Estados activos
ε			$E(\{q_0\}) = \{q_0\}$
`0`	`0`	$\{q_0\}$	$\{q_0\}$
`01`	`1`	$\{q_0, q_1\}$	$\{q_0, q_1, q_2\}$
`010`	`0`	$\{q_0\} \cup \{q_2\} \cup \emptyset$	$\{q_0, q_2\}$
`0101`	`1`	$\{q_0, q_1\} \cup \{q_3\}$	$\{q_0, q_1, q_2, q_3\}$
`01011`	`1`	$\{q_0, q_1\} \cup \emptyset \cup \{q_3\} \cup \{q_3\}$	$\{q_0, q_1, q_2, q_3\}$

Lo mismo, con un programa

terminal

```
$ build/run_nfa machines/contains_101_or_11.nfa 01011 0100
w = 01011
  {q0} -0-> {q0} -1-> {q0,q1,q2} -0-> {q0,q2} -1-> {q0,q1,q2,q3} -1-> {q0,q1,q2,q3}
  termina con {q0,q1,q2,q3} activos, y q3 está en F: acepta
w = 0100
  {q0} -0-> {q0} -1-> {q0,q1,q2} -0-> {q0,q2} -0-> {q0}
  termina con {q0} activos, y ninguno está en F: rechaza
```

Lema de los estados activos

$\delta^*(S, w)$ es exactamente el conjunto de los estados a los que llega un camino con etiqueta w desde algún estado de S.

Lo mismo, con un programa

terminal

```
$ build/run_nfa machines/contains_101_or_11.nfa 01011 0100
w = 01011
  {q0} -0-> {q0} -1-> {q0,q1,q2} -0-> {q0,q2} -1-> {q0,q1,q2,q3} -1-> {q0,q1,q2,q3}
  termina con {q0,q1,q2,q3} activos, y q3 está en F: acepta
w = 0100
  {q0} -0-> {q0} -1-> {q0,q1,q2} -0-> {q0,q2} -0-> {q0}
  termina con {q0} activos, y ninguno está en F: rechaza
```

Lema de los estados activos

$\delta^*(S, w)$ es exactamente el conjunto de los estados a los que llega un camino con etiqueta w desde algún estado de S.

Corolario. N acepta w si y solo si $\delta^*(\{q_0\}, w) \cap F \neq \emptyset$. La demostración, en la sesión B.

Cuánto cuesta simular un NFA

- Con n estados, el conjunto de estados activos tiene a lo sumo n elementos.

Cuánto cuesta simular un NFA

- Con n estados, el conjunto de estados activos tiene a lo sumo n elementos.
- Por símbolo: recorrer los activos y juntar sus destinos. Del orden de n^2 operaciones.

Cuánto cuesta simular un NFA

- Con n estados, el conjunto de estados activos tiene a lo sumo n elementos.
- Por símbolo: recorrer los activos y juntar sus destinos. Del orden de n^2 operaciones.
- La cadena completa: del orden de $|w| \cdot n^2$. No es exponencial, aunque los caminos lo sean.

Cuánto cuesta simular un NFA

- Con n estados, el conjunto de estados activos tiene a lo sumo n elementos.
- Por símbolo: recorrer los activos y juntar sus destinos. Del orden de n^2 operaciones.
- La cadena completa: del orden de $|w| \cdot n^2$. No es exponencial, aunque los caminos lo sean.

En la práctica

Dos familias de buscadores de patrones. Con *backtracking*: un camino a la vez, y retroceder; con ciertos patrones tarda un tiempo exponencial. Con el conjunto de estados activos: tiempo proporcional al texto; es el enfoque de `grep` y de RE2.

Todo DFA es un NFA

- Un DFA es un NFA en el que nunca hay que elegir: $\delta'(q, a) = \{\delta(q, a)\}$ y $\delta'(q, \varepsilon) = \emptyset$.
- Todo archivo `.dfa` es un archivo `.nfa` válido.

Todo DFA es un NFA

- Un DFA es un NFA en el que nunca hay que elegir: $\delta'(q, a) = \{\delta(q, a)\}$ y $\delta'(q, \varepsilon) = \emptyset$.
- Todo archivo `.dfa` es un archivo `.nfa` válido.
- Entonces todo lenguaje regular es reconocido por algún NFA.

Todo DFA es un NFA

- Un DFA es un NFA en el que nunca hay que elegir: $\delta'(q, a) = \{\delta(q, a)\}$ y $\delta'(q, \varepsilon) = \emptyset$.
- Todo archivo `.dfa` es un archivo `.nfa` válido.
- Entonces todo lenguaje regular es reconocido por algún NFA.

La pregunta de la sesión B

Los NFA adivinan, se saltan símbolos, resuelven en 4 estados lo que al DFA le cuesta 8. ¿Hay algún lenguaje que un NFA reconozca y que **ningún** DFA pueda reconocer?

Todo DFA es un NFA

- Un DFA es un NFA en el que nunca hay que elegir: $\delta'(q, a) = \{\delta(q, a)\}$ y $\delta'(q, \varepsilon) = \emptyset$.
- Todo archivo `.dfa` es un archivo `.nfa` válido.
- Entonces todo lenguaje regular es reconocido por algún NFA.

La pregunta de la sesión B

Los NFA adivinan, se saltan símbolos, resuelven en 4 estados lo que al DFA le cuesta 8. ¿Hay algún lenguaje que un NFA reconozca y que **ningún** DFA pueda reconocer?

Pista: para simular un NFA basta el conjunto de estados activos, y hay una cantidad **finita** de conjuntos.

SECCIÓN 04

La construcción de subconjuntos

04

La idea

- La salida de `run_nfa` es una secuencia de conjuntos, uno después de cada símbolo.
- El conjunto siguiente depende solo del conjunto actual y del símbolo leído.

La idea

- La salida de `run_nfa` es una secuencia de conjuntos, uno después de cada símbolo.
- El conjunto siguiente depende solo del conjunto actual y del símbolo leído.
- Eso es la ejecución de una máquina **determinista** cuyos estados son conjuntos.

La idea

- La salida de `run_nfa` es una secuencia de conjuntos, uno después de cada símbolo.
- El conjunto siguiente depende solo del conjunto actual y del símbolo leído.
- Eso es la ejecución de una máquina **determinista** cuyos estados son conjuntos.

Idea clave

El conjunto de estados activos de un NFA es **un solo estado** de un DFA. Un NFA con n estados tiene 2^n subconjuntos, y 2^n es finito.

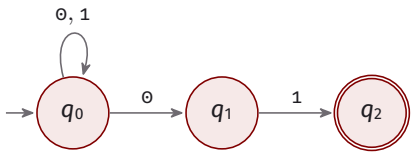
La idea

- La salida de `run_nfa` es una secuencia de conjuntos, uno después de cada símbolo.
- El conjunto siguiente depende solo del conjunto actual y del símbolo leído.
- Eso es la ejecución de una máquina **determinista** cuyos estados son conjuntos.

> **Idea clave**
>
> El conjunto de estados activos de un NFA es **un solo estado** de un DFA. Un NFA con n estados tiene 2^n subconjuntos, y 2^n es finito.

Es la jugada del producto: allí el estado nuevo era un *par* de estados; aquí es un *conjunto*.

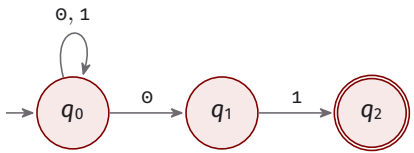
Un primer ejemplo: termina en 01



El NFA adivina dónde empieza el `01` final.

Estado inicial: $E(\{q_0\}) = \{q_0\}$.

Un primer ejemplo: termina en 01

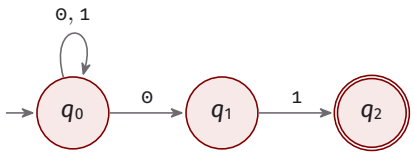


El NFA adivina dónde empieza el `01` final.

Estado inicial: $E(\{q_0\}) = \{q_0\}$.

- Desde $\{q_0\}$:
  con `0`, $\{q_0, q_1\}$ (nuevo); con `1`, $\{q_0\}$.

Un primer ejemplo: termina en 01

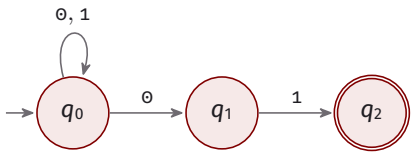


El NFA adivina dónde empieza el `01` final.

Estado inicial: $E(\{q_0\}) = \{q_0\}$.

- Desde $\{q_0\}$:
  con `0`, $\{q_0, q_1\}$ (nuevo); con `1`, $\{q_0\}$.
- Desde $\{q_0, q_1\}$:
  con `0`, $\{q_0, q_1\}$; con `1`, $\{q_0, q_2\}$ (nuevo).

Un primer ejemplo: termina en 01

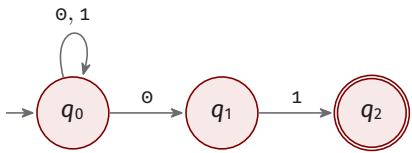


El NFA adivina dónde empieza el `01` final.

Estado inicial: $E(\{q_0\}) = \{q_0\}$.

- Desde $\{q_0\}$:
  con `0`, $\{q_0, q_1\}$ (nuevo); con `1`, $\{q_0\}$.
- Desde $\{q_0, q_1\}$:
  con `0`, $\{q_0, q_1\}$; con `1`, $\{q_0, q_2\}$ (nuevo).
- Desde $\{q_0, q_2\}$:
  con `0`, $\{q_0, q_1\}$; con `1`, $\{q_0\}$.

Un primer ejemplo: termina en 01



El NFA adivina dónde empieza el `01` final.

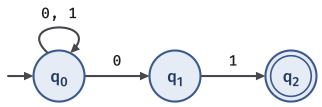
Estado inicial: $E(\{q_0\}) = \{q_0\}$.

- Desde $\{q_0\}$:
  con `0`, $\{q_0, q_1\}$ (nuevo); con `1`, $\{q_0\}$.
- Desde $\{q_0, q_1\}$:
  con `0`, $\{q_0, q_1\}$; con `1`, $\{q_0, q_2\}$ (nuevo).
- Desde $\{q_0, q_2\}$:
  con `0`, $\{q_0, q_1\}$; con `1`, $\{q_0\}$.

No apareció ninguno más: **3 estados**. De aceptación: $\{q_0, q_2\}$, el que contiene a q_2.

El DFA, estado por estado

EL NFA



LA TABLA

S	0	1
→ $\{q_0\}$		

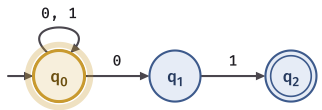
EL DFA, ESTADO POR ESTADO



El estado inicial del DFA es $E(\{q_0\}) = \{q_0\}$: los estados activos antes de leer nada.

El DFA, estado por estado

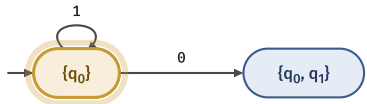
EL NFA



LA TABLA

	S	0	1
→	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_0, q_1\}$		

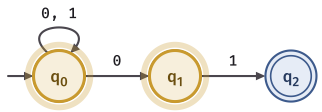
EL DFA, ESTADO POR ESTADO



Desde $\{q_0\}$: con 0 sale $\{q_0, q_1\}$, un conjunto nuevo; con 1, $\{q_0\}$, que ya estaba.

El DFA, estado por estado

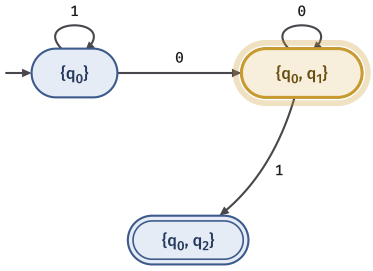
EL NFA



LA TABLA

	S	0	1
→	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
*	$\{q_0, q_2\}$		

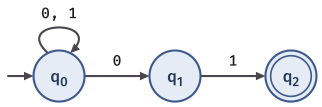
EL DFA, ESTADO POR ESTADO



Desde $\{q_0, q_1\}$: con 0, $\{q_0, q_1\} \cup \emptyset = \{q_0, q_1\}$; con 1, $\{q_0\} \cup \{q_2\} = \{q_0, q_2\}$, nuevo.

El DFA, estado por estado

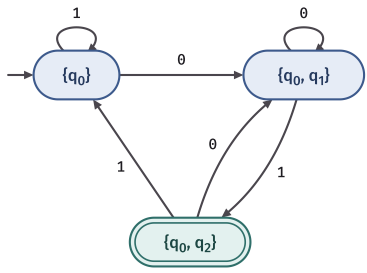
EL NFA



LA TABLA

	S	0	1
→	$\{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
	$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
*	$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

EL DFA, ESTADO POR ESTADO



Desde $\{q_0, q_2\}$: con 0, $\{q_0, q_1\}$; con 1, $\{q_0\}$. No aparece ningún conjunto nuevo.

El DFA usa 3 de los $2^3 = 8$ subconjuntos. Acepta $\{q_0, q_2\}$, el único que contiene a q_2.

El DFA que sale es el de la semana 1

terminal

```
$ build/subset machines/ends_01.nfa
E({q0}) = {q0}

     S         0         1
->   {q0}      {q0,q1}   {q0}
     {q0,q1}   {q0,q1}   {q0,q2}
   * {q0,q2}   {q0,q1}   {q0}

el NFA tiene 3 estados: 2^3 = 8 subconjuntos, 3 alcanzables
```

- Es `machines/ends_01.dfa`, con $\{q_0\}$, $\{q_0, q_1\}$, $\{q_0, q_2\}$ en lugar de q_0, q_1, q_2.
- “Termina en `0`” es “la copia que apostó por ese `0` sigue viva”.

La construcción de subconjuntos

Dado el NFA $N = (Q, \Sigma, \delta, q_0, F)$, el DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$

- $Q_D = \mathcal{P}(Q)$: un estado por cada subconjunto de Q.
- $q_D = E(\{q_0\})$: los estados activos antes de leer nada.
- $\delta_D(S, a) = E\Big(\bigcup_{q \in S} \delta(q, a)\Big)$.
- $F_D = \{\, S : S \cap F \neq \emptyset \,\}$: los que contienen **algún** estado de F.

La construcción de subconjuntos

Dado el NFA $N = (Q, \Sigma, \delta, q_0, F)$, el DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$

- $Q_D = \mathcal{P}(Q)$: un estado por cada subconjunto de Q.
- $q_D = E(\{q_0\})$: los estados activos antes de leer nada.
- $\delta_D(S, a) = E\Big(\bigcup_{q \in S} \delta(q, a)\Big)$.
- $F_D = \{\, S : S \cap F \neq \emptyset \,\}$: los que contienen **algún** estado de F.

- Es un DFA: Q_D es finito, con $2^{|Q|}$ elementos, y δ_D es total y da un solo subconjunto.

La construcción de subconjuntos

Dado el NFA $N = (Q, \Sigma, \delta, q_0, F)$, el DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$

- $Q_D = \mathcal{P}(Q)$: un estado por cada subconjunto de Q.
- $q_D = E(\{q_0\})$: los estados activos antes de leer nada.
- $\delta_D(S, a) = E\Big(\bigcup_{q \in S} \delta(q, a)\Big)$.
- $F_D = \{\, S : S \cap F \neq \emptyset \,\}$: los que contienen **algún** estado de F.

- Es un DFA: Q_D es finito, con $2^{|Q|}$ elementos, y δ_D es total y da un solo subconjunto.
- Cada pieza copia una pieza de la simulación con estados activos.

Dos detalles de la construcción

Error común: los estados de aceptación

Exigir que el subconjunto esté formado *solo* por estados de F. La condición es $S \cap F \neq \emptyset$: basta una copia en un estado de aceptación, porque el NFA acepta cuando *algún* camino acepta.

Dos detalles de la construcción

Error común: los estados de aceptación

Exigir que el subconjunto esté formado *solo* por estados de F. La condición es $S \cap F \neq \emptyset$: basta una copia en un estado de aceptación, porque el NFA acepta cuando *algún* camino acepta.

El conjunto vacío es un estado

$\delta_D(\emptyset, a) = E(\emptyset) = \emptyset$ para todo símbolo, y $\emptyset \cap F = \emptyset$.
De $\emptyset$ no se sale y no es de aceptación: es el **estado trampa**. Significa "ya murieron todas las copias".

Solo los subconjuntos alcanzables

pseudocódigo

```
algoritmo subconjuntos(N)
    inicial ← E({q0})
    estados ← {inicial}
    cola ← [inicial]
    mientras cola no esté vacía
        S ← sacar el primero de cola
        para cada símbolo a de Σ
            T ← E(unión de δ(q, a) para cada q en S)
            δD(S, a) ← T
            si T no está en estados entonces
                agregar T a estados y al final de cola
            fin si
        fin para
    fin mientras
    FD ← los S de estados con algún estado de F
    devolver (estados, Σ, δD, inicial, FD)
fin algoritmo
```

- La definición usa 2^n subconjuntos. La mayoría no aparece nunca.

Solo los subconjuntos alcanzables

pseudocódigo

```
algoritmo subconjuntos(N)
    inicial ← E({q0})
    estados ← {inicial}
    cola ← [inicial]
    mientras cola no esté vacía
        S ← sacar el primero de cola
        para cada símbolo a de Σ
            T ← E(unión de δ(q, a) para cada q en S)
            δD(S, a) ← T
            si T no está en estados entonces
                agregar T a estados y al final de cola
            fin si
        fin para
    fin mientras
    FD ← los S de estados con algún estado de F
    devolver (estados, Σ, δD, inicial, FD)
fin algoritmo
```

- La definición usa 2^n subconjuntos. La mayoría no aparece nunca.
- Los inalcanzables se pueden borrar sin cambiar el lenguaje (semana 1).

Solo los subconjuntos alcanzables

pseudocódigo

```
algoritmo subconjuntos(N)
    inicial ← E({q0})
    estados ← {inicial}
    cola ← [inicial]
    mientras cola no esté vacía
        S ← sacar el primero de cola
        para cada símbolo a de Σ
            T ← E(unión de δ(q, a) para cada q en S)
            δD(S, a) ← T
            si T no está en estados entonces
                agregar T a estados y al final de cola
            fin si
        fin para
    fin mientras
    FD ← los S de estados con algún estado de F
    devolver (estados, Σ, δD, inicial, FD)
fin algoritmo
```

- La definición usa 2^n subconjuntos. La mayoría no aparece nunca.
- Los inalcanzables se pueden borrar sin cambiar el lenguaje (semana 1).
- Recorrido en anchura de un grafo que se va calculando. Cada subconjunto entra a la cola una vez.

El tercer símbolo desde el final: 8 estados

terminal

```
$ build/subset machines/third_from_end.nfa
E({q0}) = {q0}

     S              0              1
->   {q0}           {q0}           {q0,q1}
     {q0,q1}        {q0,q2}        {q0,q1,q2}
     {q0,q2}        {q0,q3}        {q0,q1,q3}
     {q0,q1,q2}     {q0,q2,q3}     {q0,q1,q2,q3}
   * {q0,q3}        {q0}           {q0,q1}
   * {q0,q1,q3}     {q0,q2}        {q0,q1,q2}
   * {q0,q2,q3}     {q0,q3}        {q0,q1,q3}
   * {q0,q1,q2,q3}  {q0,q2,q3}     {q0,q1,q2,q3}

el NFA tiene 4 estados: 2^4 = 16 subconjuntos, 8 alcanzables
```

- Los 8 estados anunciados al principio.

El tercer símbolo desde el final: 8 estados

terminal

```
$ build/subset machines/third_from_end.nfa
E({q0}) = {q0}

     S              0              1
->   {q0}           {q0}           {q0,q1}
     {q0,q1}        {q0,q2}        {q0,q1,q2}
     {q0,q2}        {q0,q3}        {q0,q1,q3}
     {q0,q1,q2}     {q0,q2,q3}     {q0,q1,q2,q3}
   * {q0,q3}        {q0}           {q0,q1}
   * {q0,q1,q3}     {q0,q2}        {q0,q1,q2}
   * {q0,q2,q3}     {q0,q3}        {q0,q1,q3}
   * {q0,q1,q2,q3}  {q0,q2,q3}     {q0,q1,q2,q3}

el NFA tiene 4 estados: 2^4 = 16 subconjuntos, 8 alcanzables
```

- Los 8 estados anunciados al principio.
- Todos contienen a q_0: la copia que nunca apuesta.

El tercer símbolo desde el final: 8 estados

terminal

```
$ build/subset machines/third_from_end.nfa
E({q0}) = {q0}

     S              0              1
->   {q0}           {q0}           {q0,q1}
     {q0,q1}        {q0,q2}        {q0,q1,q2}
     {q0,q2}        {q0,q3}        {q0,q1,q3}
     {q0,q1,q2}     {q0,q2,q3}     {q0,q1,q2,q3}
   * {q0,q3}        {q0}           {q0,q1}
   * {q0,q1,q3}     {q0,q2}        {q0,q1,q2}
   * {q0,q2,q3}     {q0,q3}        {q0,q1,q3}
   * {q0,q1,q2,q3}  {q0,q2,q3}     {q0,q1,q2,q3}

el NFA tiene 4 estados: 2^4 = 16 subconjuntos, 8 alcanzables
```

- Los 8 estados anunciados al principio.
- Todos contienen a q_0: la copia que nunca apuesta.
- q_1, q_2, q_3 dicen cuáles de los tres últimos símbolos fueron `1`.
- $\{q_0, q_1, q_3\}$: los tres últimos fueron `101`.

El tercer símbolo desde el final: 8 estados

terminal

```
$ build/subset machines/third_from_end.nfa
E({q0}) = {q0}

     S              0              1
->   {q0}           {q0}           {q0,q1}
     {q0,q1}        {q0,q2}        {q0,q1,q2}
     {q0,q2}        {q0,q3}        {q0,q1,q3}
     {q0,q1,q2}     {q0,q2,q3}     {q0,q1,q2,q3}
   * {q0,q3}        {q0}           {q0,q1}
   * {q0,q1,q3}     {q0,q2}        {q0,q1,q2}
   * {q0,q2,q3}     {q0,q3}        {q0,q1,q3}
   * {q0,q1,q2,q3}  {q0,q2,q3}     {q0,q1,q2,q3}

el NFA tiene 4 estados: 2^4 = 16 subconjuntos, 8 alcanzables
```

- Los 8 estados anunciados al principio.
- Todos contienen a q_0: la copia que nunca apuesta.
- q_1, q_2, q_3 dicen cuáles de los tres últimos símbolos fueron `1`.
- $\{q_0, q_1, q_3\}$: los tres últimos fueron `101`.
- De aceptación: los que tienen a q_3.

El DFA tiene calculado lo que `run_nfa` calcula al leer

terminal

```
$ build/nfa_to_dfa machines/third_from_end.nfa | build/run_dfa - 0110 100
w = 0110
  {q0} -0-> {q0} -1-> {q0,q1} -1-> {q0,q1,q2} -0-> {q0,q2,q3}
  termina en {q0,q2,q3}, que está en F: acepta
w = 100
  {q0} -1-> {q0,q1} -0-> {q0,q2} -0-> {q0,q3}
  termina en {q0,q3}, que está en F: acepta
```

- `nfa_to_dfa` escribe un archivo `.dfa`: el nombre de cada estado es el conjunto.

El DFA tiene calculado lo que `run_nfa` calcula al leer

terminal

```
$ build/nfa_to_dfa machines/third_from_end.nfa | build/run_dfa - 0110 100
w = 0110
  {q0} -0-> {q0} -1-> {q0,q1} -1-> {q0,q1,q2} -0-> {q0,q2,q3}
  termina en {q0,q2,q3}, que está en F: acepta
w = 100
  {q0} -1-> {q0,q1} -0-> {q0,q2} -0-> {q0,q3}
  termina en {q0,q3}, que está en F: acepta
```

- `nfa_to_dfa` escribe un archivo `.dfa`: el nombre de cada estado es el conjunto.
- La ejecución del DFA es, conjunto por conjunto, la de `run_nfa`.

El DFA tiene calculado lo que `run_nfa` calcula al leer

terminal

```
$ build/nfa_to_dfa machines/third_from_end.nfa | build/run_dfa - 0110 100
w = 0110
  {q0} -0-> {q0} -1-> {q0,q1} -1-> {q0,q1,q2} -0-> {q0,q2,q3}
  termina en {q0,q2,q3}, que está en F: acepta
w = 100
  {q0} -1-> {q0,q1} -0-> {q0,q2} -0-> {q0,q3}
  termina en {q0,q3}, que está en F: acepta
```

- `nfa_to_dfa` escribe un archivo `.dfa`: el nombre de cada estado es el conjunto.
- La ejecución del DFA es, conjunto por conjunto, la de `run_nfa`.
- `run_nfa` calcula cada conjunto mientras lee. El DFA solo busca una casilla en su tabla.

Con transiciones ε

terminal

```
$ build/subset machines/contains_101_or_11.nfa
E({q0}) = {q0}

     S              0              1
->   {q0}           {q0}           {q0,q1,q2}
     {q0,q1,q2}     {q0,q2}        {q0,q1,q2,q3}
     {q0,q2}        {q0}           {q0,q1,q2,q3}
   * {q0,q1,q2,q3}  {q0,q2,q3}     {q0,q1,q2,q3}
   * {q0,q2,q3}     {q0,q3}        {q0,q1,q2,q3}
   * {q0,q3}        {q0,q3}        {q0,q1,q2,q3}
```

terminal

```
$ build/subset machines/abc_blocks.nfa
E({qa}) = {qa,qb,qc}

     S             a             b          c
-> * {qa,qb,qc}  {qa,qb,qc}  {qb,qc}     {qc}
   * {qb,qc}     {}           {qb,qc}     {qc}
   * {qc}        {}           {}          {qc}
     {}          {}           {}          {}
```

- La clausura aparece en dos lugares: en el estado inicial y al final de cada transición.

Con transiciones ε

terminal

```
$ build/subset machines/contains_101_or_11.nfa
E({q0}) = {q0}

     S              0              1
->   {q0}           {q0}           {q0,q1,q2}
     {q0,q1,q2}     {q0,q2}        {q0,q1,q2,q3}
     {q0,q2}        {q0}           {q0,q1,q2,q3}
   * {q0,q1,q2,q3}  {q0,q2,q3}     {q0,q1,q2,q3}
   * {q0,q2,q3}     {q0,q3}        {q0,q1,q2,q3}
   * {q0,q3}        {q0,q3}        {q0,q1,q2,q3}
```

terminal

```
$ build/subset machines/abc_blocks.nfa
E({qa}) = {qa,qb,qc}

     S             a             b          c
-> * {qa,qb,qc}  {qa,qb,qc}  {qb,qc}     {qc}
   * {qb,qc}     {}            {qb,qc}    {qc}
   * {qc}        {}            {}         {qc}
     {}          {}            {}         {}
```

- La clausura aparece en dos lugares: en el estado inicial y al final de cada transición.
- Izquierda: desde $\{q_0\}$ con 1, la clausura agrega q_2. La construcción no minimiza: las tres últimas filas son “ya ganó”.

Con transiciones ε

terminal

```
$ build/subset machines/contains_101_or_11.nfa
E({q0}) = {q0}

     S              0              1
->   {q0}           {q0}           {q0,q1,q2}
     {q0,q1,q2}     {q0,q2}        {q0,q1,q2,q3}
     {q0,q2}        {q0}           {q0,q1,q2,q3}
   * {q0,q1,q2,q3}  {q0,q2,q3}     {q0,q1,q2,q3}
   * {q0,q2,q3}     {q0,q3}        {q0,q1,q2,q3}
   * {q0,q3}        {q0,q3}        {q0,q1,q2,q3}
```

terminal

```
$ build/subset machines/abc_blocks.nfa
E({qa}) = {qa,qb,qc}

     S              a              b          c
-> * {qa,qb,qc}     {qa,qb,qc}     {qb,qc}    {qc}
   * {qb,qc}        {}             {qb,qc}    {qc}
   * {qc}           {}             {}         {qc}
     {}             {}             {}         {}
```

- La clausura aparece en dos lugares: en el estado inicial y al final de cada transición.
- Izquierda: desde $\{q_0\}$ con `1`, la clausura agrega q_2. La construcción no minimiza: las tres últimas filas son “ya ganó”.
- Derecha: el inicial $E(\{q_a\})$ ya es de aceptación, y `{}` es el estado trampa.

El error más frecuente del laboratorio

Error común

Olvidar la clausura ε del estado inicial y empezar en $\{q_0\}$ en lugar de $E(\{q_0\})$.

- En el autómata de los bloques, el DFA empezaría en $\{q_a\}$, que no es de aceptación.

El error más frecuente del laboratorio

Error común

Olvidar la clausura ε del estado inicial y empezar en $\{q_0\}$ en lugar de $E(\{q_0\})$.

- En el autómata de los bloques, el DFA empezaría en $\{q_a\}$, que no es de aceptación.
- Rechazaría la cadena vacía.
- Y también `b`: desde $\{q_a\}$, las flechas con `b` no llevan a ningún lado.

El error más frecuente del laboratorio

Error común

Olvidar la clausura ε del estado inicial y empezar en $\{q_0\}$ en lugar de $E(\{q_0\})$.

- En el autómata de los bloques, el DFA empezaría en $\{q_a\}$, que no es de aceptación.
- Rechazaría la cadena vacía.
- Y también `b`: desde $\{q_a\}$, las flechas con `b` no llevan a ningún lado.
- El verificador del laboratorio lo detecta y muestra la cadena.

SECCIÓN 05

Por qué funciona, y cuánto cuesta

05

El teorema

Teorema

Para todo NFA N hay un DFA D con $L(D) = L(N)$.

Demostración, por construcción. D es el DFA de la construcción de subconjuntos. Dos lemas, los dos por inducción sobre $|w|$:

El teorema

Teorema

Para todo NFA N hay un DFA D con $L(D) = L(N)$.

Demostración, por construcción. D es el DFA de la construcción de subconjuntos. Dos lemas, los dos por inducción sobre $|w|$:

1. **Estados activos.** $\delta^*(S, w)$ es el conjunto de los estados con un camino de etiqueta w desde S.
   La función sobre conjuntos calcula lo que pide la definición por caminos.

El teorema

Teorema

Para todo NFA N hay un DFA D con $L(D) = L(N)$.

Demostración, por construcción. D es el DFA de la construcción de subconjuntos. Dos lemas, los dos por inducción sobre $|w|$:

1. **Estados activos.** $\delta^*(S, w)$ es el conjunto de los estados con un camino de etiqueta w desde S.
   La función sobre conjuntos calcula lo que pide la definición por caminos.
2. **Construcción.** $\delta_D^*(q_D, w) = \delta^*(\{q_0\}, w)$.
   El estado en que queda D es el conjunto de estados activos de N.

Lema de los estados activos: el paso inductivo

$w = xa$, y $A = \delta^*(S, x)$. Hay que ver que $E\left(\bigcup_{q \in A} \delta(q, a)\right)$ son los estados con un camino de etiqueta xa.

Si r está en el conjunto, tiene un camino.

Caso base: $\delta^*(S, \varepsilon) = E(S)$, que es la definición de E.

Lema de los estados activos: el paso inductivo

$w = xa$, y $A = \delta^*(S, x)$. Hay que ver que $E\big(\bigcup_{q \in A} \delta(q, a)\big)$ son los estados con un camino de etiqueta xa.

Si r está en el conjunto, tiene un camino.

- Hay $q \in A$, un $r' \in \delta(q, a)$ y un camino ε de r' a r.
- Por hipótesis hay un camino de S a q con etiqueta x. Pegando: etiqueta $x \cdot a \cdot \varepsilon = xa$.

Caso base: $\delta^*(S, \varepsilon) = E(S)$, que es la definición de E.

Lema de los estados activos: el paso inductivo

$w = xa$, y $A = \delta^*(S, x)$. Hay que ver que $E\left(\bigcup_{q \in A} \delta(q, a)\right)$ son los estados con un camino de etiqueta xa.

Si r está en el conjunto, tiene un camino.

- Hay $q \in A$, un $r' \in \delta(q, a)$ y un camino ε de r' a r.
- Por hipótesis hay un camino de S a q con etiqueta x. Pegando: etiqueta $x \cdot a \cdot \varepsilon = xa$.

Si r tiene un camino con etiqueta xa, está en el conjunto.

- Sea y_j la **última** etiqueta que no es ε. Entonces $y_j = a$, lo anterior forma x y lo posterior es todo ε.

Caso base: $\delta^*(S, \varepsilon) = E(S)$, que es la definición de E.

Lema de los estados activos: el paso inductivo

$w = xa$, y $A = \delta^*(S, x)$. Hay que ver que $E\left(\bigcup_{q \in A} \delta(q, a)\right)$ son los estados con un camino de etiqueta xa.

Si r está en el conjunto, tiene un camino.

- Hay $q \in A$, un $r' \in \delta(q, a)$ y un camino ε de r' a r.
- Por hipótesis hay un camino de S a q con etiqueta x. Pegando: etiqueta $x \cdot a \cdot \varepsilon = xa$.

Si r tiene un camino con etiqueta xa, está en el conjunto.

- Sea y_j la **última** etiqueta que no es ε. Entonces $y_j = a$, lo anterior forma x y lo posterior es todo ε.
- Hasta r_{j-1}: etiqueta x, así que $r_{j-1} \in A$ por hipótesis.
- $r_j \in \delta(r_{j-1}, a)$, y de r_j a r solo hay pasos ε: r está en la clausura. ■

Caso base: $\delta^*(S, \varepsilon) = E(S)$, que es la definición de E.

Lema de la construcción

Afirmación. Para toda cadena w: $\delta_D^*(q_D, w) = \delta^*(\{q_0\}, w)$.

Caso base. $\delta_D^*(q_D, \varepsilon) = q_D = E(\{q_0\}) = \delta^*(\{q_0\}, \varepsilon)$.

Lema de la construcción

Afirmación. Para toda cadena w: $\delta_D^*(q_D, w) = \delta^*(\{q_0\}, w)$.

Caso base. $\delta_D^*(q_D, \varepsilon) = q_D = E(\{q_0\}) = \delta^*(\{q_0\}, \varepsilon)$.

Paso inductivo. $w = xa$, y el lema vale para x.

$$\delta_D^*(q_D, xa) = \delta_D(\delta_D^*(q_D, x), a) \qquad \text{definición de } \delta_D^*$$

Lema de la construcción

Afirmación. Para toda cadena w: $\delta_D^*(q_D, w) = \delta^*(\{q_0\}, w)$.

Caso base. $\delta_D^*(q_D, \varepsilon) = q_D = E(\{q_0\}) = \delta^*(\{q_0\}, \varepsilon)$.

Paso inductivo. $w = xa$, y el lema vale para x.

$$\begin{aligned} \delta_D^*(q_D, xa) &= \delta_D(\delta_D^*(q_D, x), a) && \text{definición de } \delta_D^* \\ &= \delta_D(\delta^*(\{q_0\}, x), a) && \text{hipótesis de inducción} \end{aligned}$$

Lema de la construcción

Afirmación. Para toda cadena w: $\delta_D^*(q_D, w) = \delta^*(\{q_0\}, w)$.

Caso base. $\delta_D^*(q_D, \varepsilon) = q_D = E(\{q_0\}) = \delta^*(\{q_0\}, \varepsilon)$.

Paso inductivo. $w = xa$, y el lema vale para x.

$$
\begin{aligned}
\delta_D^*(q_D, xa) &= \delta_D(\delta_D^*(q_D, x), a) && \text{definición de } \delta_D^* \\
&= \delta_D(\delta^*(\{q_0\}, x), a) && \text{hipótesis de inducción} \\
&= E\Big(\bigcup_{q \in \delta^*(\{q_0\}, x)} \delta(q, a) \Big) && \text{definición de } \delta_D
\end{aligned}
$$

Lema de la construcción

Afirmación. Para toda cadena w: $\delta_D^*(q_D, w) = \delta^*(\{q_0\}, w)$.

Caso base. $\delta_D^*(q_D, \varepsilon) = q_D = E(\{q_0\}) = \delta^*(\{q_0\}, \varepsilon)$.

Paso inductivo. $w = xa$, y el lema vale para x.

$$
\begin{aligned}
\delta_D^*(q_D, xa) &= \delta_D(\delta_D^*(q_D, x), a) && \text{definición de } \delta_D^* \\
&= \delta_D(\delta^*(\{q_0\}, x), a) && \text{hipótesis de inducción} \\
&= E\Big(\bigcup_{q \in \delta^*(\{q_0\}, x)} \delta(q, a) \Big) && \text{definición de } \delta_D \\
&= \delta^*(\{q_0\}, xa) && \text{definición de } \delta^* \text{ en el NFA}
\end{aligned}
$$

■

Es casi inmediato: δ_D se definió copiando la segunda línea de δ^*.

El teorema, como cadena de equivalencias

D acepta $w \iff \delta_D^*(q_D, w) \in F_D$ aceptar en un DFA

El teorema, como cadena de equivalencias

$$
\begin{aligned}
D \text{ acepta } w &\iff \delta_D^*(q_D, w) \in F_D && \text{aceptar en un DFA} \\
&\iff \delta_D^*(q_D, w) \cap F \neq \emptyset && \text{definición de } F_D
\end{aligned}
$$

El teorema, como cadena de equivalencias

$$
\begin{aligned}
D \text{ acepta } w &\iff \delta_D^*(q_D, w) \in F_D && \text{aceptar en un DFA} \\
&\iff \delta_D^*(q_D, w) \cap F \neq \emptyset && \text{definición de } F_D \\
&\iff \delta^*(\{q_0\}, w) \cap F \neq \emptyset && \text{lema de la construcción}
\end{aligned}
$$

El teorema, como cadena de equivalencias

D acepta w	$\iff \delta_D^*(q_D, w) \in F_D$	aceptar en un DFA
	$\iff \delta_D^*(q_D, w) \cap F \neq \emptyset$	definición de F_D
	$\iff \delta^*(\{q_0\}, w) \cap F \neq \emptyset$	lema de la construcción
	$\iff$ hay un camino de q_0 a F con etiqueta w	lema de los estados activos

El teorema, como cadena de equivalencias

D acepta w	$\iff \delta_D^*(q_D, w) \in F_D$	aceptar en un DFA
	$\iff \delta_D^*(q_D, w) \cap F \neq \emptyset$	definición de F_D
	$\iff \delta^*(\{q_0\}, w) \cap F \neq \emptyset$	lema de la construcción
	$\iff$ hay un camino de q_0 a F con etiqueta w	lema de los estados activos
	$\iff$ N acepta w	aceptar en un NFA

El teorema, como cadena de equivalencias

D acepta w	$\iff \delta_D^*(q_D, w) \in F_D$	aceptar en un DFA
	$\iff \delta_D^*(q_D, w) \cap F \neq \emptyset$	definición de F_D
	$\iff \delta^*(\{q_0\}, w) \cap F \neq \emptyset$	lema de la construcción
	$\iff$ hay un camino de q_0 a F con etiqueta w	lema de los estados activos
	$\iff$ N acepta w	aceptar en un NFA

Corolario

Un lenguaje es regular si y solo si algún NFA lo reconoce.
Desde ahora, para demostrar que un lenguaje es regular **basta dar un NFA**.

El costo: hasta 2^n estados

terminal

```
$ build/kth_table
 k   estados del NFA   subconjuntos posibles   estados del DFA (alcanzables)
 1                 2                       4                               2
 2                 3                       8                               4
 3                 4                      16                               8
 4                 5                      32                              16
 5                 6                      64                              32
 6                 7                     128                              64
 7                 8                     256                             128
 8                 9                     512                             256
 9                10                    1024                             512
10                11                    2048                            1024
11                12                    4096                            2048
12                13                    8192                            4096
```

"El k-ésimo símbolo desde el final es 1." NFA: $k + 1$ estados. DFA construido: 2^k.

El costo: hasta 2^n estados

terminal

```
$ build/kth_table
 k   estados del NFA   subconjuntos posibles   estados del DFA (alcanzables)
 1                 2                       4                               2
 2                 3                       8                               4
 3                 4                      16                               8
 4                 5                      32                              16
 5                 6                      64                              32
 6                 7                     128                              64
 7                 8                     256                             128
 8                 9                     512                             256
 9                10                    1024                             512
10                11                    2048                            1024
11                12                    4096                            2048
12                13                    8192                            4096
```

"El k-ésimo símbolo desde el final es 1." NFA: $k + 1$ estados. DFA construido: 2^k. El subconjunto es q_0 más una foto de los últimos k símbolos.

¿Se puede con menos de 2^k?

No: **todo** DFA para ese lenguaje tiene al menos 2^k estados. La intuición:

¿Se puede con menos de 2^k?

No: **todo** DFA para ese lenguaje tiene al menos 2^k estados. La intuición:

- Con menos estados, dos cadenas distintas $x \neq y$ de longitud k dejan al DFA en el mismo estado.

¿Se puede con menos de 2^k?

No: **todo** DFA para ese lenguaje tiene al menos 2^k estados. La intuición:

- Con menos estados, dos cadenas distintas $x \neq y$ de longitud k dejan al DFA en el mismo estado.
- Difieren en alguna posición. Agregando símbolos al final, esa posición pasa a ser la k-ésima desde el final.

¿Se puede con menos de 2^k?

No: **todo** DFA para ese lenguaje tiene al menos 2^k estados. La intuición:

- Con menos estados, dos cadenas distintas $x \neq y$ de longitud k dejan al DFA en el mismo estado.
- Difieren en alguna posición. Agregando símbolos al final, esa posición pasa a ser la k-ésima desde el final.
- Una queda dentro del lenguaje y la otra fuera, pero la máquina termina en el mismo estado con las dos.

¿Se puede con menos de 2^k?

No: **todo** DFA para ese lenguaje tiene al menos 2^k estados. La intuición:

- Con menos estados, dos cadenas distintas $x \neq y$ de longitud k dejan al DFA en el mismo estado.
- Difieren en alguna posición. Agregando símbolos al final, esa posición pasa a ser la k-ésima desde el final.
- Una queda dentro del lenguaje y la otra fuera, pero la máquina termina en el mismo estado con las dos.

Es la idea. La demostración completa es de la semana 4, con Myhill-Nerode.

¿Se puede con menos de 2^k?

No: **todo** DFA para ese lenguaje tiene al menos 2^k estados. La intuición:

- Con menos estados, dos cadenas distintas $x \neq y$ de longitud k dejan al DFA en el mismo estado.
- Difieren en alguna posición. Agregando símbolos al final, esa posición pasa a ser la k-ésima desde el final.
- Una queda dentro del lenguaje y la otra fuera, pero la máquina termina en el mismo estado con las dos.

Es la idea. La demostración completa es de la semana 4, con Myhill-Nerode.

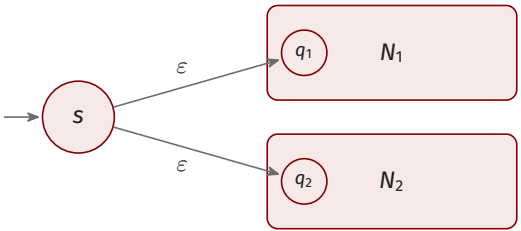
	Simular el NFA	**Convertir a DFA**
Antes de leer	nada	hasta 2^n estados que construir
Por símbolo	del orden de n^2	una búsqueda en la tabla
En la práctica	un patrón que se usa una vez	el analizador léxico de un compilador

SECCIÓN 06

Propiedades de cierre

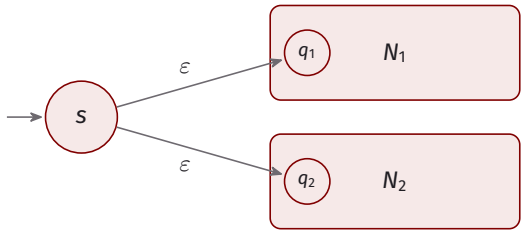
06

Cierre bajo unión



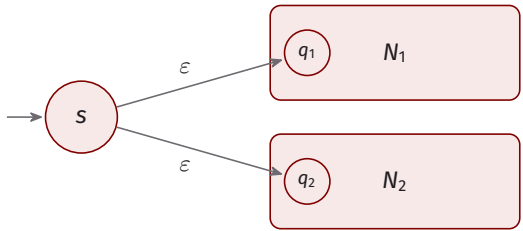
- La máquina **adivina** cuál de los dos va a aceptar, y corre ese.

Cierre bajo unión



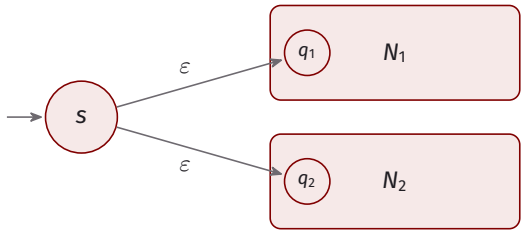
- La máquina **adivina** cuál de los dos va a aceptar, y corre ese.
- Estado inicial nuevo s, con $\delta(s, \varepsilon) = \{q_1, q_2\}$. $F = F_1 \cup F_2$. Lo demás no cambia.

Cierre bajo unión



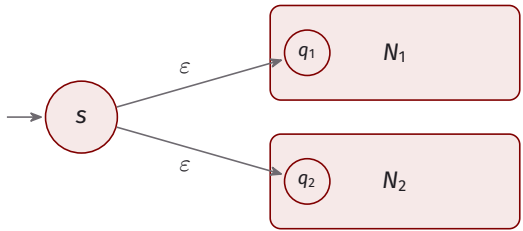
- La máquina **adivina** cuál de los dos va a aceptar, y corre ese.
- Estado inicial nuevo s, con $\delta(s, \varepsilon) = \{q_1, q_2\}$. $F = F_1 \cup F_2$. Lo demás no cambia.
- Si $w \in A_1$: el camino de N_1, con el paso $s \to q_1$ delante.

Cierre bajo unión



- La máquina **adivina** cuál de los dos va a aceptar, y corre ese.
- Estado inicial nuevo s, con $\delta(s, \varepsilon) = \{q_1, q_2\}$. $F = F_1 \cup F_2$. Lo demás no cambia.
- Si $w \in A_1$: el camino de N_1, con el paso $s \to q_1$ delante.
- Si N acepta w: el primer paso va a q_1 o a q_2, y el resto del camino no sale de esa máquina.

Cierre bajo unión



- La máquina **adivina** cuál de los dos va a aceptar, y corre ese.
- Estado inicial nuevo s, con $\delta(s, \varepsilon) = \{q_1, q_2\}$. $F = F_1 \cup F_2$. Lo demás no cambia.
- Si $w \in A_1$: el camino de N_1, con el paso $s \to q_1$ delante.
- Si N acepta w: el primer paso va a q_1 o a q_2, y el resto del camino no sale de esa máquina.

Estados: $|Q_1| + |Q_2| + 1$. El producto de la semana 1: $|Q_1| \cdot |Q_2|$.

Unión: par de unos o termina en 01

terminal

```
$ build/combine union machines/even_ones.dfa machines/ends_01.dfa | build/run_nfa - 1101 111
w = 1101
  {s,1.par,2.q0} -1-> {1.impar,2.q0} -1-> {1.par,2.q0} -0-> {1.par,2.q1} -1-> {1.impar,2.q2}
  termina con {1.impar,2.q2} activos, y 2.q2 está en F: acepta
w = 111
  {s,1.par,2.q0} -1-> {1.impar,2.q0} -1-> {1.par,2.q0} -1-> {1.impar,2.q0}
  termina con {1.impar,2.q0} activos, y ninguno está en F: rechaza
```

- `combine` antepone `1.` y `2.` a los nombres, para que no choquen.

Unión: par de unos o termina en 01

terminal

```
$ build/combine union machines/even_ones.dfa machines/ends_01.dfa | build/run_nfa - 1101 111
w = 1101
  {s,1.par,2.q0} -1-> {1.impar,2.q0} -1-> {1.par,2.q0} -0-> {1.par,2.q1} -1-> {1.impar,2.q2}
  termina con {1.impar,2.q2} activos, y 2.q2 está en F: acepta
w = 111
  {s,1.par,2.q0} -1-> {1.impar,2.q0} -1-> {1.par,2.q0} -1-> {1.impar,2.q0}
  termina con {1.impar,2.q0} activos, y ninguno está en F: rechaza
```

- `combine` antepone `1.` y `2.` a los nombres, para que no choquen.
- Después del primer símbolo, cada conjunto tiene **un estado de cada máquina**: es el par (p, q) del producto.

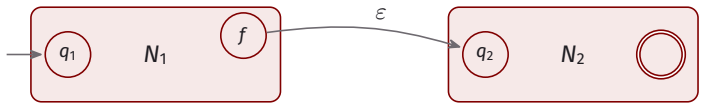
Unión: par de unos o termina en 01

terminal

```
$ build/combine union machines/even_ones.dfa machines/ends_01.dfa | build/run_nfa - 1101 111
w = 1101
  {s,1.par,2.q0} -1-> {1.impar,2.q0} -1-> {1.par,2.q0} -0-> {1.par,2.q1} -1-> {1.impar,2.q2}
  termina con {1.impar,2.q2} activos, y 2.q2 está en F: acepta
w = 111
  {s,1.par,2.q0} -1-> {1.impar,2.q0} -1-> {1.par,2.q0} -1-> {1.impar,2.q0}
  termina con {1.impar,2.q0} activos, y ninguno está en F: rechaza
```

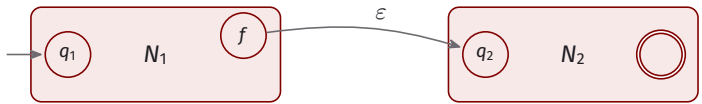
- `combine` antepone `1.` y `2.` a los nombres, para que no choquen.
- Después del primer símbolo, cada conjunto tiene **un estado de cada máquina**: es el par (p, q) del producto.
- Con la construcción de subconjuntos salen 7 estados: los 6 pares y el inicial.

Cierre bajo concatenación



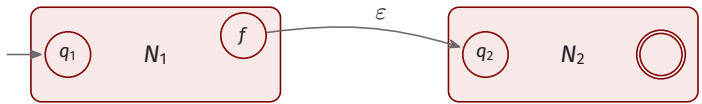
- $A_1A_2 = \{\, uv : u \in A_1, v \in A_2 \,\}$. Hay que hallar el corte: la máquina lo **adivina**.

Cierre bajo concatenación



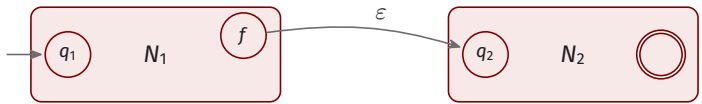
- $A_1A_2 = \{\, uv : u \in A_1, v \in A_2 \,\}$. Hay que hallar el corte: la máquina lo **adivina**.
- Empieza en q_1. De cada $f \in F_1$, una transición ε a q_2. $F = F_2$: los de F_1 dejan de ser de aceptación.

Cierre bajo concatenación



- $A_1A_2 = \{\, uv : u \in A_1, v \in A_2 \,\}$. Hay que hallar el corte: la máquina lo **adivina**.
- Empieza en q_1. De cada $f \in F_1$, una transición ε a q_2. $F = F_2$: los de F_1 dejan de ser de aceptación.
- Si $w = uv$: el camino de u, el paso ε, el camino de v.

Cierre bajo concatenación



- $A_1A_2 = \{\, uv : u \in A_1, v \in A_2 \,\}$. Hay que hallar el corte: la máquina lo **adivina**.
- Empieza en q_1. De cada $f \in F_1$, una transición ε a q_2. $F = F_2$: los de F_1 dejan de ser de aceptación.
- Si $w = uv$: el camino de u, el paso ε, el camino de v.
- Si N acepta w: el camino empieza en Q_1 y termina en Q_2, y solo puede cruzar por una transición nueva, **una vez**. Antes del cruce lee $u \in A_1$; después, $v \in A_2$.

Concatenación: termina en 01, y después par de unos

terminal

```
$ build/combine concat machines/ends_01.dfa machines/even_ones.dfa | build/run_nfa - 0111 011
w = 0111
  {1.q0} -0-> {1.q1} -1-> {1.q2,2.par} -1-> {1.q0,2.impar} -1-> {1.q0,2.par}
  termina con {1.q0,2.par} activos, y 2.par está en F: acepta
w = 011
  {1.q0} -0-> {1.q1} -1-> {1.q2,2.par} -1-> {1.q0,2.impar}
  termina con {1.q0,2.impar} activos, y ninguno está en F: rechaza
```

- La única línea nueva del archivo: `1.q2 eps 2.par`.

Concatenación: termina en 01, y después par de unos

terminal

```
$ build/combine concat machines/ends_01.dfa machines/even_ones.dfa | build/run_nfa - 0111 011
w = 0111
  {1.q0} -0-> {1.q1} -1-> {1.q2,2.par} -1-> {1.q0,2.impar} -1-> {1.q0,2.par}
  termina con {1.q0,2.par} activos, y 2.par está en F: acepta
w = 011
  {1.q0} -0-> {1.q1} -1-> {1.q2,2.par} -1-> {1.q0,2.impar}
  termina con {1.q0,2.impar} activos, y ninguno está en F: rechaza
```

- La única línea nueva del archivo: `1.q2 eps 2.par`.
- Después de `01`, una copia apuesta a que el corte está ahí, y otra sigue en la primera máquina.

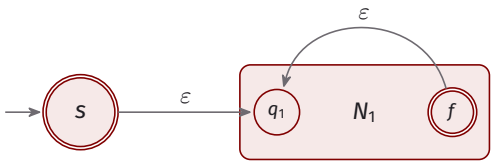
Concatenación: termina en 01, y después par de unos

terminal

```
$ build/combine concat machines/ends_01.dfa machines/even_ones.dfa | build/run_nfa - 0111 011
w = 0111
  {1.q0} -0-> {1.q1} -1-> {1.q2,2.par} -1-> {1.q0,2.impar} -1-> {1.q0,2.par}
  termina con {1.q0,2.par} activos, y 2.par está en F: acepta
w = 011
  {1.q0} -0-> {1.q1} -1-> {1.q2,2.par} -1-> {1.q0,2.impar}
  termina con {1.q0,2.impar} activos, y ninguno está en F: rechaza
```

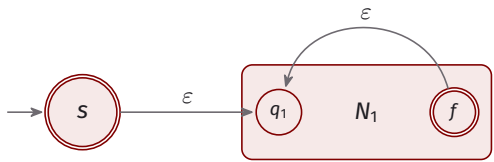
- La única línea nueva del archivo: `1.q2 eps 2.par`.
- Después de `01`, una copia apuesta a que el corte está ahí, y otra sigue en la primera máquina.
- `0111` $=$ `01` $\cdot$ `11`. Con `011`, el único corte deja `1`: una cantidad impar de unos.

Cierre bajo estrella



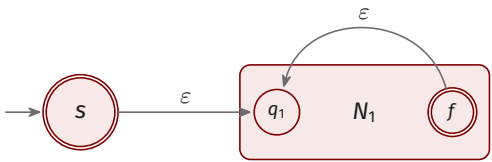
- A^*: cadenas que se parten en cero o más trozos de A. Al terminar un trozo, se puede **volver al principio**.

Cierre bajo estrella



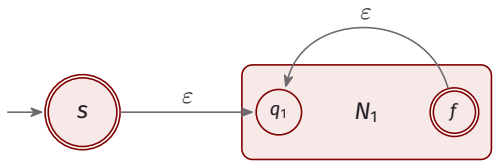
- A^*: cadenas que se parten en cero o más trozos de A. Al terminar un trozo, se puede **volver al principio**.
- Tres cambios: un estado inicial nuevo s, de aceptación; ε de s a q_1; ε de cada $f \in F_1$ a q_1.

Cierre bajo estrella



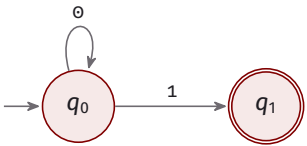
- A^*: cadenas que se parten en cero o más trozos de A. Al terminar un trozo, se puede **volver al principio**.
- Tres cambios: un estado inicial nuevo s, de aceptación; ε de s a q_1; ε de cada $f \in F_1$ a q_1.
- ε siempre se acepta: el camino de cero pasos, porque $s \in F$.

Cierre bajo estrella



- A^*: cadenas que se parten en cero o más trozos de A. Al terminar un trozo, se puede **volver al principio**.
- Tres cambios: un estado inicial nuevo s, de aceptación; ε de s a q_1; ε de cada $f \in F_1$ a q_1.
- ε siempre se acepta: el camino de cero pasos, porque $s \in F$.
- Si N acepta $w \neq \varepsilon$: se corta el camino en cada transición nueva $f \to q_1$. Cada trozo es un camino de N_1 de q_1 a F_1: su etiqueta está en A.

Por qué hace falta un estado inicial nuevo



A: ceros y después un solo 1.

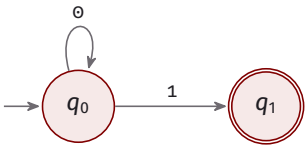
A^*: ε y las que terminan en 1.

El intento: sin s; marcar q_0 como de aceptación.

terminal

```
$ build/wrong_star machines/zeros_then_one.nfa 3
cadena   con estado inicial nuevo   sin estado inicial nuevo
0        rechaza                    acepta
00       rechaza                    acepta
10       rechaza                    acepta
000      rechaza                    acepta
010      rechaza                    acepta
100      rechaza                    acepta
110      rechaza                    acepta
7 cadenas de longitud a lo sumo 3 en las que no coinciden
```

Por qué hace falta un estado inicial nuevo



A: ceros y después un solo 1.

A^*: ε y las que terminan en 1.

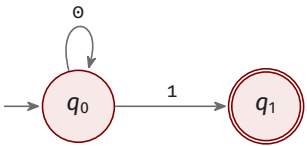
El intento: sin s; marcar q_0 como de aceptación.

terminal

```
$ build/wrong_star machines/zeros_then_one.nfa 3
cadena   con estado inicial nuevo   sin estado inicial nuevo
0        rechaza                    acepta
00       rechaza                    acepta
10       rechaza                    acepta
000      rechaza                    acepta
010      rechaza                    acepta
100      rechaza                    acepta
110      rechaza                    acepta
7 cadenas de longitud a lo sumo 3 en las que no coinciden
```

- 0 vuelve a q_0 por su propia flecha, sin completar ningún trozo, y queda aceptada. Pero $0 \notin A^*$.

Por qué hace falta un estado inicial nuevo



A: ceros y después un solo 1.

A^*: ε y las que terminan en 1.

El intento: sin s; marcar q_0 como de aceptación.

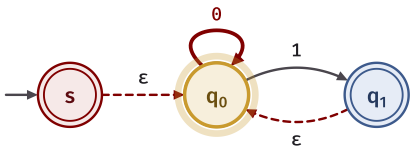
terminal

```
$ build/wrong_star machines/zeros_then_one.nfa 3
cadena   con estado inicial nuevo   sin estado inicial nuevo
0        rechaza                    acepta
00       rechaza                    acepta
10       rechaza                    acepta
000      rechaza                    acepta
010      rechaza                    acepta
100      rechaza                    acepta
110      rechaza                    acepta
7 cadenas de longitud a lo sumo 3 en las que no coinciden
```

- 0 vuelve a q_0 por su propia flecha, sin completar ningún trozo, y queda aceptada. Pero $0 \notin A^*$.
- "Estar en q_0" pasó a significar dos cosas. El estado s las separa.

Las dos construcciones, con la cadena 0

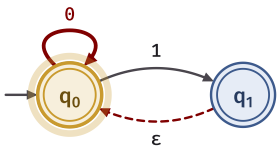
CON ESTADO INICIAL NUEVO



con 0: $\{s, q_0\} \xrightarrow{0} \{q_0\}$

q_0 no está en F: rechaza 0

SIN ESTADO NUEVO: INCORRECTA



con 0: $\{q_0\} \xrightarrow{0} \{q_0\}$

q_0 ahora está en F: acepta 0

A son las cadenas de ceros seguidos de un solo 1, así que 0 no está en A^*. El bucle de q_0 vuelve al estado inicial a mitad de un trozo: sin s, estar en q_0 significa a la vez «acabo de terminar un trozo» y «estoy a mitad de uno».

Cierre bajo reverso

$A^R = \{ w^R : w \in A \}$. La idea: recorrer la máquina **hacia atrás**.

terminal

```
$ build/combine reverse machines/ends_01.dfa
# Reverso de machines/ends_01.dfa.
states: s q0 q1 q2
alphabet: 0 1
start: s
accept: q0
delta:
s eps q2
q0 1 q0 q2
q1 0 q0 q1 q2
q2 1 q1
```

Termina en 01 pasa a ser *empieza con 10*.

Cierre bajo reverso

$A^R = \{ w^R : w \in A \}$. La idea: recorrer la máquina **hacia atrás**.

- Todas las flechas, al revés.
- Estado inicial nuevo s, con ε a cada estado que era de aceptación.
- Único estado de aceptación: el que era inicial.

terminal

```
$ build/combine reverse machines/ends_01.dfa
# Reverso de machines/ends_01.dfa.
states: s q0 q1 q2
alphabet: 0 1
start: s
accept: q0
delta:
s eps q2
q0 1 q0 q2
q1 0 q0 q1 q2
q2 1 q1
```

Termina en 01 pasa a ser *empieza con 10*.

Cierre bajo reverso

$A^R = \{ w^R : w \in A \}$. La idea: recorrer la máquina **hacia atrás**.

- Todas las flechas, al revés.
- Estado inicial nuevo s, con ε a cada estado que era de aceptación.
- Único estado de aceptación: el que era inicial.
- Un camino de N_1 con etiqueta w, invertido, es un camino con etiqueta w^R.

terminal

```
$ build/combine reverse machines/ends_01.dfa
# Reverso de machines/ends_01.dfa.
states: s q0 q1 q2
alphabet: 0 1
start: s
accept: q0
delta:
s eps q2
q0 1 q0 q2
q1 0 q0 q1 q2
q2 1 q1
```

Termina en 01 pasa a ser *empieza con 10*.

Cierre bajo reverso

$A^R = \{ w^R : w \in A \}$. La idea: recorrer la máquina **hacia atrás**.

- Todas las flechas, al revés.
- Estado inicial nuevo s, con ε a cada estado que era de aceptación.
- Único estado de aceptación: el que era inicial.
- Un camino de N_1 con etiqueta w, invertido, es un camino con etiqueta w^R.
- El reverso de un DFA no es un DFA: de q_1 salen tres flechas con 0.

terminal

```
$ build/combine reverse machines/ends_01.dfa
# Reverso de machines/ends_01.dfa.
states: s q0 q1 q2
alphabet: 0 1
start: s
accept: q0
delta:
s eps q2
q0 1 q0 q2
q1 0 q0 q1 q2
q2 1 q1
```

Termina en 01 pasa a ser *empieza con 10*.

El complemento no se hace en el NFA

terminal

```
$ build/swap_accept machines/third_from_end.nfa 100 000
F = {q3}    Q − F = {q0,q1,q2}
w = 100
  activos al final: {q0,q3}
  el NFA:                          acepta
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: rechaza
w = 000
  activos al final: {q0}
  el NFA:                          rechaza
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: acepta
```

El intento: intercambiar F por $Q - F$, como en un DFA.

El complemento no se hace en el NFA

terminal

```
$ build/swap_accept machines/third_from_end.nfa 100 000
F = {q3}    Q − F = {q0,q1,q2}
w = 100
  activos al final: {q0,q3}
  el NFA:                          acepta
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: rechaza
w = 000
  activos al final: {q0}
  el NFA:                          rechaza
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: acepta
```

El intento: intercambiar F por $Q - F$, como en un DFA.

- `100` se acepta antes por q_3 y después por q_0.

El complemento no se hace en el NFA

terminal

```
$ build/swap_accept machines/third_from_end.nfa 100 000
F = {q3}    Q − F = {q0,q1,q2}
w = 100
  activos al final: {q0,q3}
  el NFA:                          acepta
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: rechaza
w = 000
  activos al final: {q0}
  el NFA:                          rechaza
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: acepta
```

El intento: intercambiar F por $Q - F$, como en un DFA.

- `100` se acepta antes por q_3 y después por q_0.
- N acepta si *algún* camino termina en F. El complemento pide que *ninguno* termine en F.
- El intercambio da "*algún* camino termina fuera de F".

El complemento no se hace en el NFA

terminal

```
$ build/swap_accept machines/third_from_end.nfa 100 000
F = {q3}    Q − F = {q0,q1,q2}
w = 100
  activos al final: {q0,q3}
  el NFA:                          acepta
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: rechaza
w = 000
  activos al final: {q0}
  el NFA:                          rechaza
  el NFA con Q − F:                acepta
  el DFA de subconjuntos, Q − F: acepta
```

El intento: intercambiar F por $Q - F$, como en un DFA.

- `100` se acepta antes por q_3 y después por q_0.
- N acepta si *algún* camino termina en F. El complemento pide que *ninguno* termine en F.
- El intercambio da "*algún* camino termina fuera de F".
- Lo correcto: primero la construcción de subconjuntos, después el intercambio en el DFA.

Las propiedades de cierre, juntas

Operación	Construcción	Estados
Unión	producto de dos DFA, o NFA con inicial nuevo	$\lvert Q_1\rvert \cdot \lvert Q_2\rvert$, o $\lvert Q_1\rvert + \lvert Q_2\rvert + 1$
Intersección	producto	$\lvert Q_1\rvert \cdot \lvert Q_2\rvert$
Complemento	intercambiar F en un **DFA**	$\lvert Q\rvert$
Concatenación	NFA con ε de F_1 a q_2	$\lvert Q_1\rvert + \lvert Q_2\rvert$
Estrella	NFA con inicial nuevo	$\lvert Q_1\rvert + 1$
Reverso	NFA con las flechas invertidas	$\lvert Q_1\rvert + 1$

Operaciones regulares

Unión, concatenación y estrella. La próxima semana: con esas tres se obtienen **todos** los lenguajes regulares. Eso son las expresiones regulares.

Laboratorio: de NFA a DFA

Entrada: un NFA con transiciones ε. Salida: un DFA equivalente.

1. Leer el NFA y numerar sus estados.
2. Calcular la clausura ε con una búsqueda en el grafo.
3. Generar solo los subconjuntos alcanzables, con una cola, y darles nombres nuevos.
4. Pasar el verificador: compara **lenguajes**, no nombres.
5. Llenar la tabla de k de 1 a 12.

Dónde: `code/week-02/lab/nfa_to_dfa`.
Hasta 60 estados en el NFA; a lo sumo 10 000 en el DFA.

casos/03.in

```
states: qa qb qc
alphabet: a b c
start: qa
accept: qc
delta:
qa a qa
qa eps qb
qb b qb
qb eps qc
qc c qc
```

Sin las líneas de comentario. Generar los 2^n subconjuntos no pasa.

Para la próxima clase

- Leer los apuntes de la semana 2 y terminar el laboratorio.
- Preguntas de repaso de la guía: seguir a mano un NFA con transiciones ε; por qué basta un estado de F en el subconjunto; un NFA donde intercambiar F no da el complemento; qué falla en la estrella sin estado inicial nuevo.
- Próxima semana: expresiones regulares, y por qué describen exactamente los lenguajes regulares.

Pregunta

Con unión, concatenación y estrella, partiendo de $\{0\}$ y $\{1\}$, ¿cómo se escribe “el tercer símbolo desde el final es 1”?