

Autómatas y Lenguajes Formales 2016-1

Maestría en Ciencia e Ingeniería de la Computación UNAM

Tema 3: Autómatas Finitos Deterministas II, Autómatas No-Deterministas

Dr. Favio Ezequiel Miranda Perea
favio@ciencias.unam.mx

Facultad de Ciencias UNAM

29 de enero de 2020



Métodos de Diseño

AFD

- Problema de diseño: Dado un lenguaje $L \subseteq \Sigma^*$ diseñar un AFD M que acepte exactamente a L , es decir, tal que $L(M) = L$.



Métodos de Diseño

AFD

- Problema de diseño: Dado un lenguaje $L \subseteq \Sigma^*$ diseñar un AFD M que acepte exactamente a L , es decir, tal que $L(M) = L$.
- El ensayo y error es inconveniente, las posibilidades son demasiadas.



Métodos de Diseño

AFD

- Problema de diseño: Dado un lenguaje $L \subseteq \Sigma^*$ diseñar un AFD M que acepte exactamente a L , es decir, tal que $L(M) = L$.
- El ensayo y error es inconveniente, las posibilidades son demasiadas.
- El autómata debe ser **completo**, es decir, debe aceptar todas las palabras de L , en otras palabras es necesario que $L \subseteq L(M)$.



Métodos de Diseño

AFD

- Problema de diseño: Dado un lenguaje $L \subseteq \Sigma^*$ diseñar un AFD M que acepte exactamente a L , es decir, tal que $L(M) = L$.
- El ensayo y error es inconveniente, las posibilidades son demasiadas.
- El autómata debe ser **completo**, es decir, debe aceptar todas las palabras de L , en otras palabras es necesario que $L \subseteq L(M)$.
- El autómata debe ser **correcto**, es decir, debe aceptar sólo palabras de L , en otras palabras es necesario que $L(M) \subseteq L$.



Métodos de Diseño

AFD

- Se debe determinar explicitamente qué condición “recuerda” cada estado.



Métodos de Diseño

AFD

- Se debe determinar explicitamente qué condición “recuerda” cada estado.
- La única memoria que tiene un AFD son los estados.



Métodos de Diseño

AFD

- Se debe determinar explicitamente qué condición “recuerda” cada estado.
- La única memoria que tiene un AFD son los estados.
- Primero se propone un conjunto de estados que “recuerden” condiciones importantes.



Métodos de Diseño

AFD

- Se debe determinar explicitamente qué condición “recuerda” cada estado.
- La única memoria que tiene un AFD son los estados.
- Primero se propone un conjunto de estados que “recuerden” condiciones importantes.
- Despues se proponen las transiciones para cambiar de un estado a otro.



Métodos de Diseño

AFD

- Se debe determinar explicitamente qué condición “recuerda” cada estado.
- La única memoria que tiene un AFD son los estados.
- Primero se propone un conjunto de estados que “recuerden” condiciones importantes.
- Después se proponen las transiciones para cambiar de un estado a otro.
- Finalmente se determina que estados son finales, observando que condiciones de estado corresponden con la definición del lenguaje aceptado.



Métodos de Diseño

AFD

- Se debe determinar explicitamente qué condición “recuerda” cada estado.
- La única memoria que tiene un AFD son los estados.
- Primero se propone un conjunto de estados que “recuerden” condiciones importantes.
- Después se proponen las transiciones para cambiar de un estado a otro.
- Finalmente se determina que estados son finales, observando que condiciones de estado corresponden con la definición del lenguaje aceptado.
- Ejemplo: $L = \{w \in \{0, 1\}^* \mid w \text{ tiene un número par de } 0 \text{ y } 1\}$



Diseño por conjuntos de estados

AFD

- Muchas veces es preferible generalizar condiciones para definir grupos de estados en lugar de dar condiciones estado por estado.



Diseño por conjuntos de estados

AFD

- Muchas veces es preferible generalizar condiciones para definir grupos de estados en lugar de dar condiciones estado por estado.
- De esta forma se disminuye la posibilidad de error y se facilita la solución de un problema complejo.



Diseño por conjuntos de estados

AFD

- Muchas veces es preferible generalizar condiciones para definir grupos de estados en lugar de dar condiciones estado por estado.
- De esta forma se disminuye la posibilidad de error y se facilita la solución de un problema complejo.
- Las condiciones para grupos de estados deben ser:



Diseño por conjuntos de estados

AFD

- Muchas veces es preferible generalizar condiciones para definir grupos de estados en lugar de dar condiciones estado por estado.
- De esta forma se disminuye la posibilidad de error y se facilita la solución de un problema complejo.
- Las condiciones para grupos de estados deben ser:
 - ▶ Excluyentes: un grupo de estados debe cumplir únicamente una condición.



Diseño por conjuntos de estados

AFD

- Muchas veces es preferible generalizar condiciones para definir grupos de estados en lugar de dar condiciones estado por estado.
- De esta forma se disminuye la posibilidad de error y se facilita la solución de un problema complejo.
- Las condiciones para grupos de estados deben ser:
 - ▶ Excluyentes: un grupo de estados debe cumplir únicamente una condición.
 - ▶ Comprensivas: los grupos cumplen todos los casos posibles.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$


Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
- Contiene 00 pero no 11.

Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
- Contiene 00 pero no 11.
- Contiene 11.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$


Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
 - ▶ No se han leído símbolos.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
 - ▶ No se han leído símbolos.
 - ▶ Se leyó un 0.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
 - ▶ No se han leído símbolos.
 - ▶ Se leyó un 0.
 - ▶ Se leyó un 1.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
 - ▶ No se han leído símbolos.
 - ▶ Se leyó un 0.
 - ▶ Se leyó un 1.
- Contiene 00 pero no 11.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
 - ▶ No se han leído símbolos.
 - ▶ Se leyó un 0.
 - ▶ Se leyó un 1.
- Contiene 00 pero no 11.
 - ▶ Se leyó otro 0.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
 - ▶ No se han leído símbolos.
 - ▶ Se leyó un 0.
 - ▶ Se leyó un 1.
- Contiene 00 pero no 11.
 - ▶ Se leyó otro 0.
 - ▶ Se leyó un 1.



Diseño por conjuntos de estados

Ejemplo

$$L = \{w \in \{0, 1\}^* \mid w \text{ no contiene } 11 \text{ pero sí } 00\}$$

Los grupos de estados se diseñan según las condiciones:

- La palabra consumida no contiene ni 00 ni 11.
 - ▶ No se han leído símbolos.
 - ▶ Se leyó un 0.
 - ▶ Se leyó un 1.
- Contiene 00 pero no 11.
 - ▶ Se leyó otro 0.
 - ▶ Se leyó un 1.
- Contiene 11.



Diseño por complemento

AFD

- Dado un lenguaje L a veces es más fácil diseñar un autómata para el complemento $\bar{L} = \Sigma^* - L$.



Diseño por complemento

AFD

- Dado un lenguaje L a veces es más fácil diseñar un autómata para el complemento $\bar{L} = \Sigma^* - L$.
- Si $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ reconoce a L , es decir $L(M) = L$ entonces



Diseño por complemento

AFD

- Dado un lenguaje L a veces es más fácil diseñar un autómata para el complemento $\bar{L} = \Sigma^* - L$.
- Si $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ reconoce a L , es decir $L(M) = L$ entonces
- $M^c = \langle Q, \Sigma, \delta, q_0, Q - F \rangle$ reconoce a $\Sigma^* - L$, es decir $L(M^c) = \bar{L}$



Diseño por complemento

AFD

- Dado un lenguaje L a veces es más fácil diseñar un autómata para el complemento $\bar{L} = \Sigma^* - L$.
- Si $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ reconoce a L , es decir $L(M) = L$ entonces
- $M^c = \langle Q, \Sigma, \delta, q_0, Q - F \rangle$ reconoce a $\Sigma^* - L$, es decir $L(M^c) = \bar{L}$
- Este método sólo funciona para autómatas deterministas.



Diseño por complemento

AFD

- Dado un lenguaje L a veces es más fácil diseñar un autómata para el complemento $\bar{L} = \Sigma^* - L$.
- Si $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ reconoce a L , es decir $L(M) = L$ entonces
- $M^c = \langle Q, \Sigma, \delta, q_0, Q - F \rangle$ reconoce a $\Sigma^* - L$, es decir $L(M^c) = \bar{L}$
- Este método sólo funciona para autómatas deterministas.
- Ejemplo: Diseñar un AFD para
 $L = \{w \in \{a, b\}^* \mid w \text{ no contiene } abaab\}$



Funcionamiento de un autómata

Informal

- M reconoce o acepta una cadena w si



Funcionamiento de un autómata

Informal

- M reconoce o acepta una cadena w si
- Se consumen todos los símbolos de w al iniciar en q_0 siguiendo las transiciones de acuerdo a δ .



Funcionamiento de un autómata

Informal

- M reconoce o acepta una cadena w si
- Se consumen todos los símbolos de w al iniciar en q_0 siguiendo las transiciones de acuerdo a δ .
- Al terminar, el estado actual de la máquina es final.



Funcionamiento de un autómata

Informal

- M reconoce o acepta una cadena w si
- Se consumen todos los símbolos de w al iniciar en q_0 siguiendo las transiciones de acuerdo a δ .
- Al terminar, el estado actual de la máquina es final.
- El lenguaje aceptado es entonces:

$$L(M) := \{u \in \Sigma^* : M \text{ se detiene al procesar } u \text{ en un estado } q \in F\}.$$



Funcionamiento de un autómata

Informal

- M reconoce o acepta una cadena w si
- Se consumen todos los símbolos de w al iniciar en q_0 siguiendo las transiciones de acuerdo a δ .
- Al terminar, el estado actual de la máquina es final.
- El lenguaje aceptado es entonces:

$$L(M) := \{u \in \Sigma^* : M \text{ se detiene al procesar } u \text{ en un estado } q \in F\}.$$

- Esta definición es todavía demasiado informal.



Función de transición extendida

 δ^*

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$



Función de transición extendida

 δ^*

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

definida recursivamente como sigue:



Función de transición extendida

 δ^*

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$



Función de transición extendida

 δ^*

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$



Función de transición extendida

 δ^*

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

δ^* se llama la función de transición extendida de M .



Función de transición extendida

 δ^*

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

δ^* se llama la función de transición extendida de M .

Una definición alternativa de utilidad es:



Función de transición extendida

 δ^*

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, wa) = \delta(\delta^*(q, w), a)$$

δ^* se llama la función de transición extendida de M .

Una definición alternativa de utilidad es:

$$\delta^*(q, aw) = \delta^*(\delta(q, a), w)$$



Función de transición extendida

δ^*

La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

- De la definición se sigue que $\delta^*(q, w)$ devuelve el estado en el que la máquina termina al procesar w .



Función de transición extendida

δ^*

La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow Q$$

- De la definición se sigue que $\delta^*(q, w)$ devuelve el estado en el que la máquina termina al procesar w .
- Dado que δ^* es una extensión de δ es usual sobrecargar la operación y escribir δ en lugar de δ^* .



Función de transición extendida

Lenguaje de aceptación

- La definición informal de lenguaje de aceptación de un autómata puede ahora formalizarse haciendo referencia a la función de transición extendida:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$



Función de transición extendida

Lenguaje de aceptación

- La definición informal de lenguaje de aceptación de un autómata puede ahora formalizarse haciendo referencia a la función de transición extendida:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

- De esta manera es posible verificar formalmente la completud y correctud del diseño de un Autómata M para un lenguaje L :



Función de transición extendida

Lenguaje de aceptación

- La definición informal de lenguaje de aceptación de un autómata puede ahora formalizarse haciendo referencia a la función de transición extendida:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

- De esta manera es posible verificar formalmente la completud y correctud del diseño de un Autómata M para un lenguaje L :
- Correctud ($L(M) \subseteq L$): Si $\delta^*(q_0, w) \in F$ entonces $w \in L$.



Función de transición extendida

Lenguaje de aceptación

- La definición informal de lenguaje de aceptación de un autómata puede ahora formalizarse haciendo referencia a la función de transición extendida:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$$

- De esta manera es posible verificar formalmente la completud y correctud del diseño de un Autómata M para un lenguaje L :
- Correctud ($L(M) \subseteq L$): Si $\delta^*(q_0, w) \in F$ entonces $w \in L$.
- Completud ($L \subseteq L(M)$): Si $w \in L$ entonces $\delta^*(q_0, w) \in F$.



Modularización

- Algunas veces la especificación de un lenguaje L tiene una forma lógica que permite descomponerlo en lenguajes más sencillos



Modularización

- Algunas veces la especificación de un lenguaje L tiene una forma lógica que permite descomponerlo en lenguajes más sencillos
- De esta manera el diseño de un AFD se puede modularizar.



Modularización

- Algunas veces la especificación de un lenguaje L tiene una forma lógica que permite descomponerlo en lenguajes más sencillos
- De esta manera el diseño de un AFD se puede modularizar.
- Por ejemplo si $L = L_1 \cup L_2$ basta construir autómatas para L_1 y L_2 y ejecutarlos en paralelo.



Modularización

- Algunas veces la especificación de un lenguaje L tiene una forma lógica que permite descomponerlo en lenguajes más sencillos
- De esta manera el diseño de un AFD se puede modularizar.
- Por ejemplo si $L = L_1 \cup L_2$ basta construir autómatas para L_1 y L_2 y ejecutarlos en paralelo.
- Lo mismo sucede si $L = L_1 \cap L_2$ o si $L = L_1 - L_2$.



Modularización

- Algunas veces la especificación de un lenguaje L tiene una forma lógica que permite descomponerlo en lenguajes más sencillos
- De esta manera el diseño de un AFD se puede modularizar.
- Por ejemplo si $L = L_1 \cup L_2$ basta construir autómatas para L_1 y L_2 y ejecutarlos en paralelo.
- Lo mismo sucede si $L = L_1 \cap L_2$ o si $L = L_1 - L_2$.
- Por ejemplo
$$L = \{w \in \{0, 1\}^* \mid w \text{ acaba en } 0 \text{ y } 11 \text{ no es subcadena de } w\}$$



Modularización

- Algunas veces la especificación de un lenguaje L tiene una forma lógica que permite descomponerlo en lenguajes más sencillos
- De esta manera el diseño de un AFD se puede modularizar.
- Por ejemplo si $L = L_1 \cup L_2$ basta construir autómatas para L_1 y L_2 y ejecutarlos en paralelo.
- Lo mismo sucede si $L = L_1 \cap L_2$ o si $L = L_1 - L_2$.
- Por ejemplo
$$L = \{w \in \{0, 1\}^* \mid w \text{ acaba en } 0 \text{ y } 11 \text{ no es subcadena de } w\}$$
- se descompone como $L = L_1 - L_2$ donde
$$L_1 = \{w \in \{0, 1\}^* \mid w \text{ acaba en } 0\}$$
 y
$$L_2 = \{w \in \{0, 1\}^* \mid 11 \text{ es subcadena de } w\}$$



El autómata producto

Modularización

Sean $M_1 = \langle Q_1, \Sigma, \delta_1, q_{01}, F_1 \rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{02}, F_2 \rangle$ AFD tales que $L_1 = L(M_1)$ y $L_2 = L(M_2)$. El autómata producto, denotado $M_1 \times M_2$ se define como sigue:

$$M_1 \times M_2 = \langle Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F \rangle$$

donde

- $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ y



El autómata producto

Modularización

Sean $M_1 = \langle Q_1, \Sigma, \delta_1, q_{01}, F_1 \rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{02}, F_2 \rangle$ AFD tales que $L_1 = L(M_1)$ y $L_2 = L(M_2)$. El autómata producto, denotado $M_1 \times M_2$ se define como sigue:

$$M_1 \times M_2 = \langle Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F \rangle$$

donde

- $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ y
- Si $F = F_1 \times F_2$ entonces $L(M_1 \times M_2) = L_1 \cap L_2$



El autómata producto

Modularización

Sean $M_1 = \langle Q_1, \Sigma, \delta_1, q_{01}, F_1 \rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{02}, F_2 \rangle$ AFD tales que $L_1 = L(M_1)$ y $L_2 = L(M_2)$. El autómata producto, denotado $M_1 \times M_2$ se define como sigue:

$$M_1 \times M_2 = \langle Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F \rangle$$

donde

- $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ y
- Si $F = F_1 \times F_2$ entonces $L(M_1 \times M_2) = L_1 \cap L_2$
- Si $F = F_1 \times \overline{F_2}$ entonces $L(M_1 \times M_2) = L_1 - L_2$



El autómata producto

Modularización

Sean $M_1 = \langle Q_1, \Sigma, \delta_1, q_{01}, F_1 \rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{02}, F_2 \rangle$ AFD tales que $L_1 = L(M_1)$ y $L_2 = L(M_2)$. El autómata producto, denotado $M_1 \times M_2$ se define como sigue:

$$M_1 \times M_2 = \langle Q_1 \times Q_2, \Sigma, \delta, (q_{01}, q_{02}), F \rangle$$

donde

- $\delta((p, q), a) = (\delta_1(p, a), \delta_2(q, a))$ y
- Si $F = F_1 \times F_2$ entonces $L(M_1 \times M_2) = L_1 \cap L_2$
- Si $F = F_1 \times \overline{F_2}$ entonces $L(M_1 \times M_2) = L_1 - L_2$
- Si $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ entonces $L(M_1 \times M_2) = L_1 \cup L_2$



No Determinismo

AFN

- El determinismo de un autómata, deseable desde el punto de vista teórico, puede provocar complicaciones en la práctica.



No Determinismo

AFN

- El determinismo de un autómata, deseable desde el punto de vista teórico, puede provocar complicaciones en la práctica.
- El no-determinismo es deseable.



No Determinismo

AFN

- El determinismo de un autómata, deseable desde el punto de vista teórico, puede provocar complicaciones en la práctica.
- El no-determinismo es deseable.
- Determinismo: dado un estado q y un símbolo a existe una única transición $\delta(q, a) = p$, es decir δ es una función.



No Determinismo

AFN

- El determinismo de un autómata, deseable desde el punto de vista teórico, puede provocar complicaciones en la práctica.
- El no-determinismo es deseable.
- Determinismo: dado un estado q y un símbolo a existe una única transición $\delta(q, a) = p$, es decir δ es una función.
- No-determinismo: no hay una transición única al leer un símbolo a en un estado dado q .



No Determinismo

AFN

- Hay más de una transición, es decir, $\delta(q, a)$ deja de ser función.



No Determinismo

AFN

- Hay más de una transición, es decir, $\delta(q, a)$ deja de ser función.
- O bien no hay transición, es decir, $\delta(q, a)$ no está definida (δ se vuelve función parcial).



No Determinismo

AFN

- Hay más de una transición, es decir, $\delta(q, a)$ deja de ser función.
- O bien no hay transición, es decir, $\delta(q, a)$ no está definida (δ se vuelve función parcial).
- Sin embargo la máquina funciona únicamente al leer un símbolo.



No Determinismo

AFN

- Hay más de una transición, es decir, $\delta(q, a)$ deja de ser función.
- O bien no hay transición, es decir, $\delta(q, a)$ no está definida (δ se vuelve función parcial).
- Sin embargo la máquina funciona únicamente al leer un símbolo.
- Existe el no-determinismo sin lectura de símbolos.



Autómatas Finitos No-Deterministas

Definición

Un autómata finito **no** determinista (AFN) es una quintupla

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$



Autómatas Finitos No-Deterministas

Definición

Un autómata finito **no** determinista (AFN) es una quintupla

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

donde

- Q es un conjunto finito de estados.



Autómatas Finitos No-Deterministas

Definición

Un autómata finito **no** determinista (AFN) es una quintupla

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

donde

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.



Autómatas Finitos No-Deterministas

Definición

Un autómata finito **no** determinista (AFN) es una quintupla

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

donde

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ es la función de transición.



Autómatas Finitos No-Deterministas

Definición

Un autómata finito **no** determinista (AFN) es una quintupla

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

donde

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.



Autómatas Finitos No-Deterministas

Definición

Un autómata finito **no** determinista (AFN) es una quintupla

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle$$

donde

- Q es un conjunto finito de estados.
- Σ es el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.



Autómatas Finitos No-Deterministas

Función de transición

$$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$$



Autómatas Finitos No-Deterministas

Función de transición

$$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$$

- Obsérvese que la imagen de δ es ahora un elemento de $\mathcal{P}(Q)$, es decir es un subconjunto de estados de Q .



Autómatas Finitos No-Deterministas

Función de transición

$$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$$

- Obsérvese que la imagen de δ es ahora un elemento de $\mathcal{P}(Q)$, es decir es un subconjunto de estados de Q .
- $\delta(q, a) = \{q_1, q_2, \dots, q_n\}$ indica que al leer a en el estado q la máquina puede pasar a cualquiera de los estados q_1, \dots, q_n .



Autómatas Finitos No-Deterministas

Función de transición

$$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$$

- Obsérvese que la imagen de δ es ahora un elemento de $\mathcal{P}(Q)$, es decir es un subconjunto de estados de Q .
- $\delta(q, a) = \{q_1, q_2, \dots, q_n\}$ indica que al leer a en el estado q la máquina puede pasar a cualquiera de los estados q_1, \dots, q_n .
- Si $\delta(q, a) = \emptyset$ entonces no hay transición posible desde q al leer a , es decir, la máquina está bloqueada.



Función δ^*

AFN

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFN. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$



Función δ^*

AFN

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFN. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

definida recursivamente como sigue:



Función δ^*

AFN

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFN. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$



Función δ^*

AFN

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFN. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, wa) = \bigcup_{p \in \delta^*(q, w)} \delta(p, a)$$



Función δ^*

AFN

Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFN. La función de transición δ se extiende a cadenas mediante una función

$$\delta^* : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$$

definida recursivamente como sigue:

$$\delta^*(q, \varepsilon) = q$$

$$\delta^*(q, wa) = \bigcup_{p \in \delta^*(q, w)} \delta(p, a)$$

Alternativamente:

$$\delta^*(q, aw) = \bigcup_{p \in \delta(q, a)} \delta^*(p, w)$$

Función de transición extendida

Lenguaje aceptado

El lenguaje de aceptación se define mediante δ^* como sigue:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$$



Función de transición extendida

Lenguaje aceptado

El lenguaje de aceptación se define mediante δ^* como sigue:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$$



Función de transición extendida

Lenguaje aceptado

El lenguaje de aceptación se define mediante δ^* como sigue:

$$L(M) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \cap F \neq \emptyset\}$$

Es decir, $w \in L(M)$ si y sólo si existe al menos un cómputo que conduce a un estado final al iniciar la máquina en q_0 y w .



Eliminación del No-determinismo

AFN ⇔ AFD

- Todo AFD es a la vez un AFN con la particularidad de que $\delta(p, a)$ consta de un único estado.



Eliminación del No-determinismo

AFN ⇔ AFD

- Todo AFD es a la vez un AFN con la particularidad de que $\delta(p, a)$ consta de un único estado.
- La idea para transformar un AFN en un AFD es considerar a cada conjunto de estados $\delta(p, a)$ del AFN como un único estado del nuevo AFD.



Eliminación del No-determinismo

AFN ⇔ AFD

- Todo AFD es a la vez un AFN con la particularidad de que $\delta(p, a)$ consta de un único estado.
- La idea para transformar un AFN en un AFD es considerar a cada conjunto de estados $\delta(p, a)$ del AFN como un único estado del nuevo AFD.
- Este método se conoce como la **construcción de subconjuntos**.



AFN ⇒ AFD

Construcción de subconjuntos

Dado un AFN $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ definimos un AFD $M^d = \langle Q^d, \Sigma^d, \delta, q_0^d, F^d \rangle$ como sigue:



AFN ⇒ AFD

Construcción de subconjuntos

Dado un AFN $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ definimos un AFD $M^d = \langle Q^d, \Sigma^d, \delta, q_0^d, F^d \rangle$ como sigue:

- $Q^d = \mathcal{P}(Q)$.

AFN ⇒ AFD

Construcción de subconjuntos

Dado un AFN $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ definimos un AFD $M^d = \langle Q^d, \Sigma^d, \delta, q_0^d, F^d \rangle$ como sigue:

- $Q^d = \mathcal{P}(Q)$.
- $\Sigma^d = \Sigma$.



AFN ⇒ AFD

Construcción de subconjuntos

Dado un AFN $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ definimos un AFD $M^d = \langle Q^d, \Sigma^d, \delta, q_0^d, F^d \rangle$ como sigue:

- $Q^d = \mathcal{P}(Q)$.
- $\Sigma^d = \Sigma$.
- $\delta(S, a) = \delta_N(S, a) = \bigcup_{q \in S} \delta_N(q, a)$



AFN ⇒ AFD

Construcción de subconjuntos

Dado un AFN $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ definimos un AFD $M^d = \langle Q^d, \Sigma^d, \delta, q_0^d, F^d \rangle$ como sigue:

- $Q^d = \mathcal{P}(Q)$.
- $\Sigma^d = \Sigma$.
- $\delta(S, a) = \delta_N(S, a) = \bigcup_{q \in S} \delta_N(q, a)$
- $q_0^d = \{q_0\}$



AFN ⇒ AFD

Construcción de subconjuntos

Dado un AFN $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ definimos un AFD $M^d = \langle Q^d, \Sigma^d, \delta, q_0^d, F^d \rangle$ como sigue:

- $Q^d = \mathcal{P}(Q)$.
- $\Sigma^d = \Sigma$.
- $\delta(S, a) = \delta_N(S, a) = \bigcup_{q \in S} \delta_N(q, a)$
- $q_0^d = \{q_0\}$
- $F^d = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$



AFN ⇒ AFD

Construcción de subconjuntos

Dado un AFN $M = \langle Q, \Sigma, \delta_N, q_0, F \rangle$ definimos un AFD $M^d = \langle Q^d, \Sigma^d, \delta, q_0^d, F^d \rangle$ como sigue:

- $Q^d = \mathcal{P}(Q)$.
- $\Sigma^d = \Sigma$.
- $\delta(S, a) = \delta_N(S, a) = \bigcup_{q \in S} \delta_N(q, a)$
- $q_0^d = \{q_0\}$
- $F^d = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$
- Ambos autómatas son equivalentes, es decir, $L(M) = L(M^d)$.