No determinismo en AF's

- Hasta el momento, cada paso de computación nos lleva por un camino unívoco hacia otro estado.

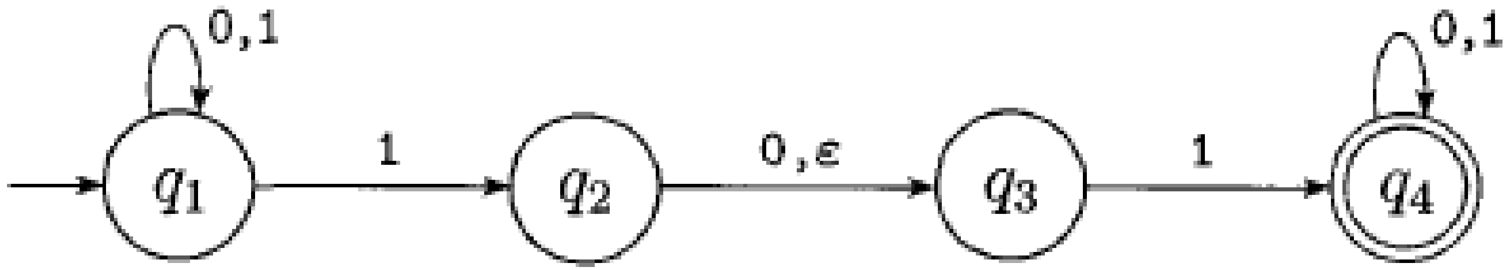
- Es decir, estando en un cierto estado y recibiendo una entrada determinada, podemos conocer exactamente en que estado estaremos posteriormente.

No determinismo en AF's

- En una máquina no determinística, pueden existir varios estados posteriores como resultado de una acción.

- El no determinismo es una generalización del determinismo, por lo cuál, todo AFD es un AFN.

AFN N1



AFN

- Supongamos que estamos ejecutando un AFN sobre una entrada y llegamos a un estado donde tenemos múltiples transiciones para una misma acción.

- Ejemplo: el estado q1 en el autómata N1, recibiendo la entrada 1.

AFN

- Luego de recibir la entrada, la máquina produce múltiples copias de sí misma (Una por cada posible transición).
- Posteriormente se siguen todas las posibilidades en paralelo, y cada una de las copias prosigue la ejecución.
- El proceso se repite recursivamente para cada una de las copias.

AFN

- Si para una entrada determinada, en alguna copia no existe una transición asociada, la misma muere.
- Por otra parte, si no existen más entradas y alguna de las copias se encuentra en estado de aceptación, el AFN acepta la secuencia.

AFN: transición epsilon

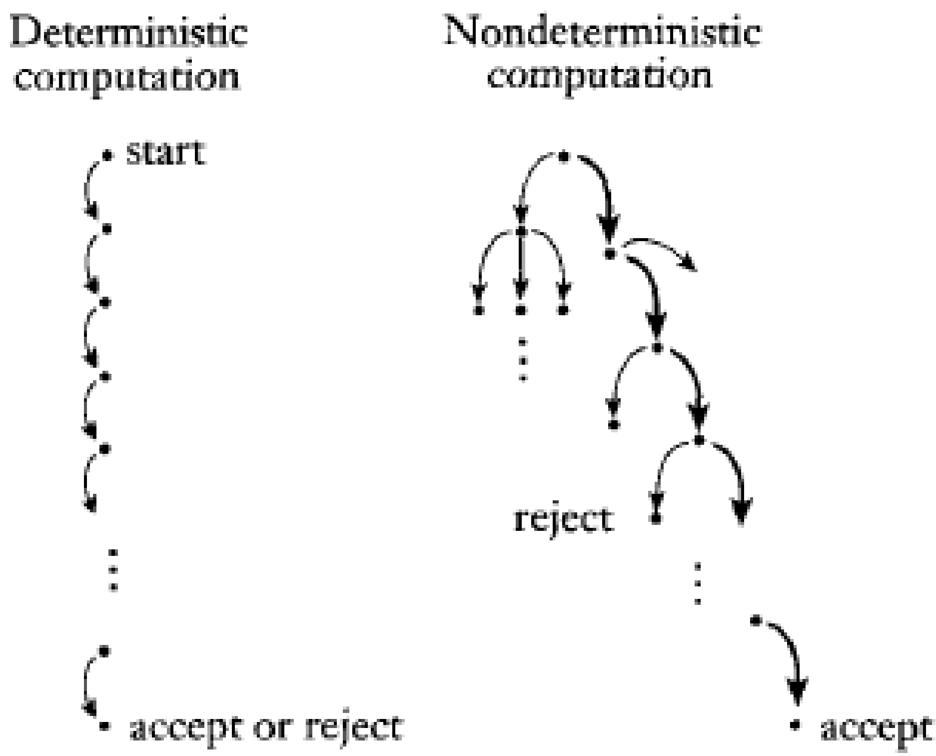
- Si en algún estado encontramos una transición ε ocurre un proceso similar, es decir, se producen tantas copias del AFN como transiciones existan.

- Una copia se mantendrá en el estado original, mientras que las otras asumirán el estado correspondiente a la transición

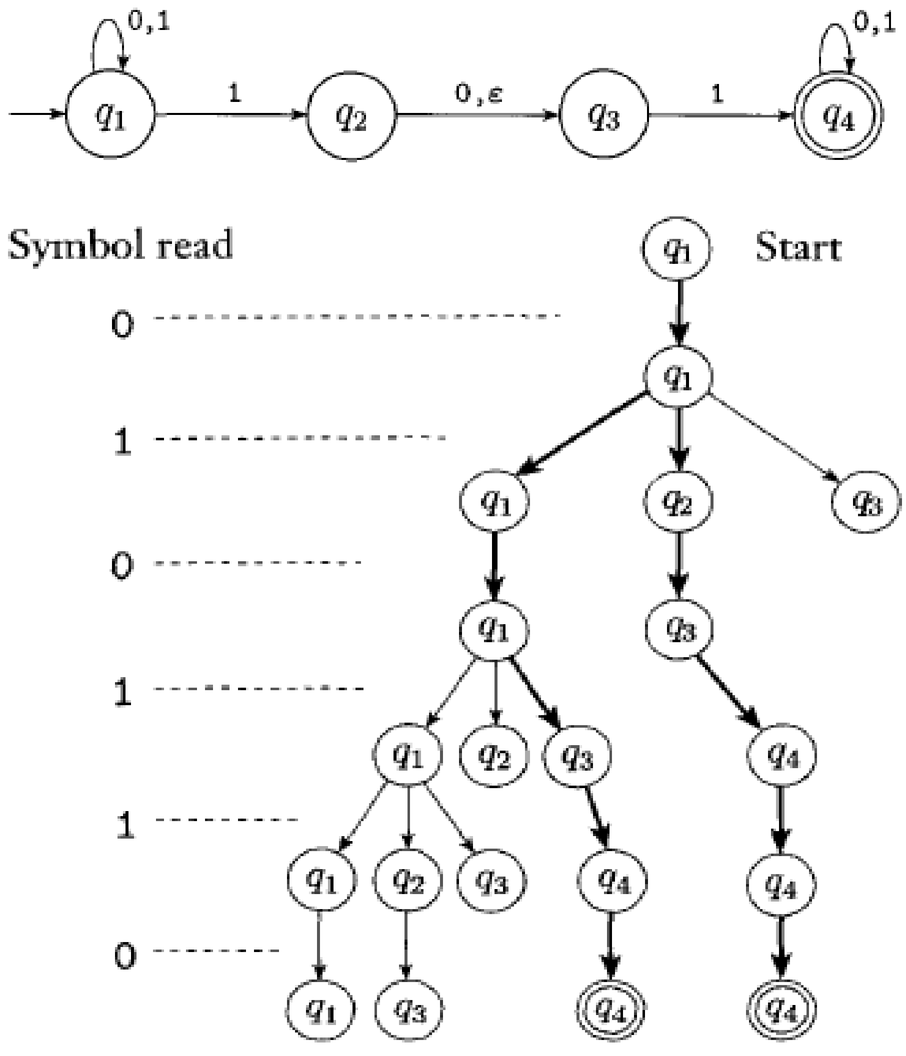
No Determinismo

- El no determinismo podría ser visto como una especie de computación en paralelo, donde varios procesos o threads pueden ejecutarse concurrentemente.

No determinismo



Ejemplo Autómata N1: 010110



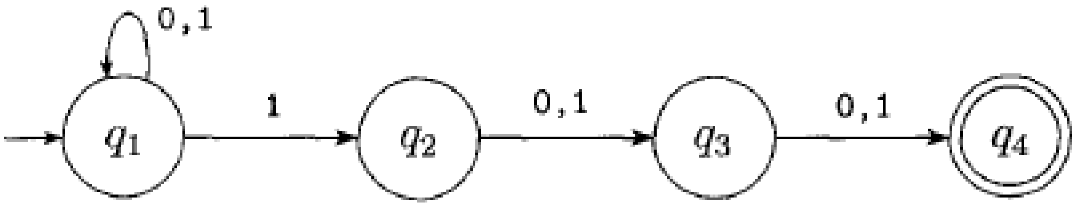
AFN

- Probar ejemplo 010

AFN

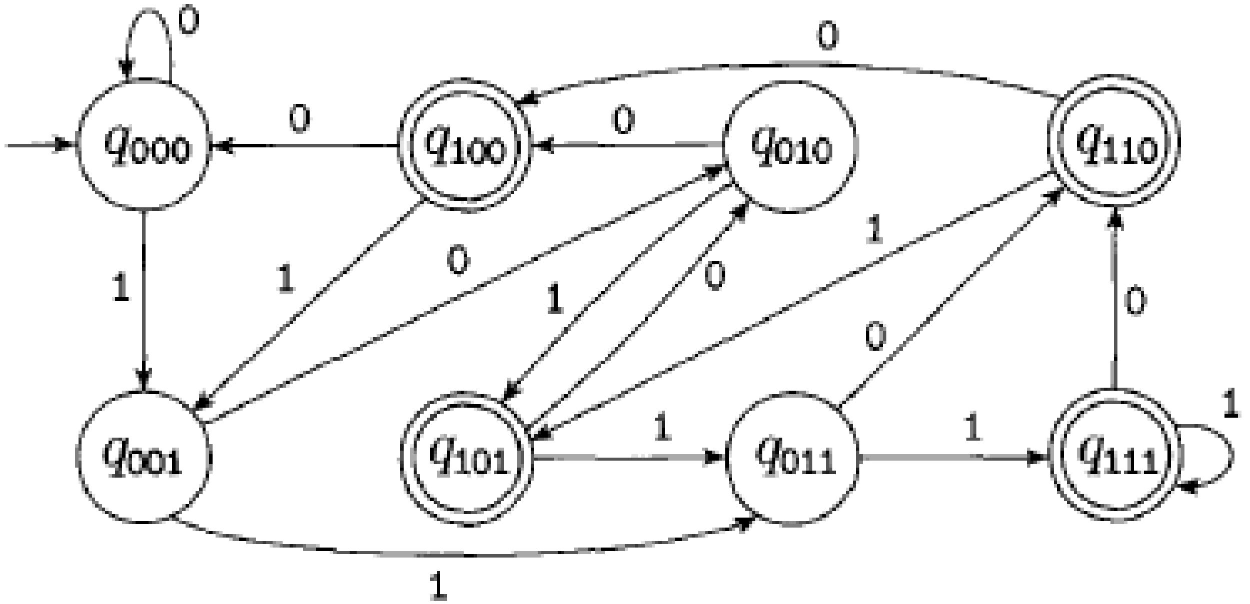
- Los AFN pueden ser convertidos en su AFD equivalente.
- Construir AFN es más fácil que construir AFD, debido a que son más pequeños y su funcionamiento es más claro.
- Constituyen una buena introducción a modelos computacionales más poderosos.

Ejemplo AFN/AFD



¿Qué lenguaje reconoce?

Equivalente AFD

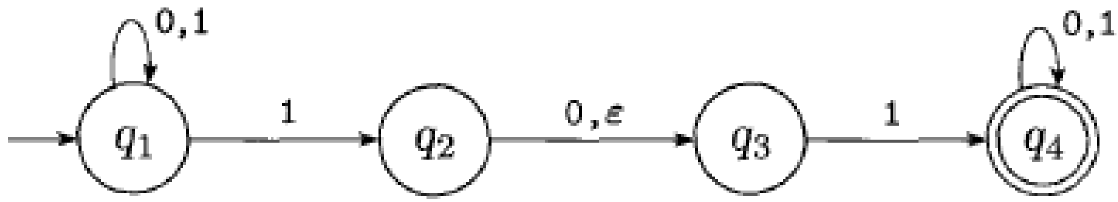


AFN: Definición

A ***nondeterministic finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ is the transition function,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

AFN: Definición



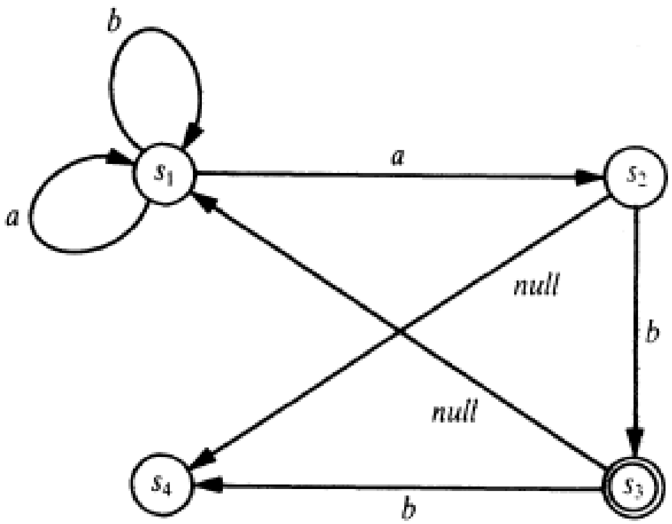
The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

4. q_1 is the start state, and
5. $F = \{q_4\}$.

Ejemplo AFN en Prolog (PL)

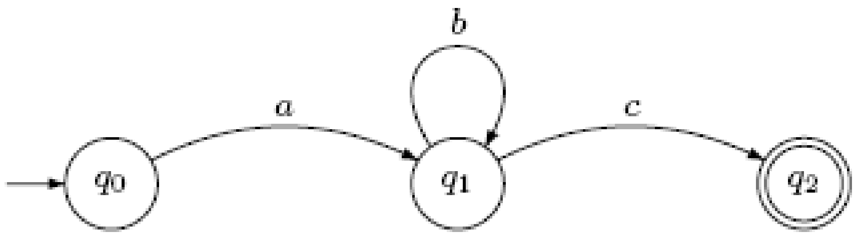



```
final( s3).

trans( s1, a, s1).
trans( s1, a, s2).
trans( s1, b, s1).
trans( s2, b, s3).
trans( s3, b, s4).

silent( s2, s4).
silent( s3, s1).

accepts( S, [] )  :-
  final( S).

accepts( S, [X | Rest] )  :-
  trans( S, X, S1),
  accepts( S1, Rest).

accepts( S, String)  :-
  silent( S, S1),
  accepts( S1, String).
```

```
?-  accepts( s1, [a,a,a,b] ).

yes

?-  accepts( S, [a,b] ).

S = s1;

S = s3
```

```
?-  String = [_, _, _], accepts( s1, String).

String = [a,a,b];

String = [b,a,b];

no
```

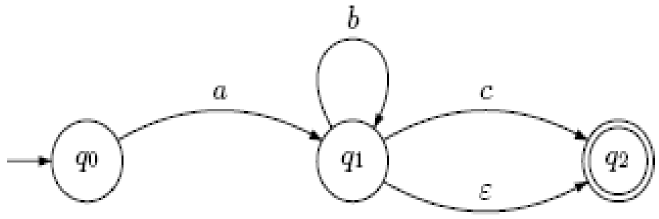
Ejemplo AF en Prolog II (PL)




```
% The start state
start(q0).

% The final states
final(q2).

% The transitions
% transition(SourceState, Symbol, DestinationState)
transition(q0, a, q1).
transition(q1, b, q1).
transition(q1, c, q2).
```

```
accept(Symbols) :-
  start(StartState),
  accept(Symbols, StartState).

% accept(+Symbols, +State)
accept([], State) :-
  final(State).
accept([Symbol | Symbols], State) :-
  transition(State, Symbol, NextState),
  accept(Symbols, NextState).
```

Ejemplo AFN en Prolog III (PL)




```
% The start state
start(q0).

% The final states
final(q2).

% The transitions
% transition(SourceState, Symbol, DestinationState)
transition(q0, a, q1).
transition(q1, b, q1).
transition(q1, c, q2).

epsilon(q1, q2).
```

```
accept([], State) :-
  final(State).
accept([Symbol | Symbols], State) :-
  transition(State, Symbol, NextState),
  accept(Symbols, NextState).
accept(Symbols, State) :-
  epsilon(State, NextState),
  accept(Symbols, NextState).
```

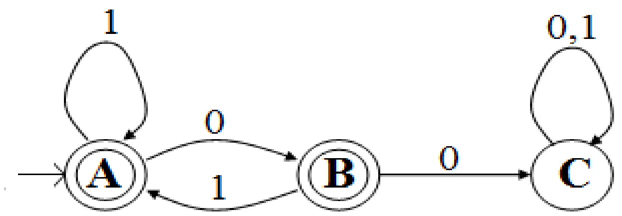
Operaciones entre autómatas

- Sean
  - $M1 = (K1, \Sigma, \delta 1, s1, F1)$ y
    $M2 = (K2, \Sigma, \delta 2, s2, F2)$ dos AFDs que aceptan los lenguajes L1 y L2, respectivamente.

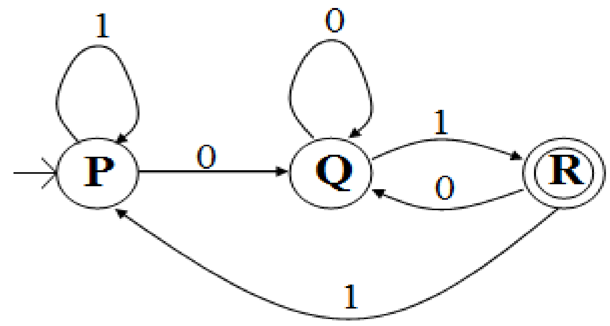
- Cómo obtener autómatas que reconozcan
  - $L1 \cup L2$
  - $L1 \cap L2$
  - $L1 - L2$

Respuesta

- Consideremos los lenguajes sobre el alfabeto $\Sigma=\{0, 1\}$:
  - $L1 = \{x \mid 00$ no es una subcadena de $x\}$
  - $L2 = \{x \mid x$ termina con $01\}$

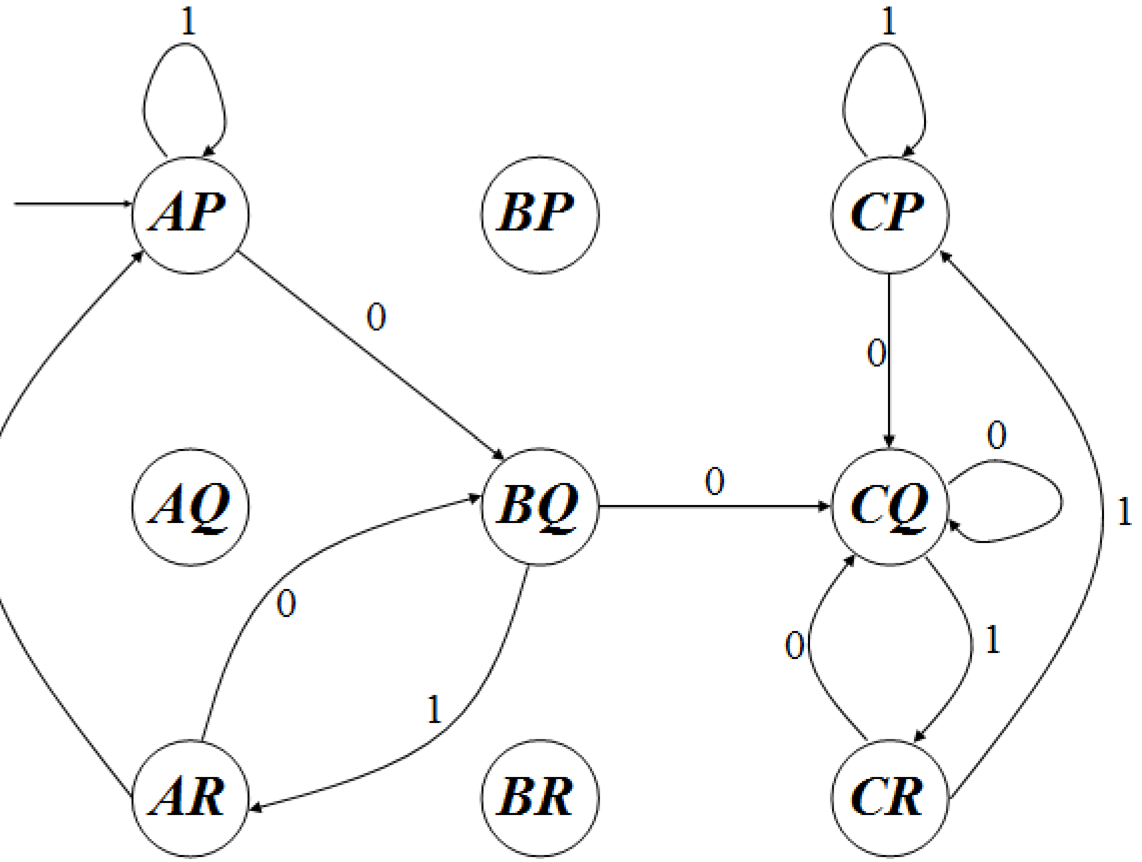


L_1

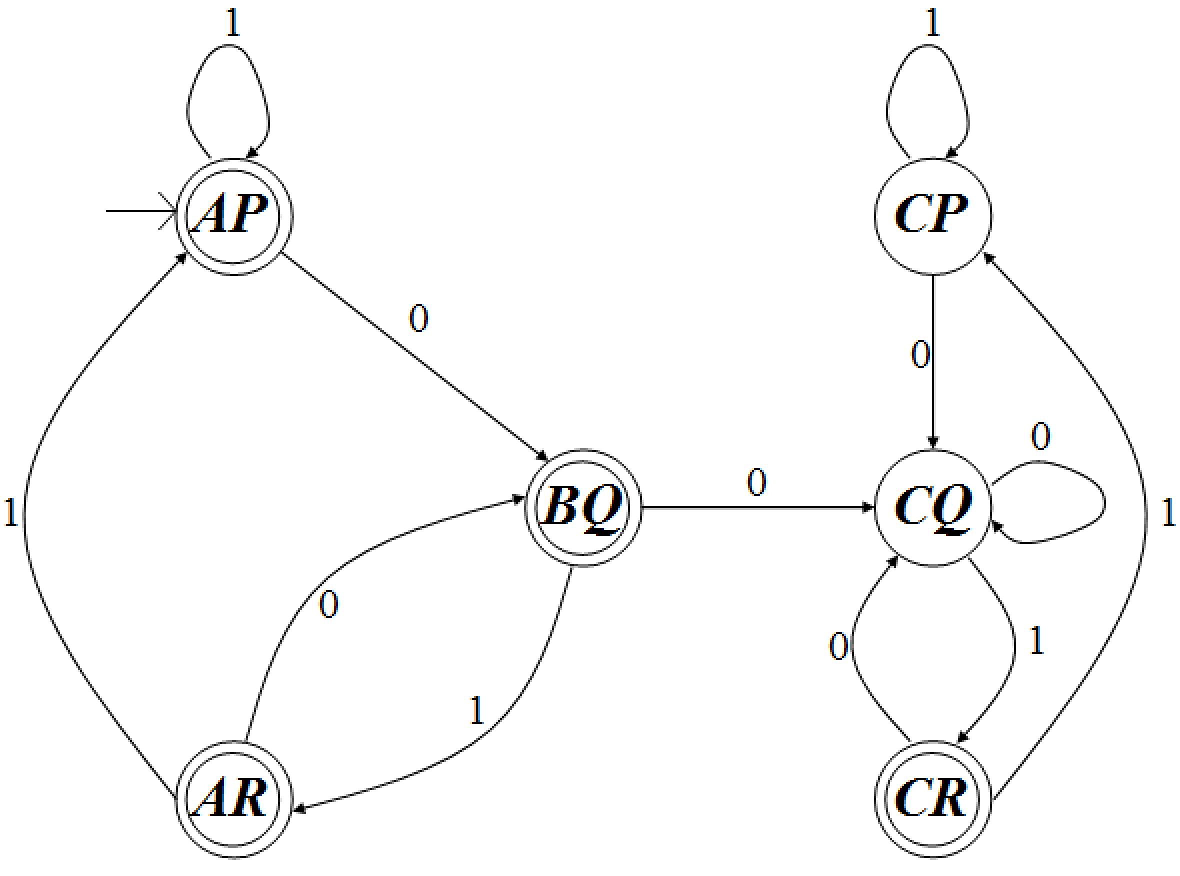


L_2

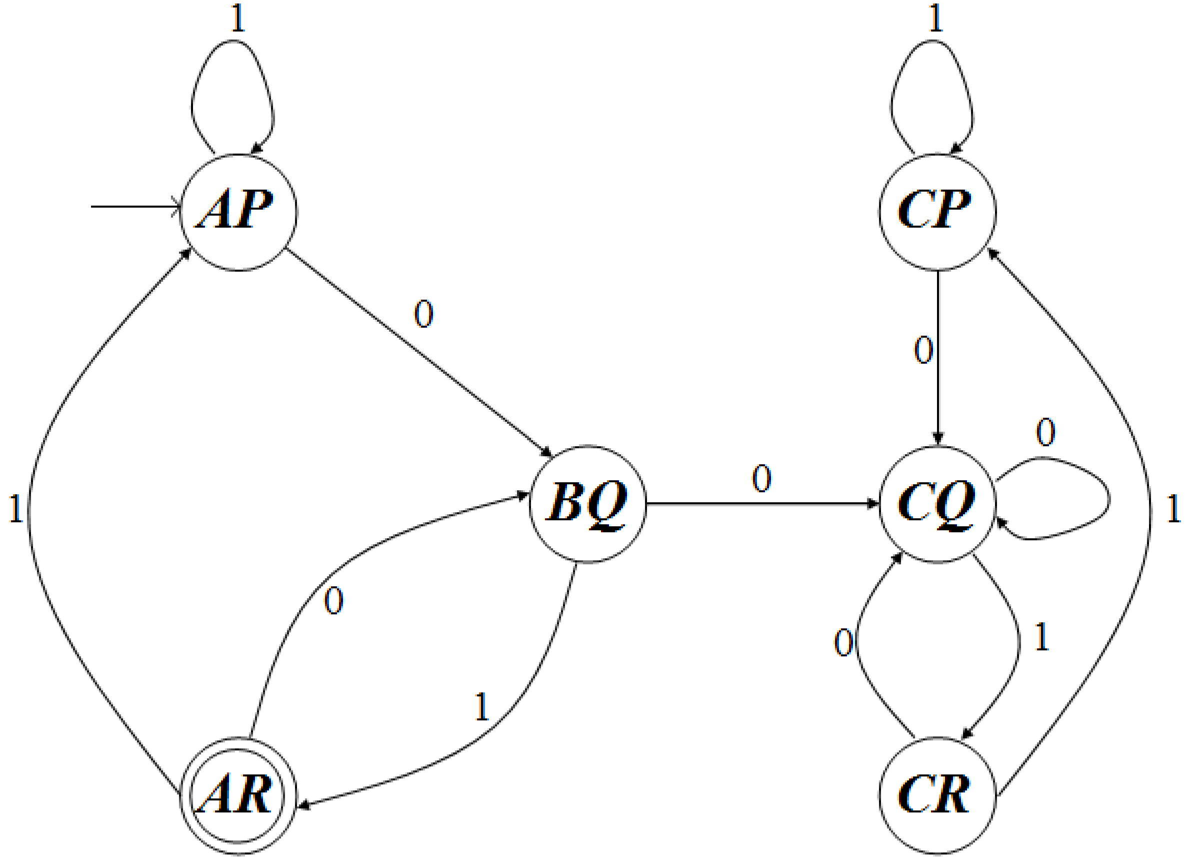
$K_1 \times K_2$



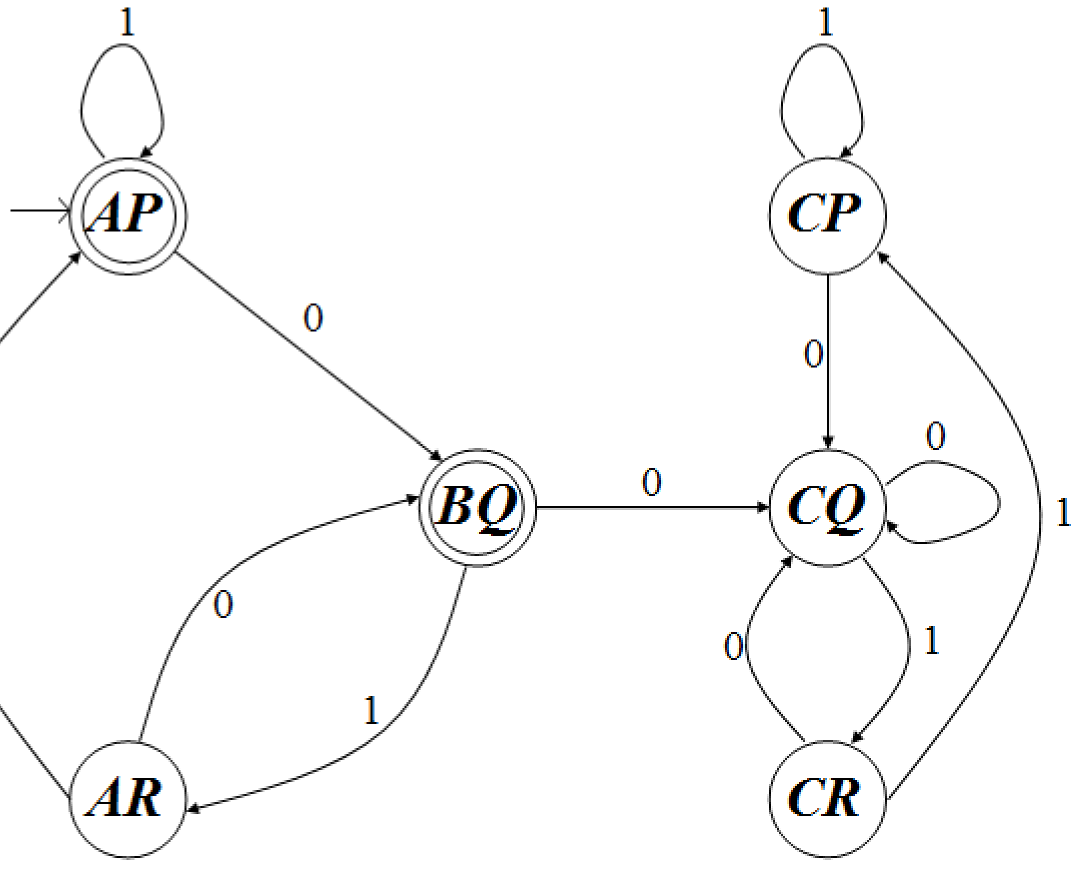
$L_1 \cup L_2$ (*Conector "o"*)



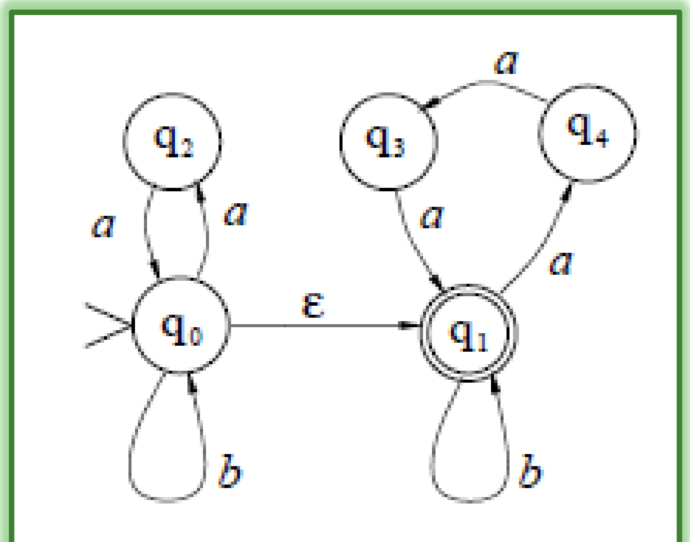
$L_1 \cap L_2$ (Conector "y ")



$L_1 - L_2$



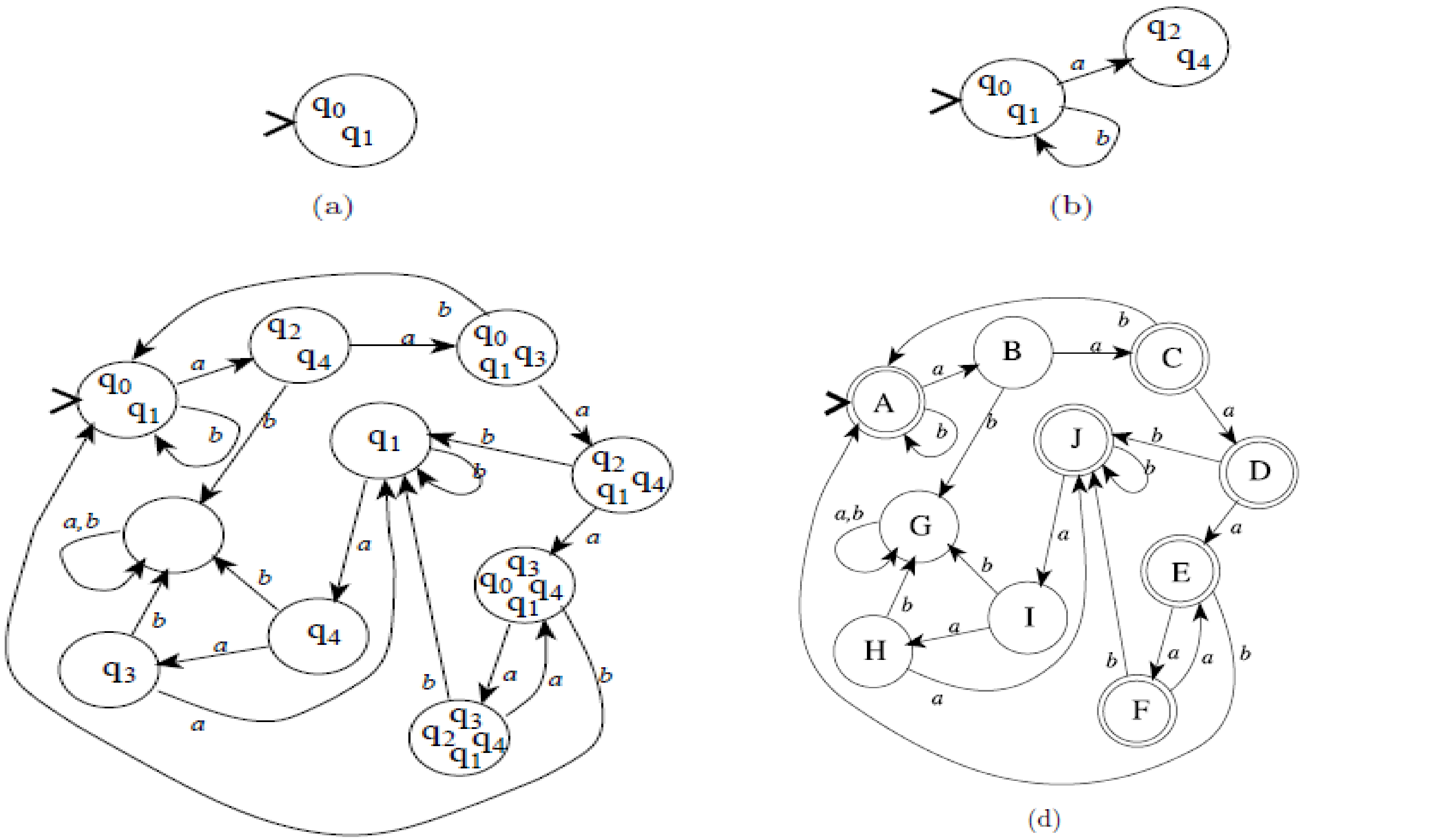
Convertir AFN en AFD



baaaaab

Entrada	*Estados*
	$\{q_0, q_1\}$
b	$\{q_0, q_1\}$
a	$\{q_2, q_4\}$
a	$\{q_0, q_1, q_3\}$
a	$\{q_1, q_2, q_4\}$
a	$\{q_0, q_1, q_3, q_4\}$
a	$\{q_1, q_2, q_3, q_4\}$
b	$\{q_1\}$

Convertir AFN en AFD



(a) (b) (c) (d)