THEORY OF AUTOMATA & FORMAL LANGUAGES

Conversion of ε-NFA to NFA

Removing ε-transitions without changing the language

Close → Move → Close → Verify

Why remove ε-transitions?

An equivalent NFA recognizes exactly the same language.

Original ε-NFA

Allows moves without reading input.

May reach other states before a symbol.

May also move after the last symbol.

Accepts if any complete path ends in F.

We must preserve all these possibilities.

Equivalent ordinary NFA

Every edge consumes one input symbol.

Keep Q, Σ and the initial state.

Recalculate every symbol transition.

Recalculate the accepting state set.

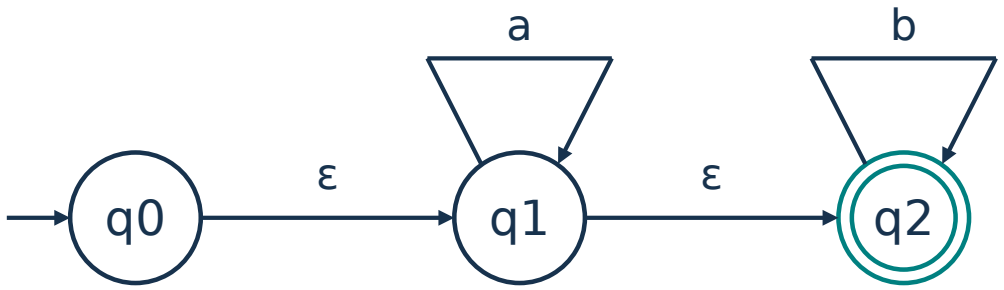
No ε-column in the new table.

This is not determinization: multiple destinations are still allowed.

Worked example: original ε-NFA

Alphabet $\Sigma = \{a,b\}$; initial q_0; accepting set $F = \{q_2\}$.

Original state diagram



Original transition table

State	a	b	ε
q_0	∅	∅	$\{q_1\}$
q_1	$\{q_1\}$	∅	$\{q_2\}$
q_2	∅	$\{q_2\}$	∅

Language: all a’s followed by b’s.

The original machine recognizes a*b*, including the empty string.

Step 1 — Calculate ε-closures

E(q) includes q and all states reachable by ε alone.

Follow the ε-edges

$q_0 \to q_1 \to q_2$ using ε.

$E(q_0) = \{q_0, q_1, q_2\}$

$q_1 \to q_2$ using ε.

$E(q_1) = \{q_1, q_2\}$

$E(q_2) = \{q_2\}$

Why closures matter

No input is consumed in these moves.

A state is always in its own closure.

Follow chains, not just one edge.

For a set, take the union of closures.

$E(\varnothing) = \varnothing$.

The closure captures every possible position before reading a symbol.

Step 2 — Recalculate transitions

$\delta'(q,x) = E(\text{move}(E(q),x))$ for $x \in \Sigma$.

Example: $\delta'(q_0,a)$

First close: $\{q_0,q_1,q_2\}$.

Read a from every state in this set.

Only q_1 has an a-edge, to q_1.

Move result = $\{q_1\}$.

Close again: $\{q_1,q_2\}$.

Example: $\delta'(q_0,b)$

First close: $\{q_0,q_1,q_2\}$.

Read b from every state in this set.

Only q_2 has a b-edge, to q_2.

Move result = $\{q_2\}$.

Close again: $\{q_2\}$.

A new transition summarizes ε-moves before and after one consumed symbol.

Step 3 — Complete the new table

Apply the same rule for every state and every alphabet symbol.

Remaining calculations

$\delta'(q_1,a) = E(\{q_1\}) = \{q_1,q_2\}$

$\delta'(q_1,b) = E(\{q_2\}) = \{q_2\}$

$\delta'(q_2,a) = E(\emptyset) = \emptyset$

$\delta'(q_2,b) = E(\{q_2\}) = \{q_2\}$

All six entries are now computed.

New NFA transition table

State	a	b
q_0	$\{q_1,q_2\}$	$\{q_2\}$
q_1	$\{q_1,q_2\}$	$\{q_2\}$
q_2	$\emptyset$	$\{q_2\}$

There is no ε-column.

Multiple target states are permitted because the result is an NFA.

Step 4 — Update accepting states

$F' = \{q \in Q : E(q) \cap F \neq \emptyset\}$.

Check each original state

Original $F = \{q_2\}$.

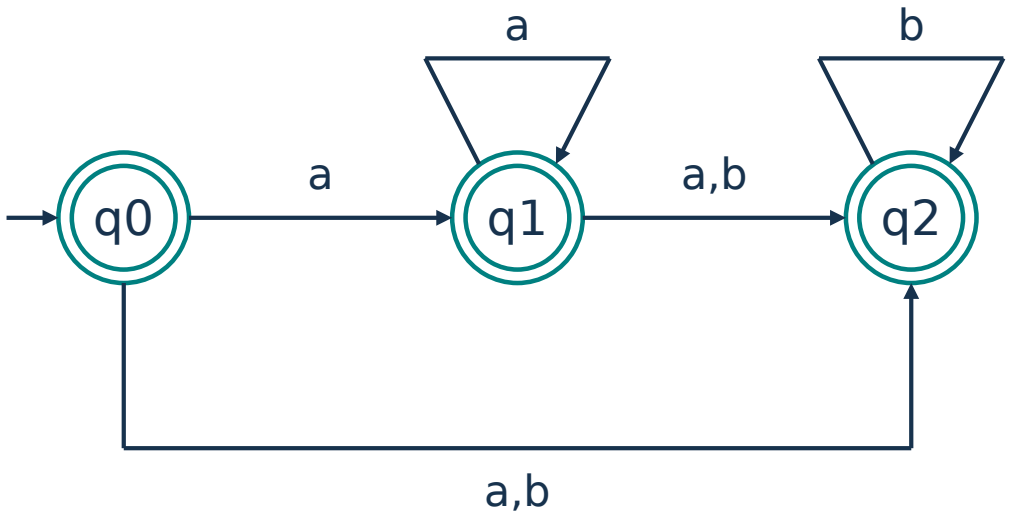
$E(q_0)$ contains q_2 → accepting.

$E(q_1)$ contains q_2 → accepting.

$E(q_2)$ contains q_2 → accepting.

New $F' = \{q_0, q_1, q_2\}$.

Resulting ordinary NFA



The initial state becomes accepting because ε was accepted originally.

Step 5 — Verify language equivalence

Both machines recognize a*b*: zero or more a’s followed by zero or more b’s.

Accepted example: aab

Start new NFA at $\{q_0\}$.

Read a: $\{q_1, q_2\}$.

Read a: $\{q_1, q_2\}$.

Read b: $\{q_2\}$.

$q_2 \in F' \rightarrow$ accept.

Rejected example: ba

Start new NFA at $\{q_0\}$.

Read b: $\{q_2\}$.

Read a: $\emptyset$.

No state remains.

$\emptyset \cap F' = \emptyset \rightarrow$ reject.

Also test ε: $q_0 \in F'$, so both machines accept the empty string.

Practice and summary

A reliable conversion procedure for any ε-NFA.

Practice questions

1. Must Q change during conversion?

2. Why close both before and after a move?

3. Can an initial state become accepting?

4. Does the example accept abb?

5. Does the example accept aba?

Explained answers

1. No; keep the original state set.

2. To preserve every possible ε-path.

3. Yes, if its closure intersects F.

4. Yes: $\{q_0\} \to \{q_1,q_2\} \to \{q_2\} \to \{q_2\}$.

5. No: the final a has no destination.

Next: convert an ordinary NFA into an equivalent DFA using subset construction.