COMP-330
Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

Lec. 7 : Regular Expressions & GNFA

Regarding HW-1

1.46 Prove that the following languages are not regular. You may use the MYHILL-NERODE THEOREM and the closure of the class of regular languages under union, intersection, and complement.

a. $\{0^n 1^m 0^n \mid m, n \geq 0\}$

A**b.** $\{0^m 1^n \mid m \neq n\}$

Regular Expressions

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions,

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions,
5. $R_1 \circ R_2$, where R_1 and R_2 are regular expressions, or

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions,
5. $R_1 \circ R_2$, where R_1 and R_2 are regular expressions, or
6. R_1^* , where R_1 is a regular expression.

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions,
5. $R_1 \circ R_2$, where R_1 and R_2 are regular expressions, or
6. R_1^* , where R_1 is a regular expression.

$R_1^+ = R_1 \circ R_1^*$

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions,
5. $R_1 \circ R_2$, where R_1 and R_2 are regular expressions, or
6. R_1^* , where R_1 is a regular expression.

$R_1^+ = R_1 \circ R_1^*$

In items 1 and 2, the regular expressions a and ε represent the

Regular Expressions

DEFINITION 1.52

Say that R is a ***regular expression*** if R is

1. a for some a in the alphabet Σ,
2. ε,
3. $\emptyset$,
4. $R_1 \cup R_2$, where R_1 and R_2 are regular expressions,
5. $R_1 \circ R_2$, where R_1 and R_2 are regular expressions, or
6. R_1^* , where R_1 is a regular expression.

$R_1^+ = R_1 \circ R_1^*$

In items 1 and 2, the regular expressions a and ε represent the languages $\{a\}$ and $\{\varepsilon\}$, respectively. In item 3, the regular expression $\emptyset$ represents the empty language. In items 4, 5, and 6, the expressions represent the languages obtained by taking the union or concatenation of the languages R_1 and R_2, or the star of the language R_1, respectively.

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

À

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.

À

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w$ contains a single $1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w$ has at least one $1\}$.

À

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w|\ w$ contains a single $1\}$.
2. $\Sigma^*1\Sigma^* = \{w|\ w$ has at least one $1\}$.
3. $\Sigma^*001\Sigma^* = \{w|\ w$ contains the string 001 as a substring$\}$.

À

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w$ contains a single $1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w$ has at least one $1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w$ contains the string 001 as a substring$\}$.
4. $1^*(01^+)^* = \{w \mid$ every 0 in w is followed by at least one $1\}$.

À

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]

À

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w$ contains a single $1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w$ has at least one $1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w$ contains the string 001 as a substring$\}$.
4. $1^*(01^+)^* = \{w \mid$ every 0 in w is followed by at least one $1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w$ is a string of even length$\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid$ the length of w is a multiple of three$\}$.

À

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$.

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$.
7. $01 \cup 10 = \{01, 10\}$.

EXAMPLE **1.53**

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
   The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
   The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.

EXAMPLE 1.53

In the following instances we assume that the alphabet Σ is $\{0,1\}$.

1. $0^*10^* = \{w \mid w \text{ contains a single } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ has at least one } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$.
4. $1^*(01^+)^* = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$.
5. $(\Sigma\Sigma)^* = \{w \mid w \text{ is a string of even length}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{the length of } w \text{ is a multiple of three}\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol}\}$.
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
   The expression $0 \cup \varepsilon$ describes the language $\{0, \varepsilon\}$, so the concatenation operation adds either 0 or ε before every string in 1^*.
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
11. $1^*\emptyset = \emptyset$.
    Concatenating the empty set to any set yields the empty set.
12. $\emptyset^* = \{\varepsilon\}$.
    The star operation puts together any number of strings from the language to get a string in the result. If the language is empty, the star operation can put together 0 strings, giving only the empty string.

Regular Expressions vs Regular Languages

THEOREM 1.54

A language is regular if and only if some regular expression describes it.

LEMMA 1.55

If a language is described by a regular expression, then it is regular.

LEMMA 1.60

If a language is regular, then it is described by a regular expression.

Regular Expressions vs Regular Languages

LEMMA 1.55

If a language is described by a regular expression, then it is regular.

LEMMA 1.60

If a language is regular, then it is described by a regular expression.

Regular Expressions vs Regular Languages

LEMMA 1.55

If a language is described by a regular expression, then it is regular.

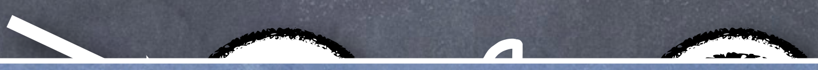
LEMMA 1.60

If a language is regular, then it is described by a regular expression.

Regular Expressions generate Reg. Languages

Regular Expressions generate Reg. Languages

- Let R_A generating L_A.



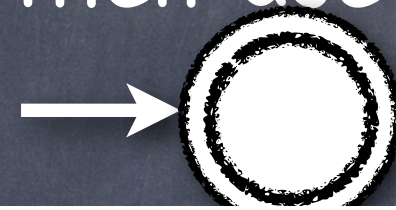
Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

.

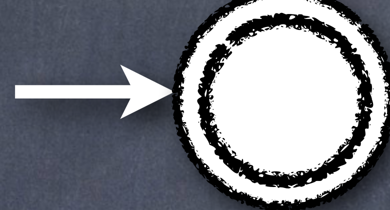
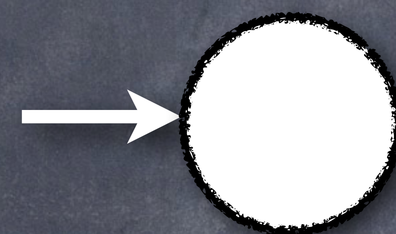
Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use  .

- If R_A is "ε" then use  .

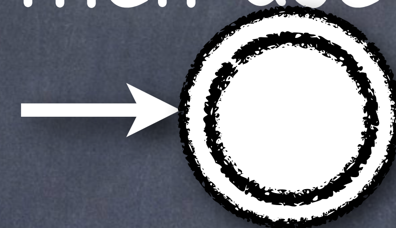
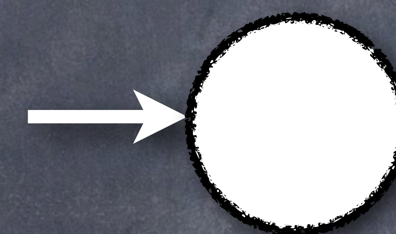
Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

.
- If R_A is "ε" then use  .
- If R_A is "∅" then use  .

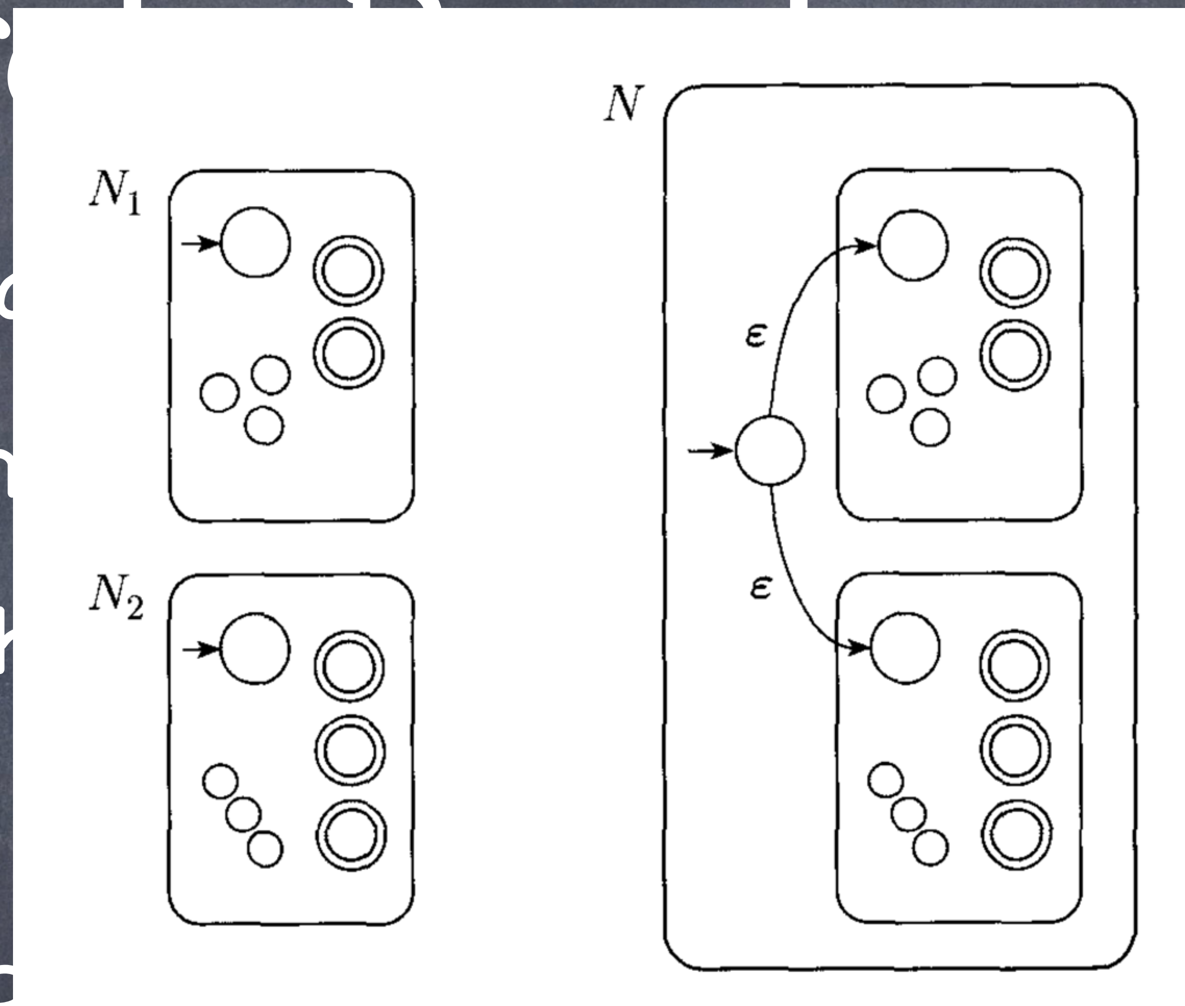
Regular Expressions generate Reg. Languages

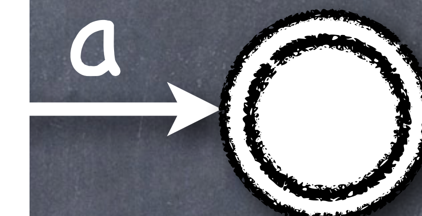
- Let R_A generating L_A.
- If R_A is a symbol "a" then use .

- If R_A is "ε" then use .
- If R_A is "∅" then use .
- If R_A is $R_1 \cup R_2$ then use Thm 1.45 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

Regular Expressions gener… uages

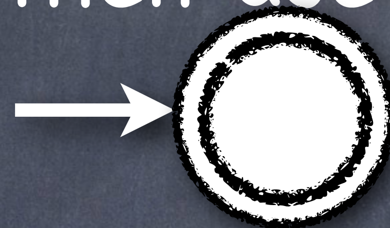
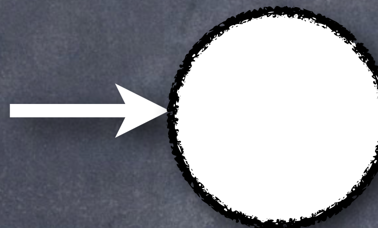
- Let R_A gener…
- If R_A is a sym… .
- If R_A is "ε" th…
- If … .
- If R_A is $R_1 \cup R$… ecursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.





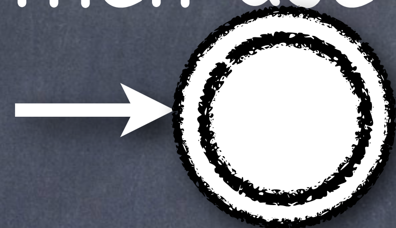
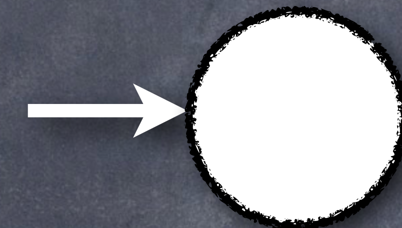
Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

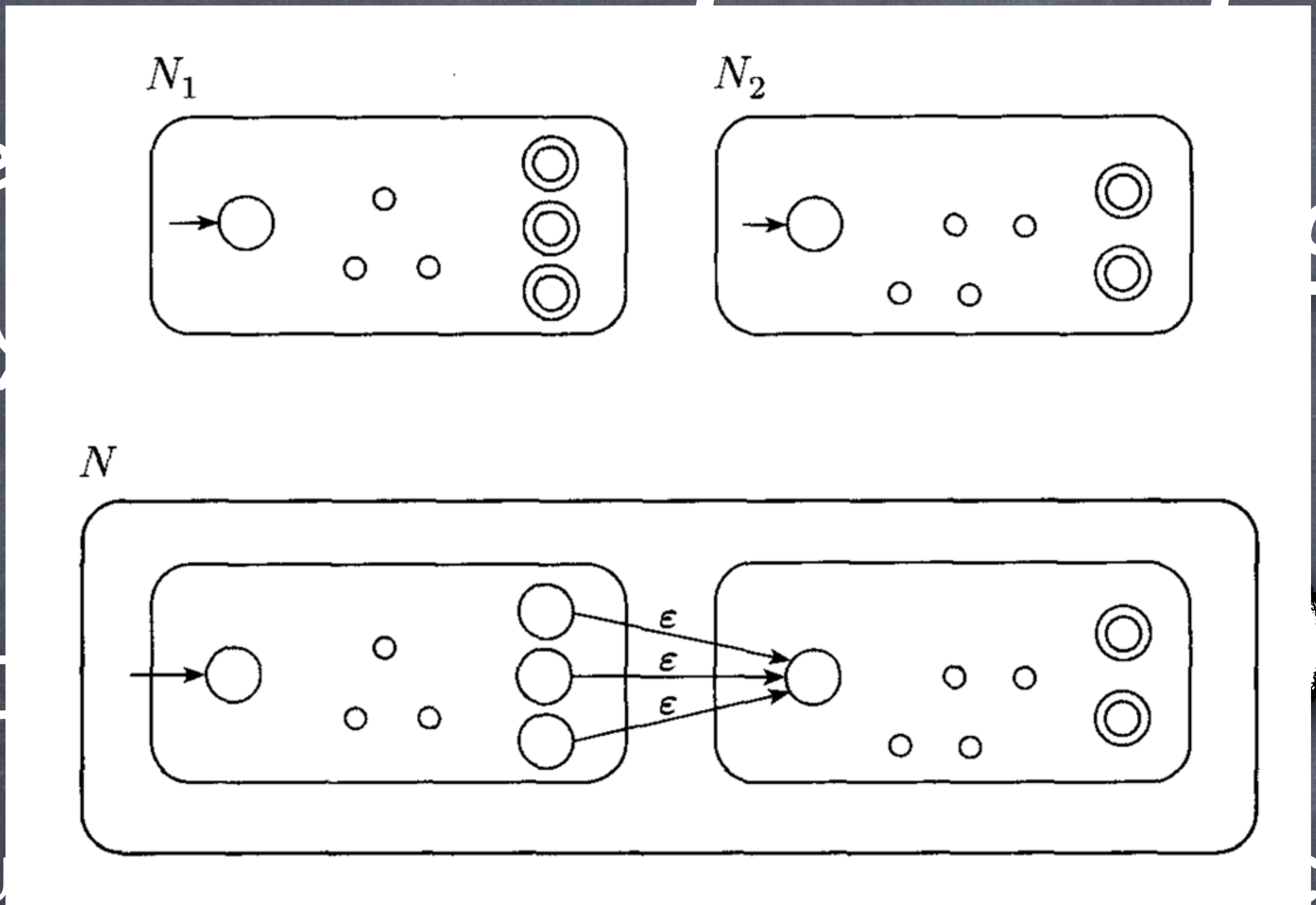
.
- If R_A is "ε" then use .
- If R_A is "∅" then use .
- If R_A is $R_1 \cup R_2$ then use Thm 1.45 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

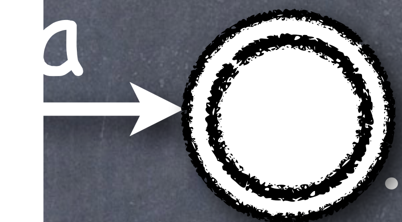
Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

.
- If R_A is "ε" then use  .
- If R_A is "∅" then use  .
- If R_A is $R_1 \cup R_2$ then use Thm 1.45 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is $R_1 \circ R_2$ then use Thm 1.47 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

Regular Expressions generate Reg. Languages

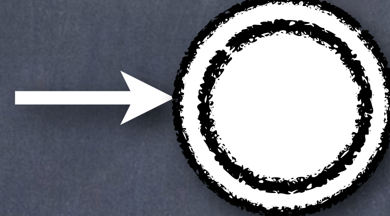
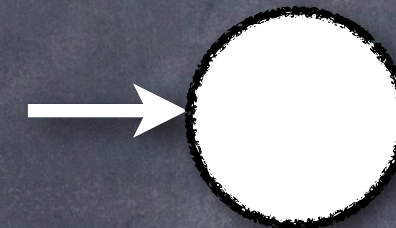




- Let R_A gene
- If R_A is a sy .
- If R_A is "ε"
- I .
- If R_A is R_1 ∪ cursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is $R_1 \circ R_2$ then use Thm 1.47 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

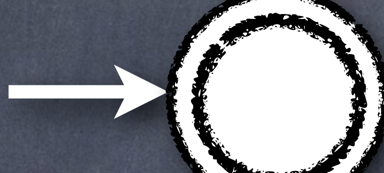
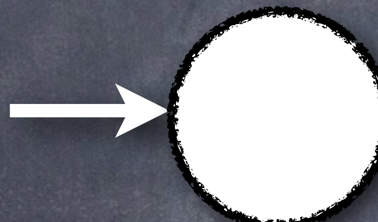
Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

.
- If R_A is "ε" then use  .
- If R_A is "∅" then use  .
- If R_A is $R_1 \cup R_2$ then use Thm 1.45 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is $R_1 \circ R_2$ then use Thm 1.47 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.

Regular Expressions generate Reg. Languages

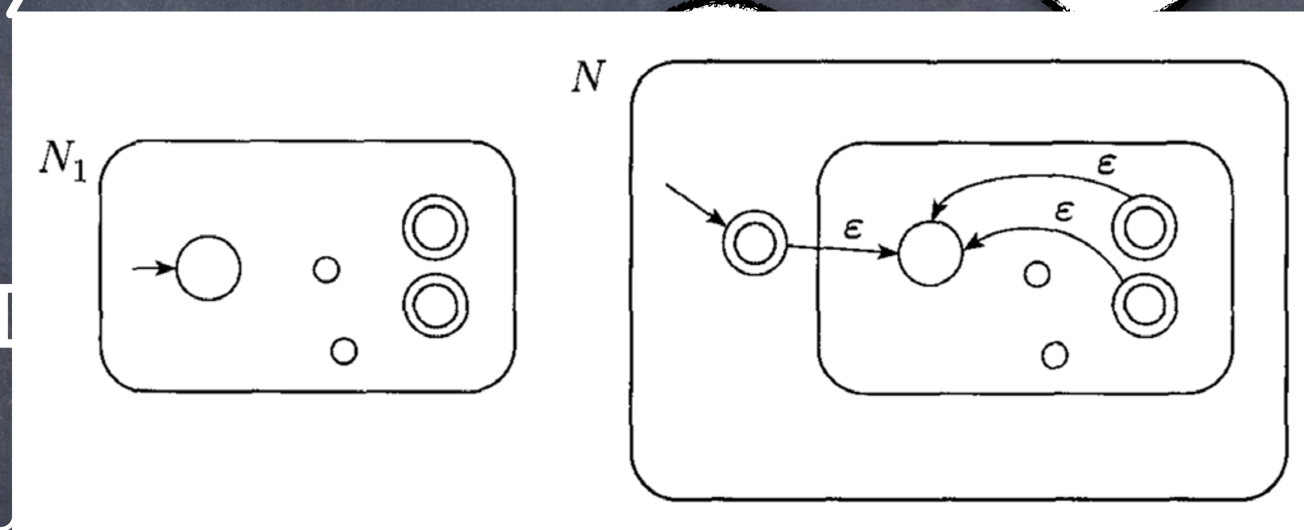
- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

.
- If R_A is "ε" then use  .
- If R_A is "∅" then use  .
- If R_A is $R_1 \cup R_2$ then use Thm 1.45 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is $R_1 \circ R_2$ then use Thm 1.47 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is R^* then use Thm 1.49 and recursively use N s.t. $L(N)=L(R)$.

Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

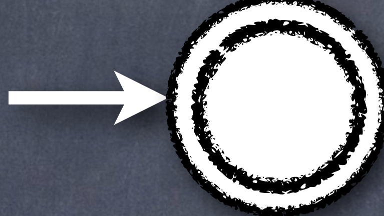
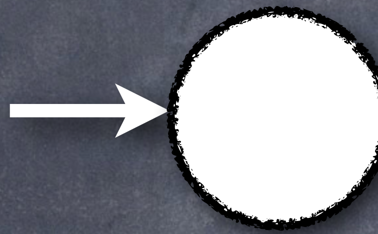
.
- If R_A is "ε"



- I .
- If R_A is R_1 ∪ ... cursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is $R_1 \circ R_2$ then use Thm 1.47 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is R^* then use Thm 1.49 and recursively use N s.t. $L(N)=L(R)$.

Regular Expressions generate Reg. Languages

- Let R_A generating L_A.
- If R_A is a symbol "a" then use 

.
- If R_A is "ε" then use  .
- If R_A is "∅" then use  .
- If R_A is $R_1 \cup R_2$ then use Thm 1.45 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is $R_1 \circ R_2$ then use Thm 1.47 and recursively use N_1 and N_2 s.t. $L(N_1) = L(R_1)$ and $L(N_2) = L(R_2)$.
- If R_A is R^* then use Thm 1.49 and recursively use N s.t. $L(N)=L(R)$.

Reg. Expression → NFA

- Two examples

$(ab \cup a)^*$

$(a \cup b)^*aba$

FIGURE 1.57

Building an NFA from the regular expression $(ab \cup a)^*$

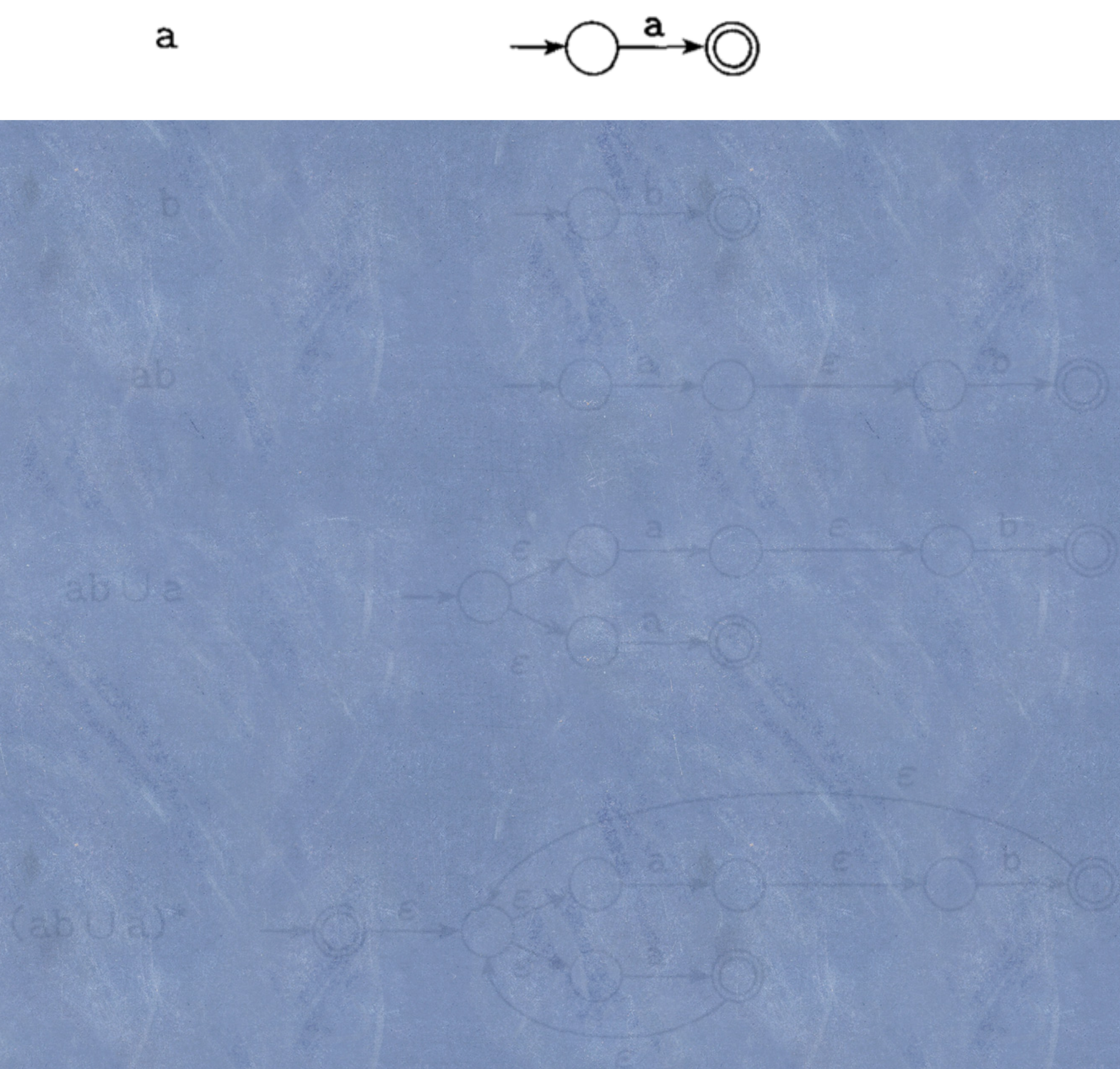
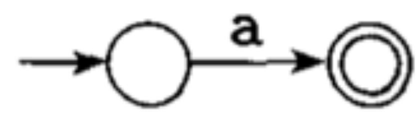


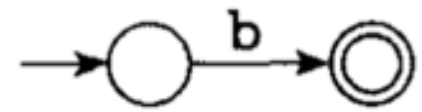
FIGURE 1.57
Building an NFA from the regular expression

(ab∪a)*

a



b



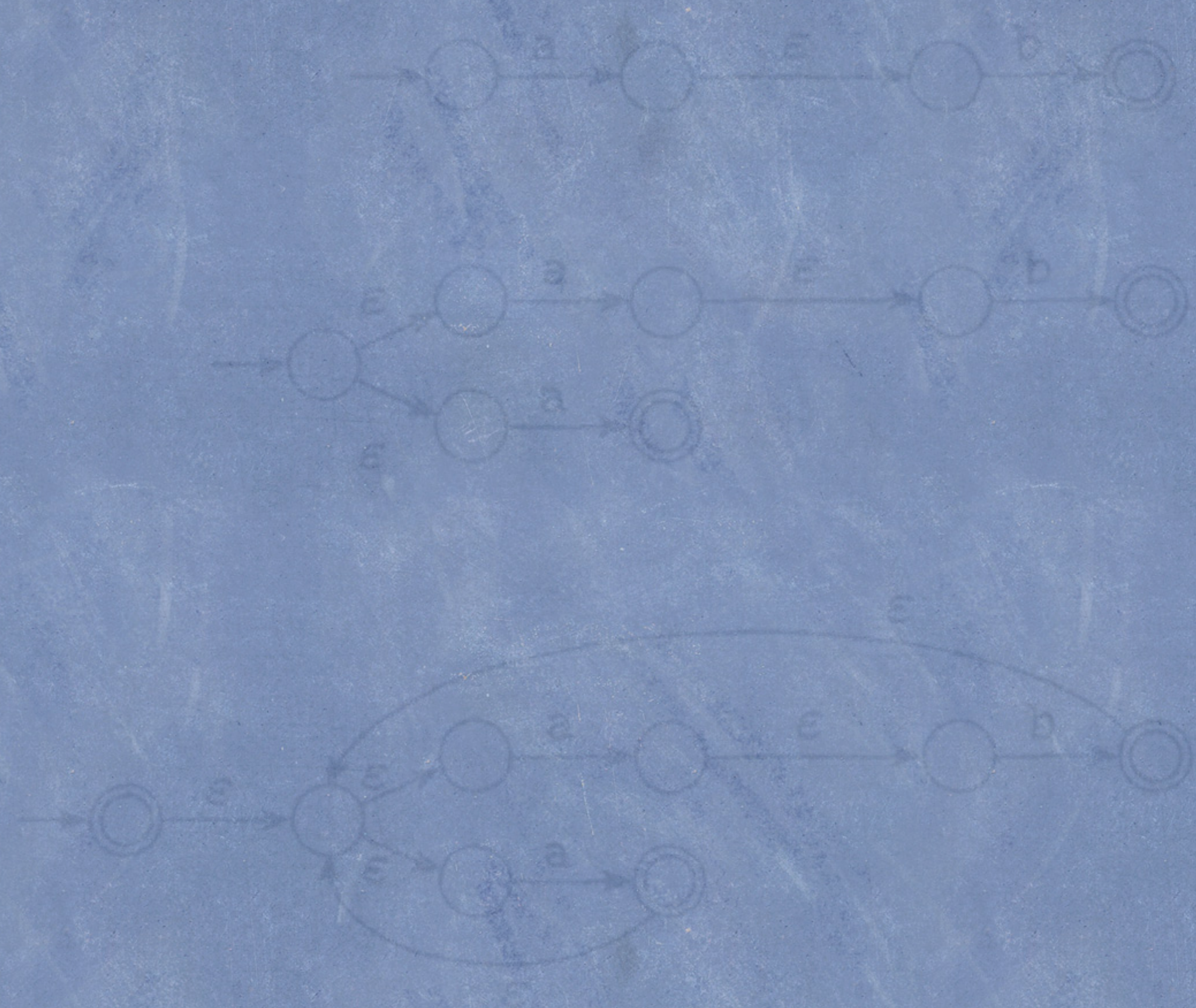


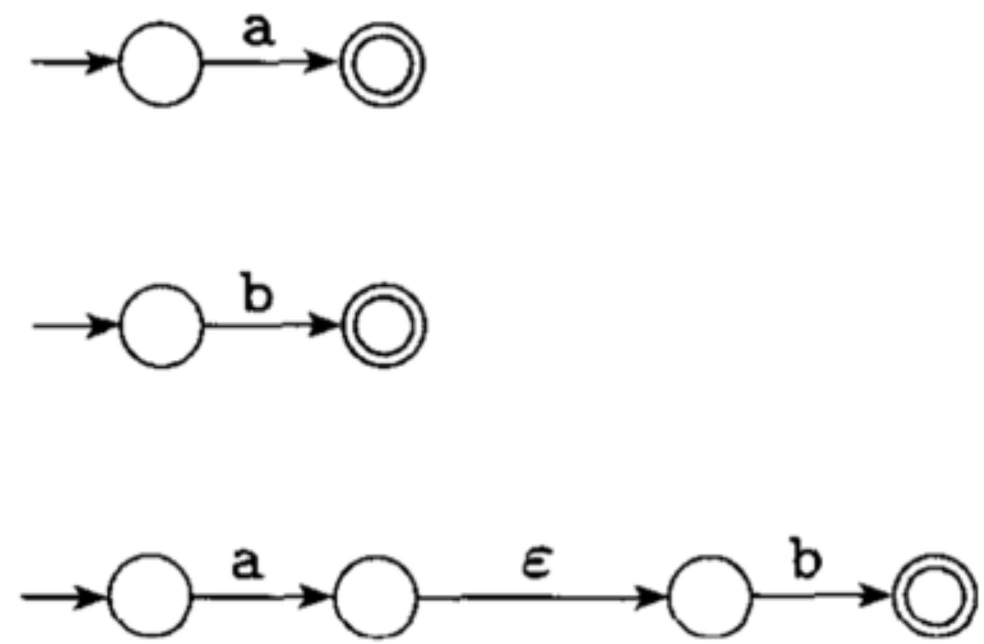
FIGURE 1.57
Building an NFA from the regular expression

(ab∪a)*

a

b

ab



ab∪a

(ab∪a)*

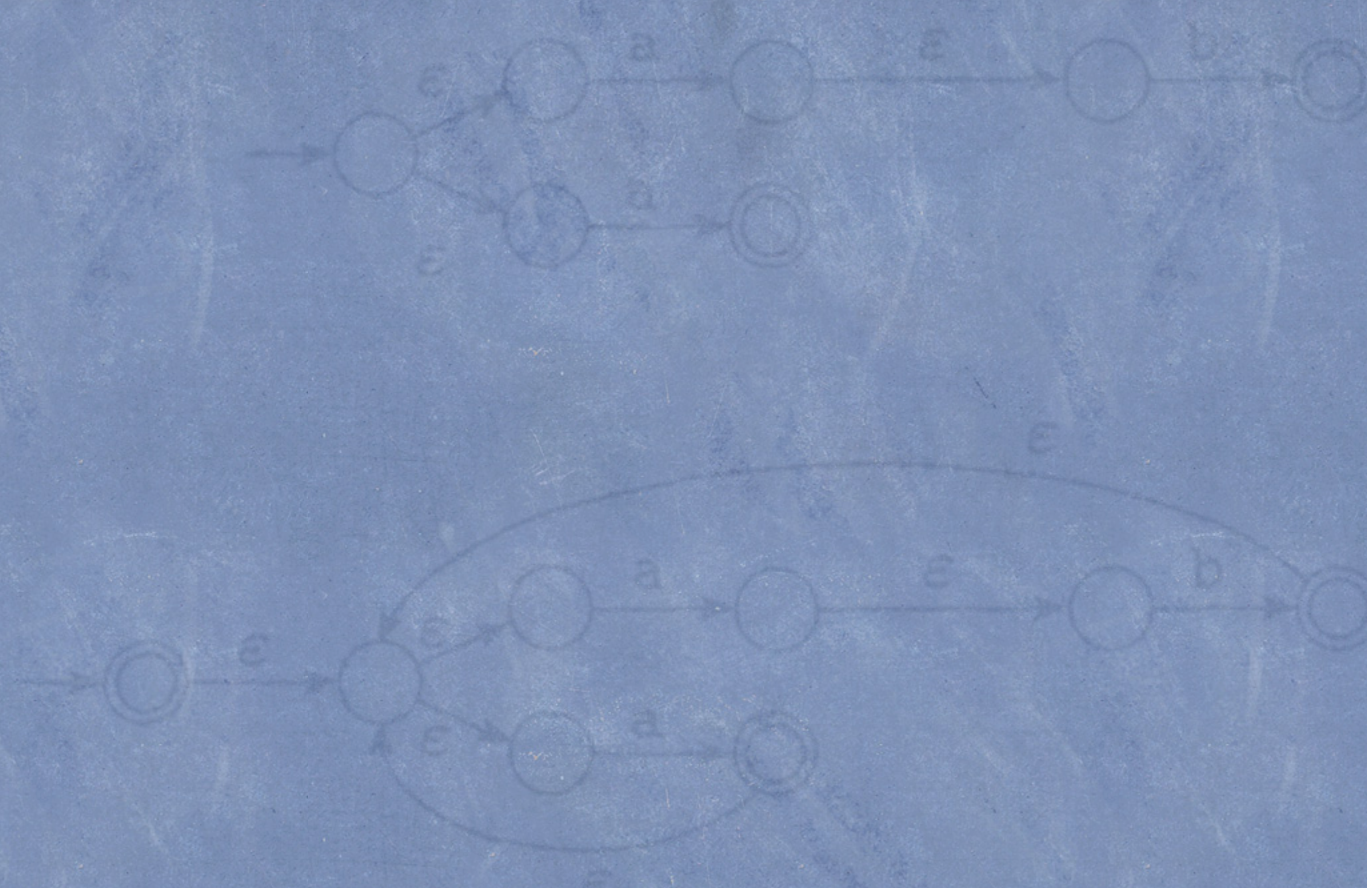
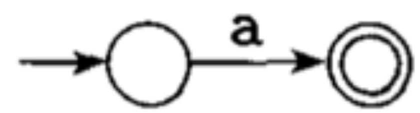


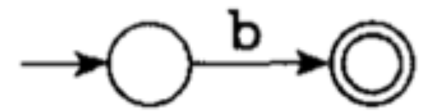
FIGURE 1.57
Building an NFA from the regular expression

(ab∪a)*

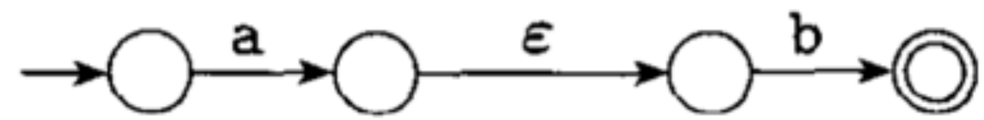
a



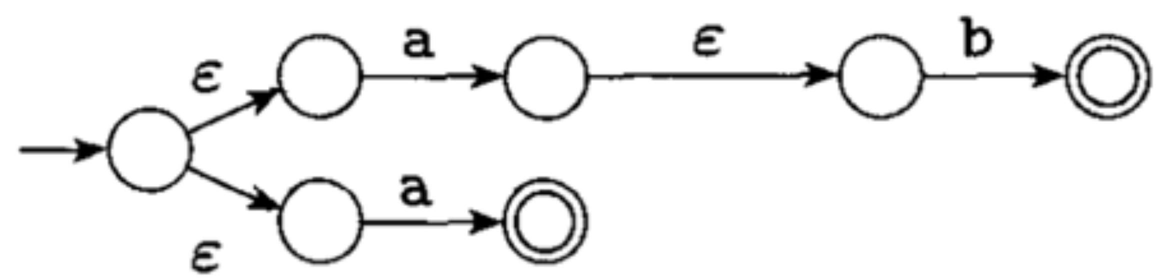
b



ab



ab ∪ a



(ab∪a)*

FIGURE 1.57
Building an NFA from the regular expression

(ab∪a)*

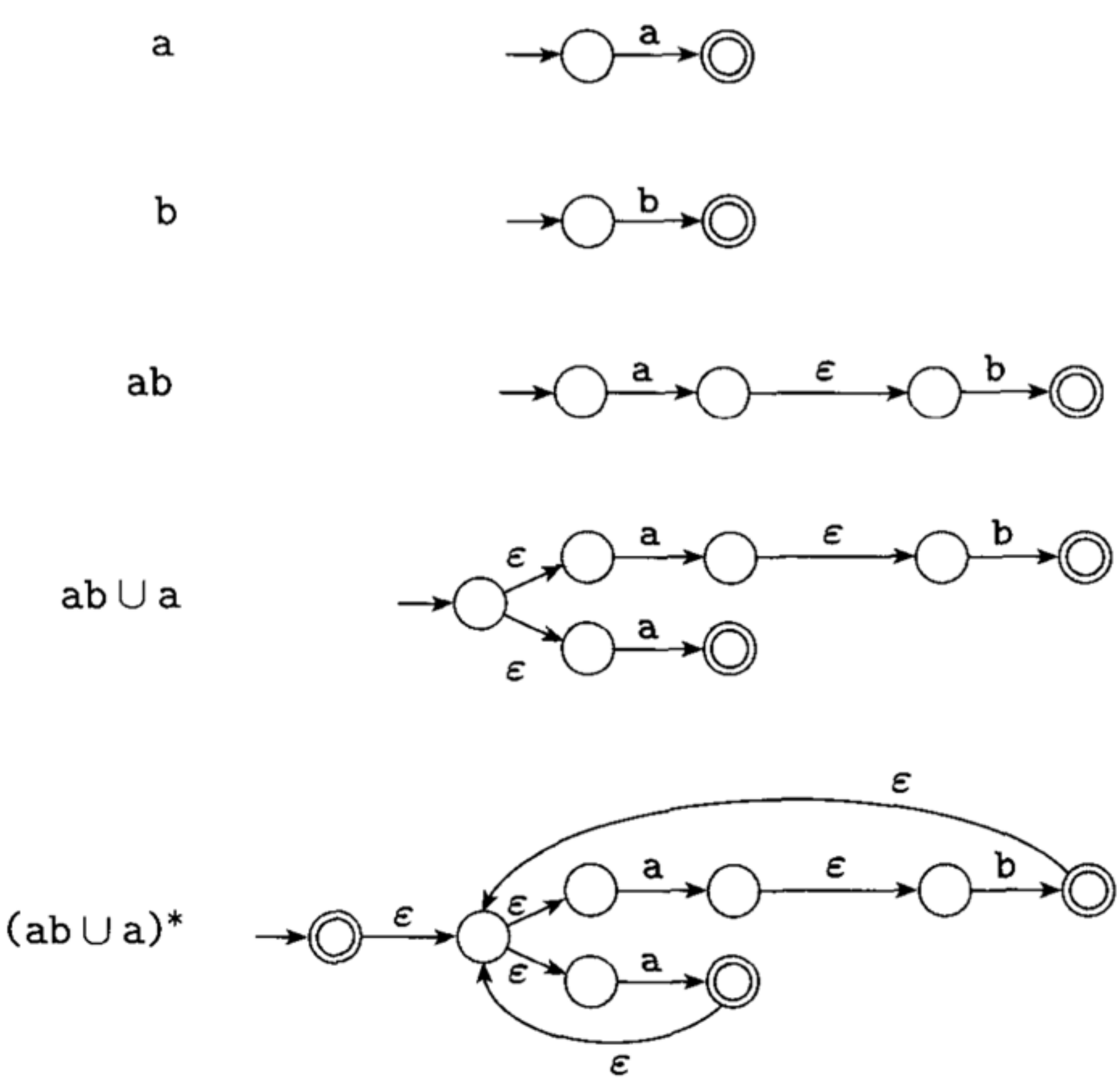


FIGURE 1.57
Building an NFA from the regular expression

(ab∪a)*

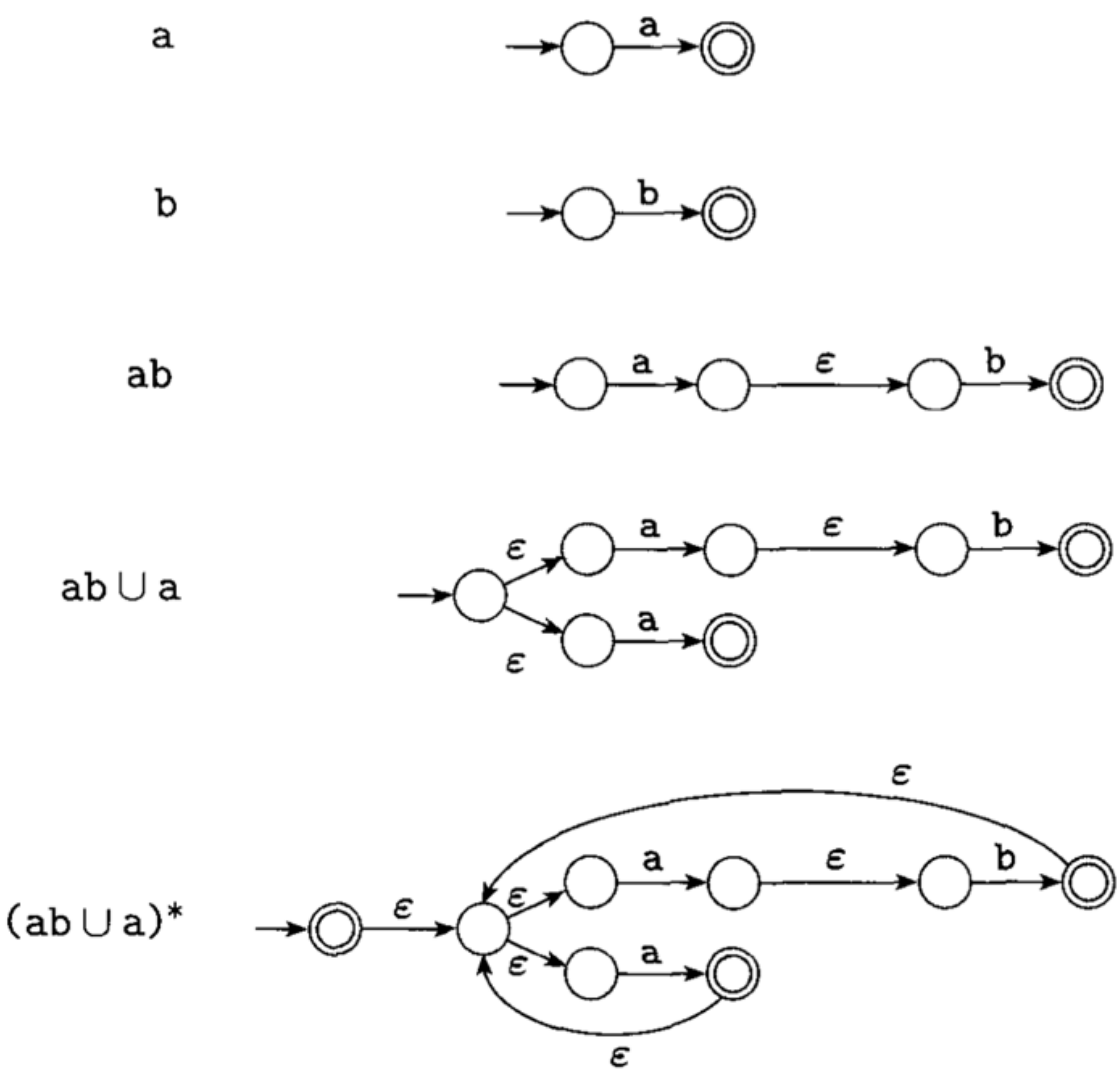


FIGURE 1.57
Building an NFA from the regular expression **(ab∪a)***

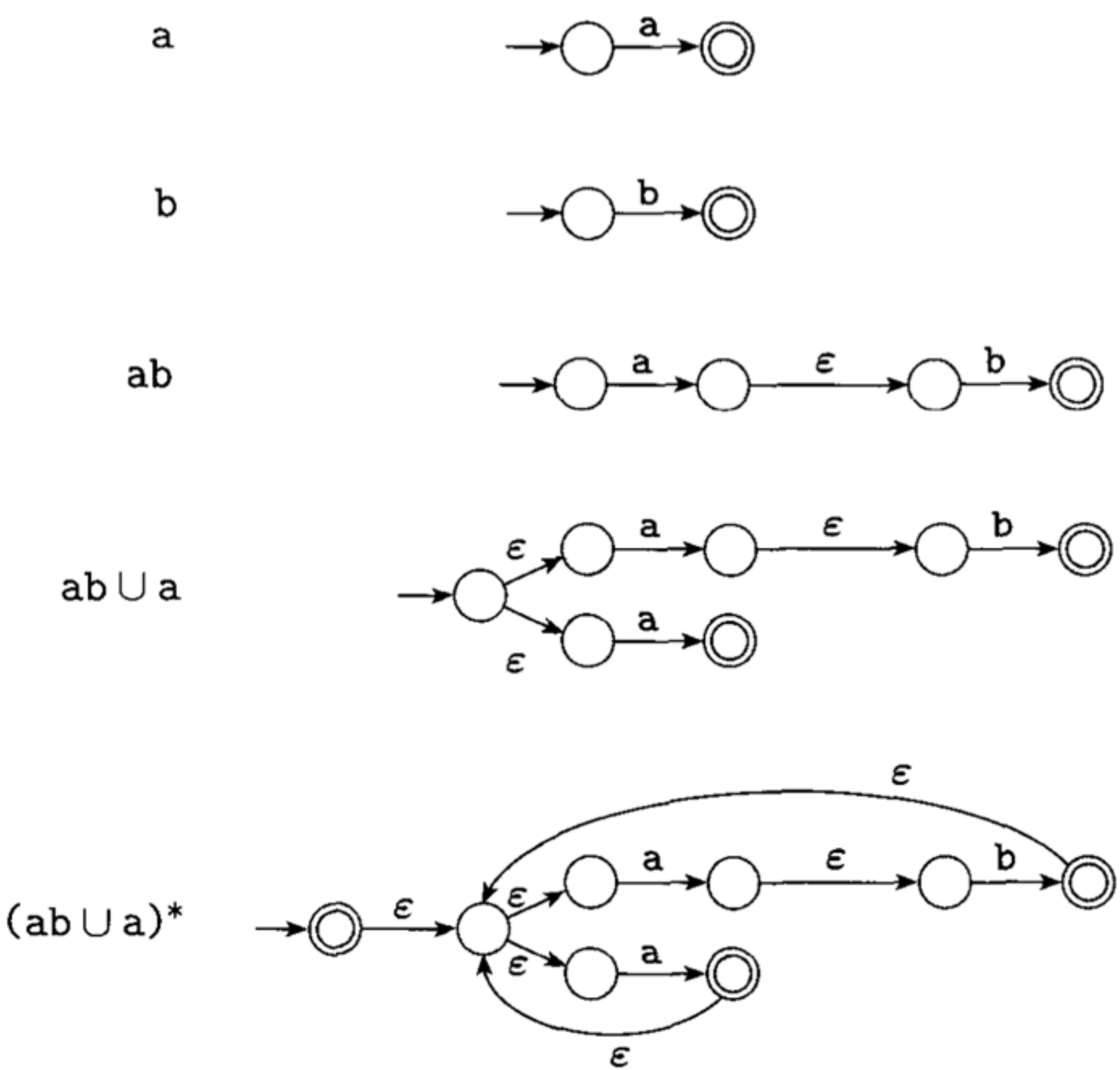


FIGURE 1.57
Building an NFA from the regular expression **(ab∪a)***

FIGURE 1.59

Building an NFA from the regular expression **(a∪b)*aba**

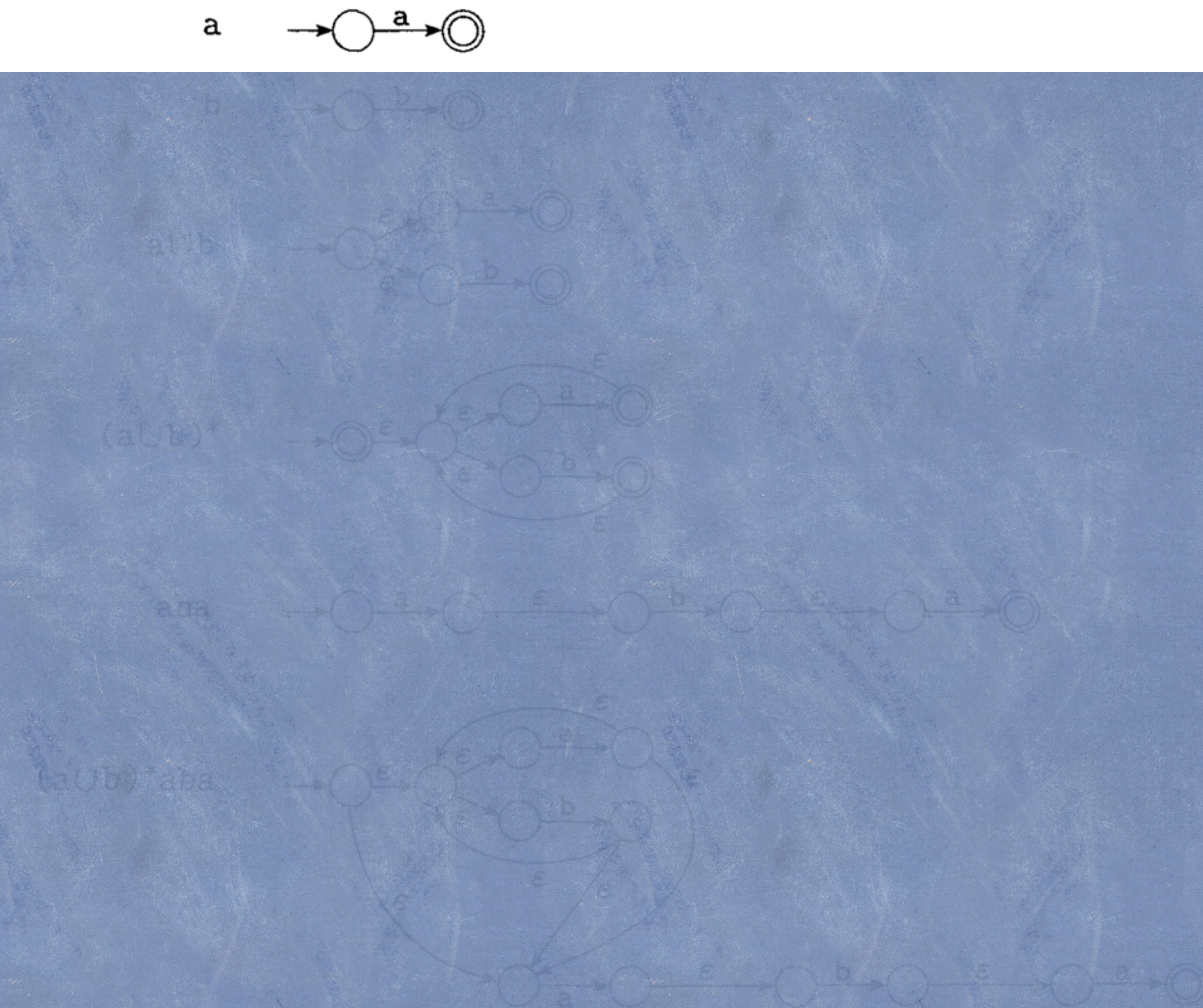


FIGURE 1.59
Building an NFA from the regular expression

(a∪b)*aba

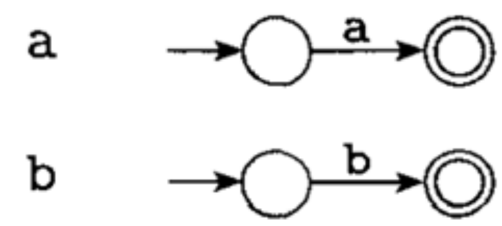


FIGURE 1.59
Building an NFA from the regular expression

(a∪b)*aba

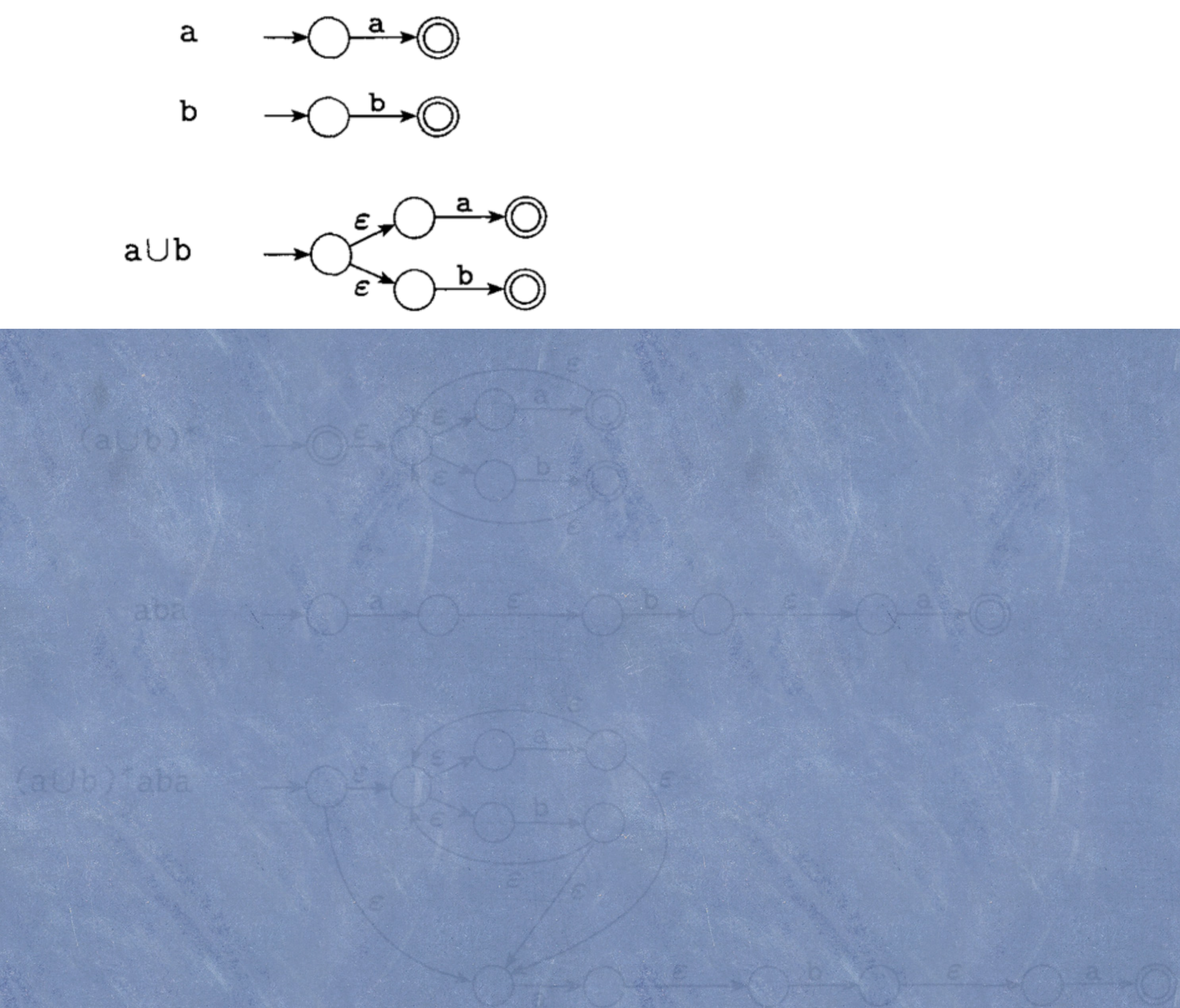


FIGURE 1.59
Building an NFA from the regular expressio

(a∪b)*aba

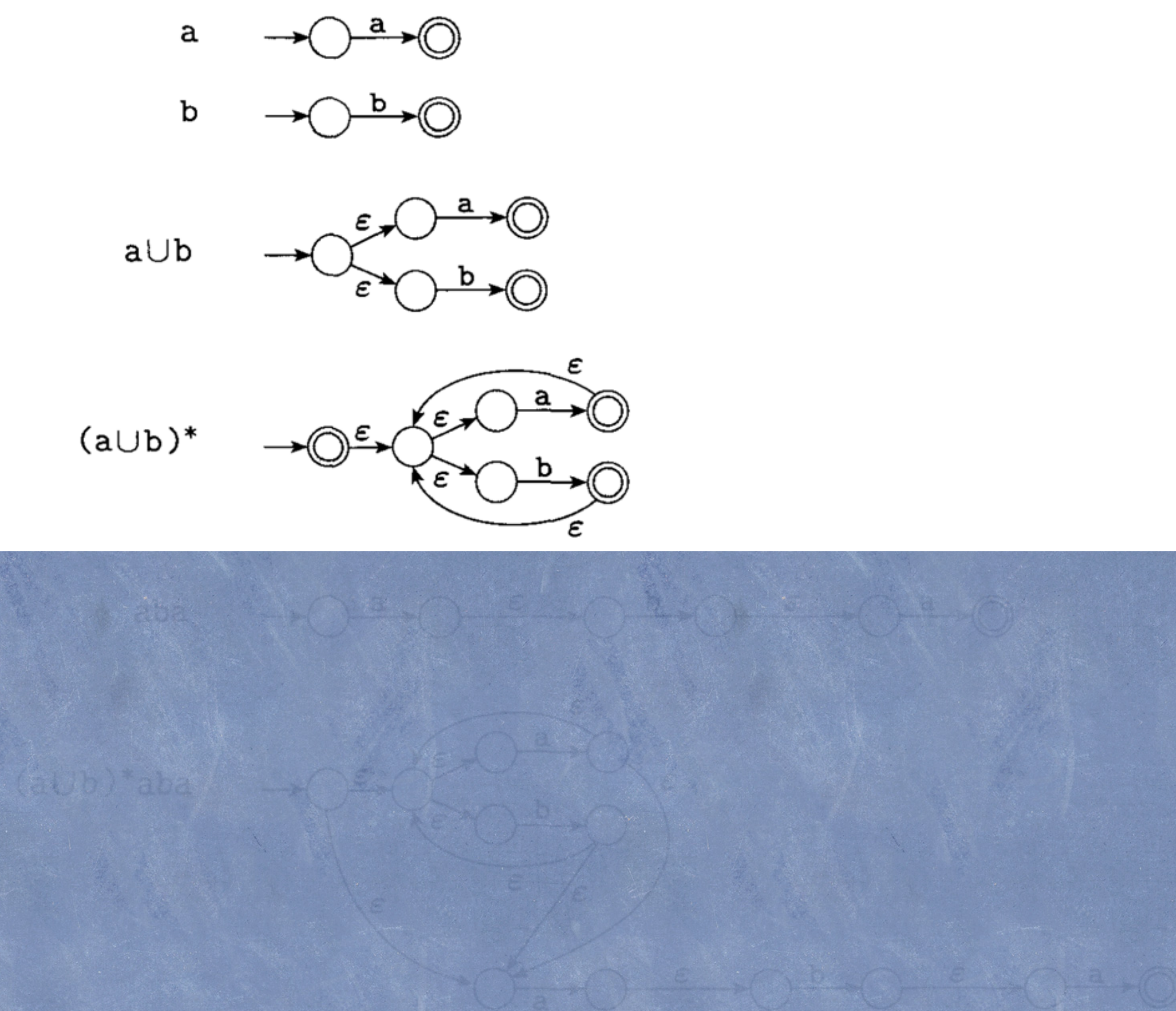
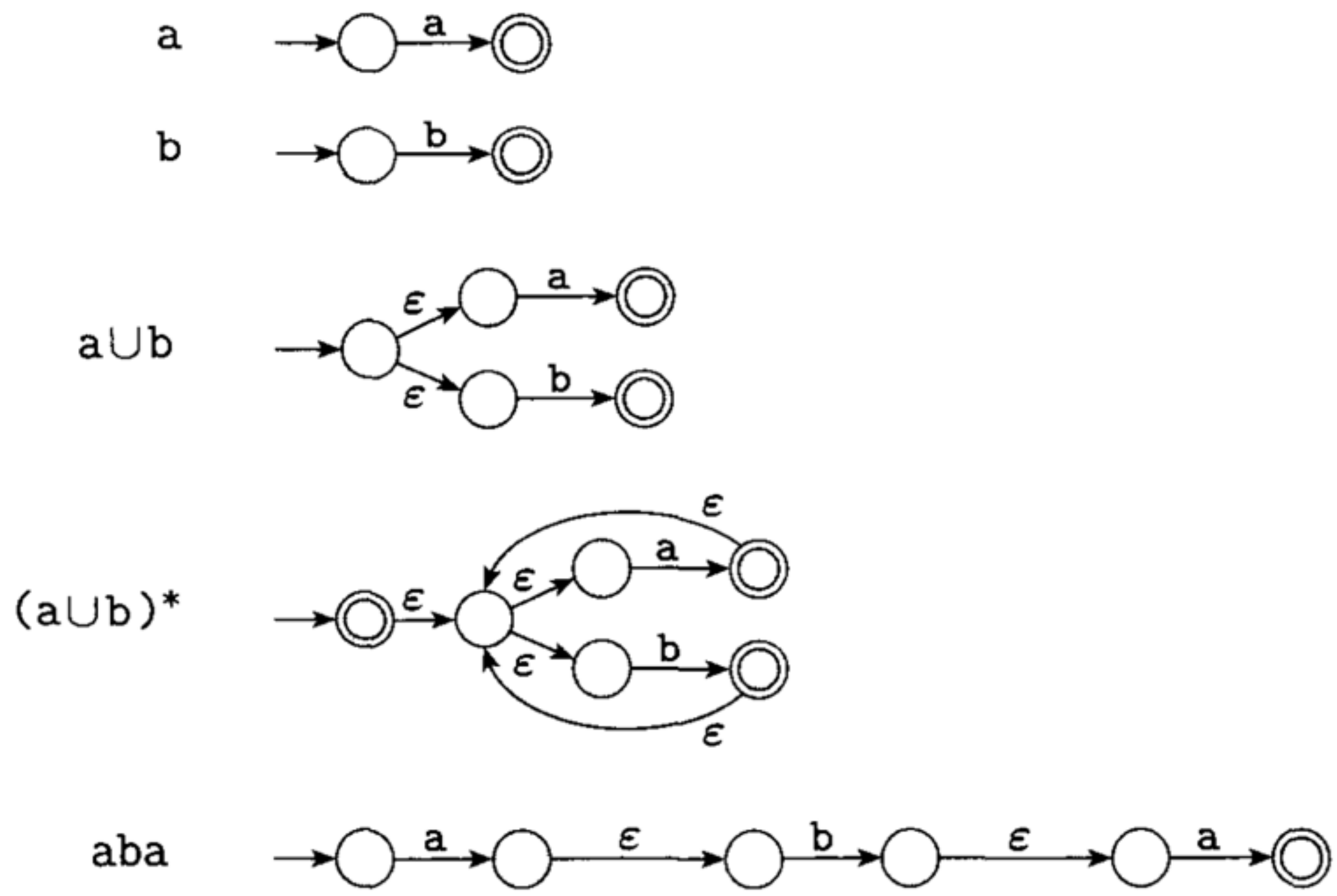


FIGURE 1.59
Building an NFA from the regular expression

(a∪b)*aba



(a∪b)*aba

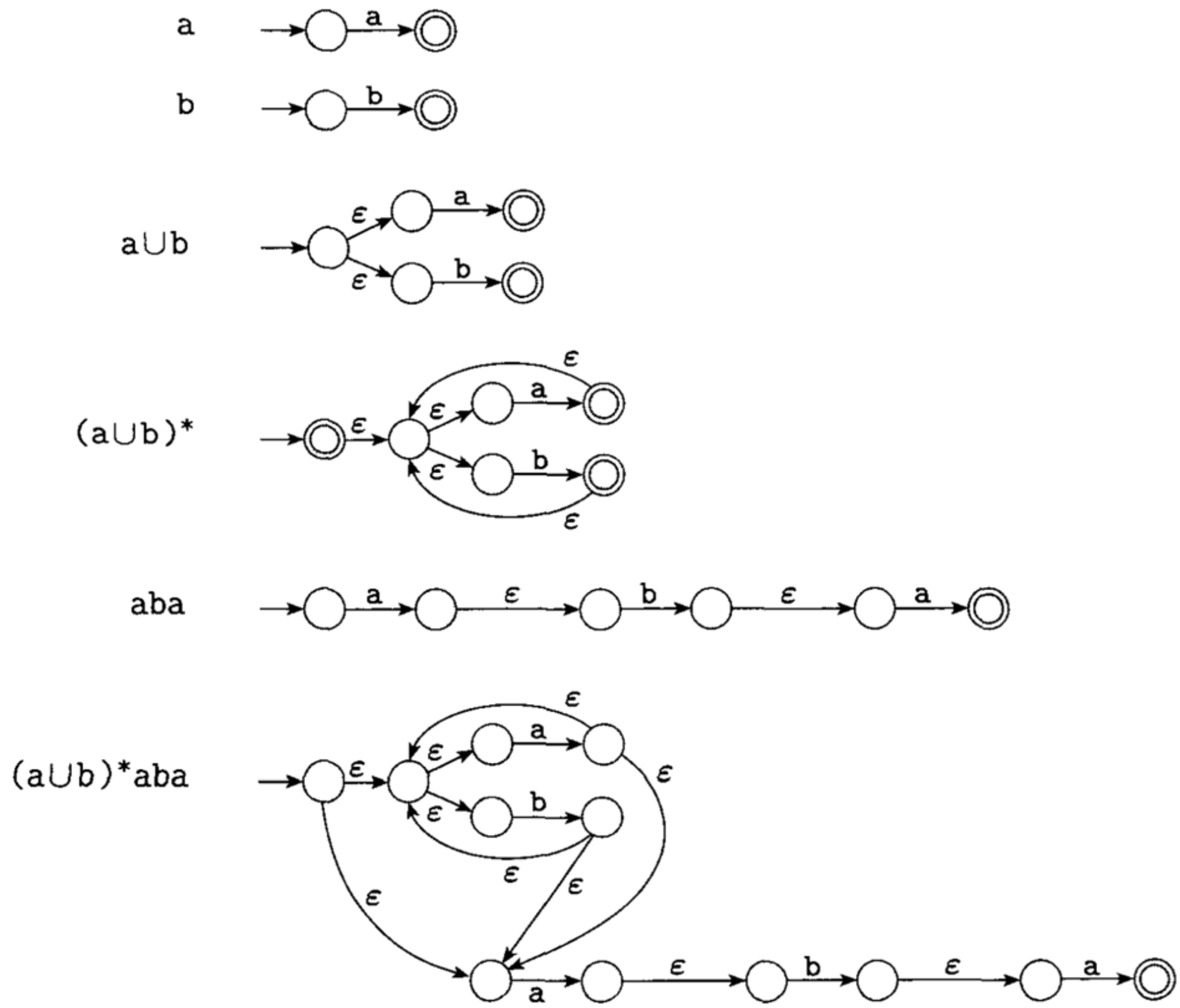


FIGURE 1.59
Building an NFA from the regular expression **(a∪b)*aba**

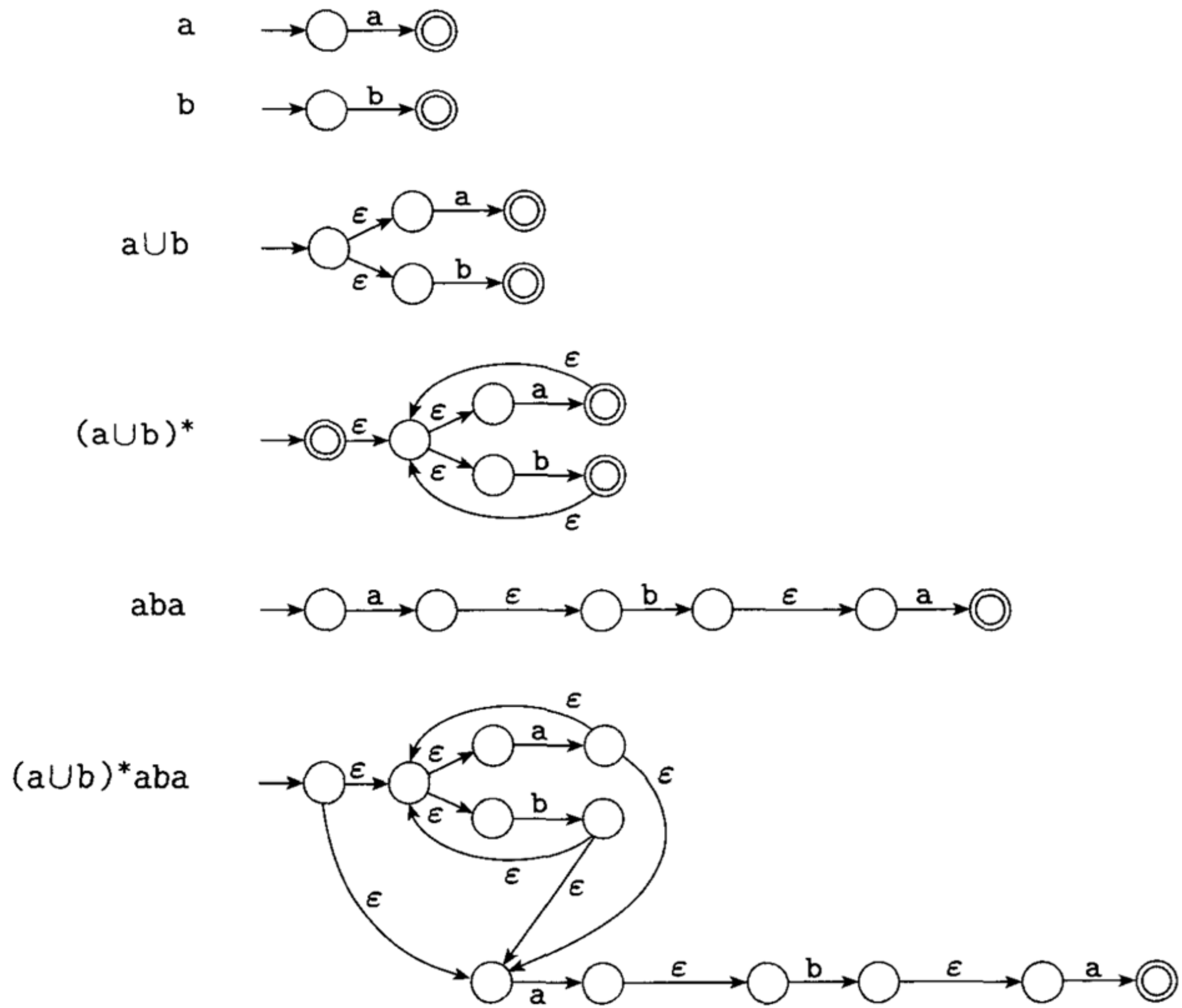


FIGURE 1.59
Building an NFA from the regular expression **(a∪b)*aba**

Automata recognize Regular Expressions

LEMMA 1.60

If a language is regular, then it is described by a regular expression.

Generalized NFA

Example of GNFA

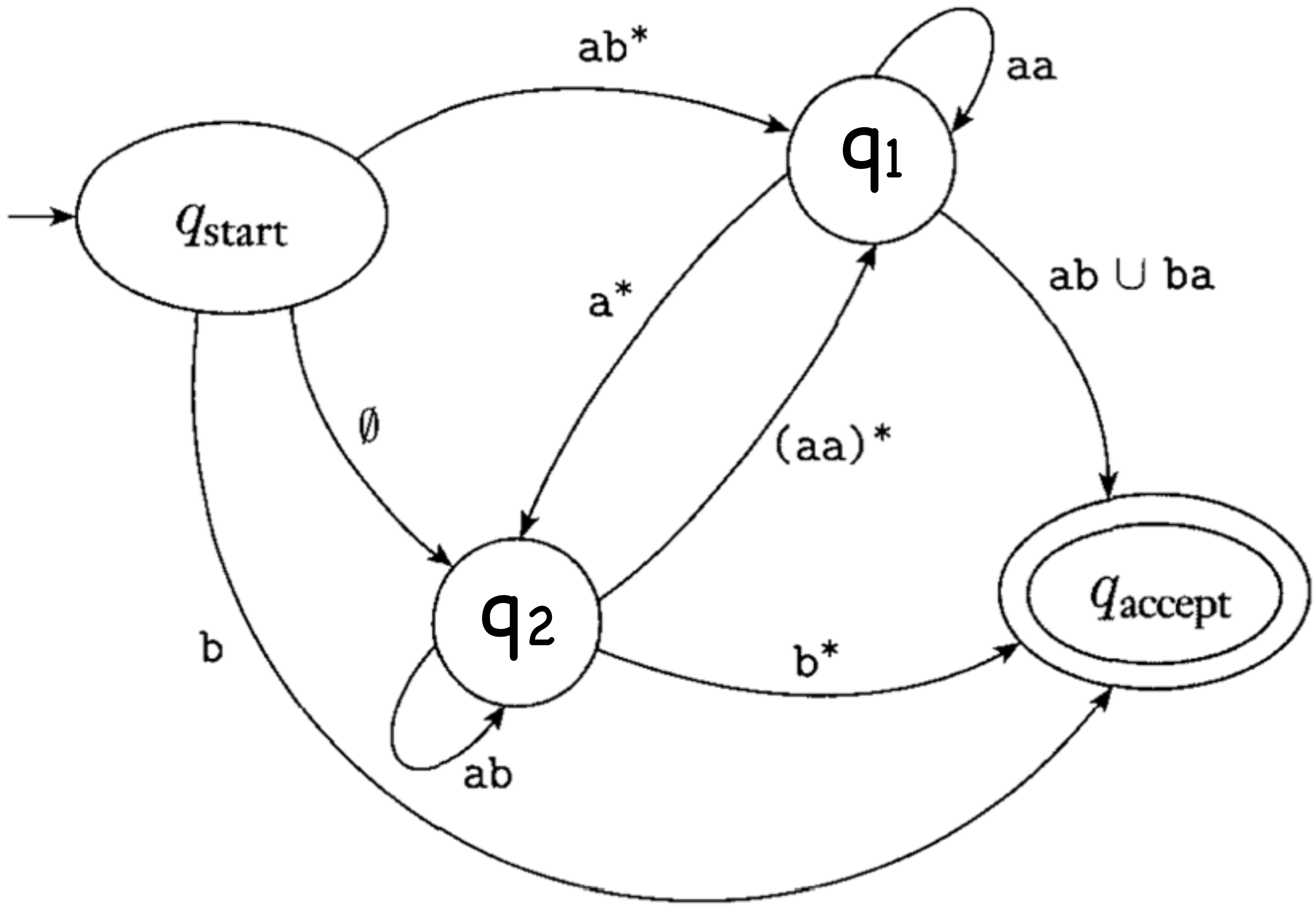
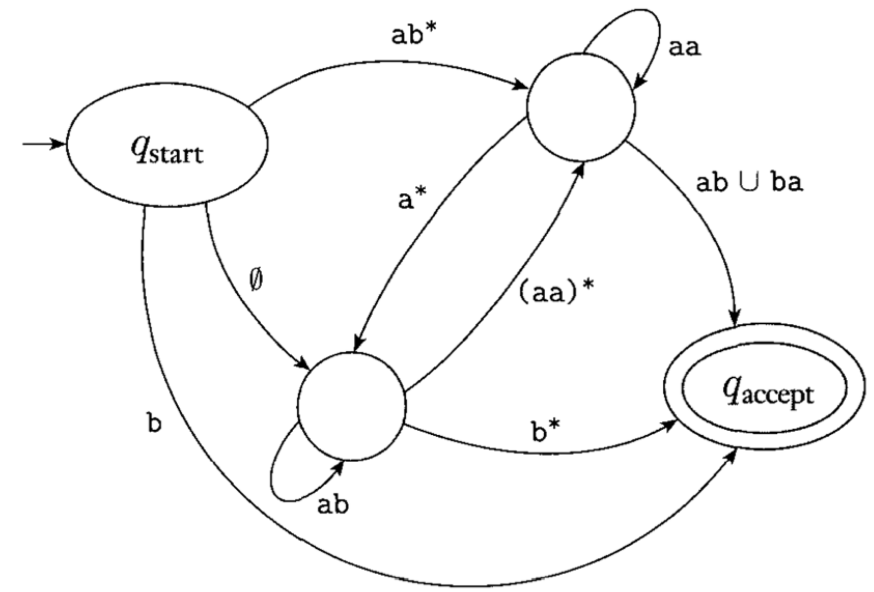


FIGURE 1.61
A generalized nondeterministic finite automaton

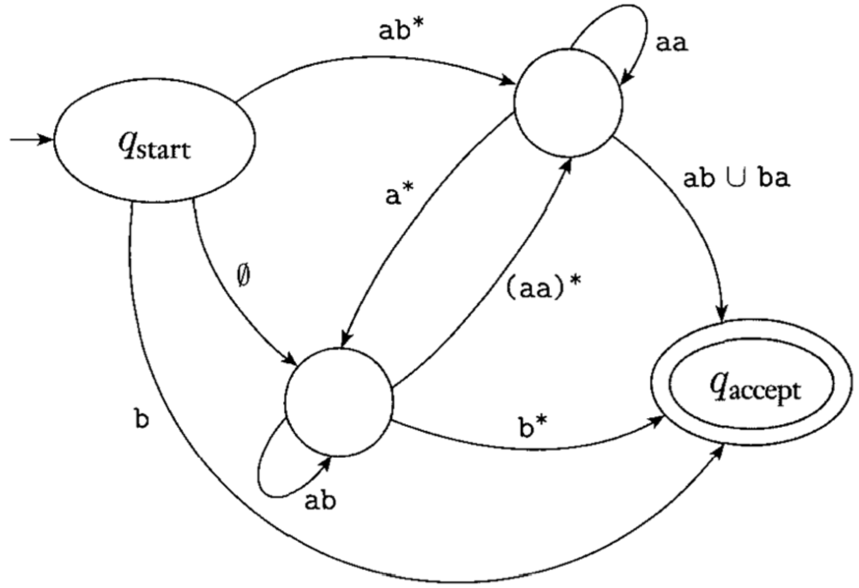
Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

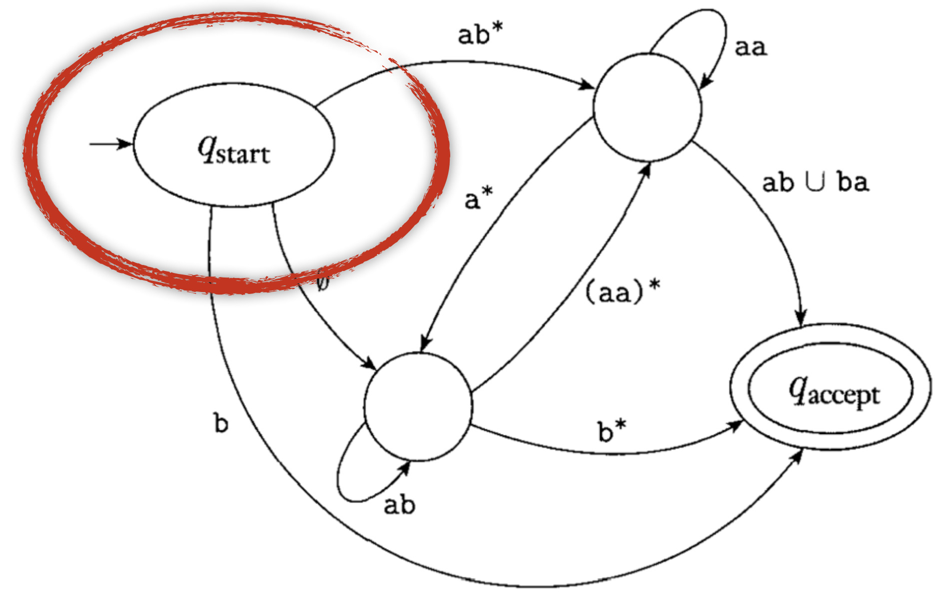
Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

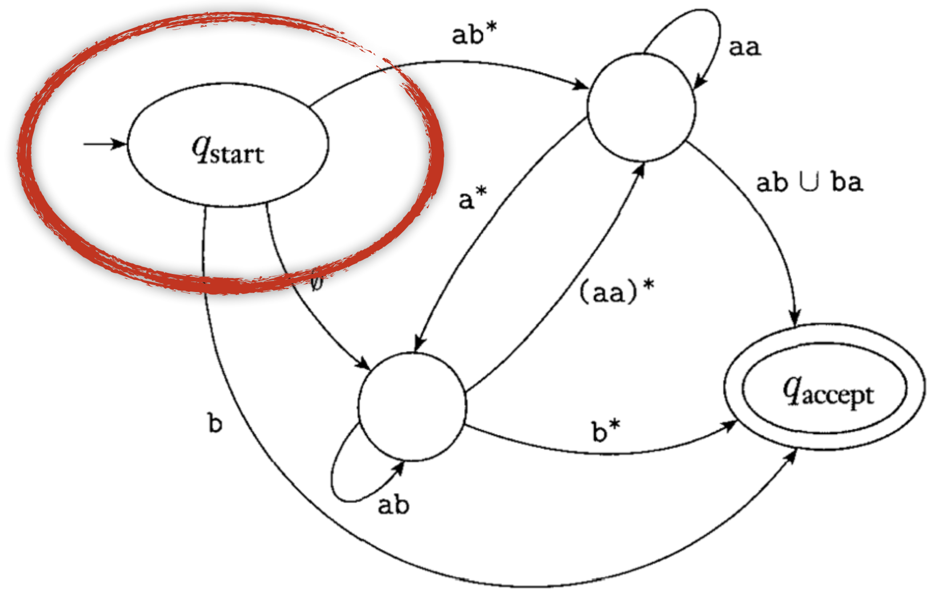
Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

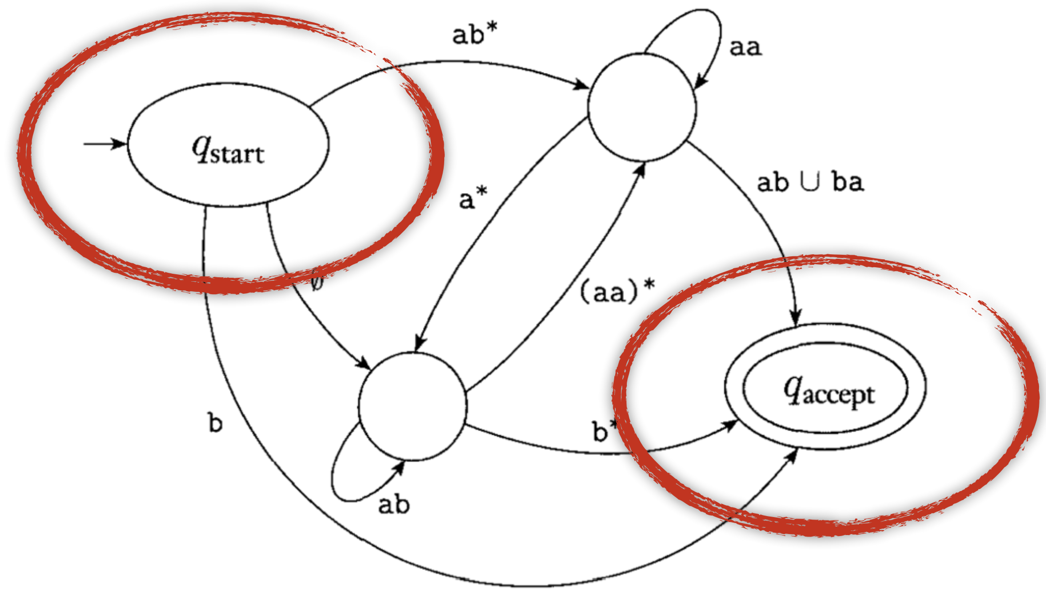
Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

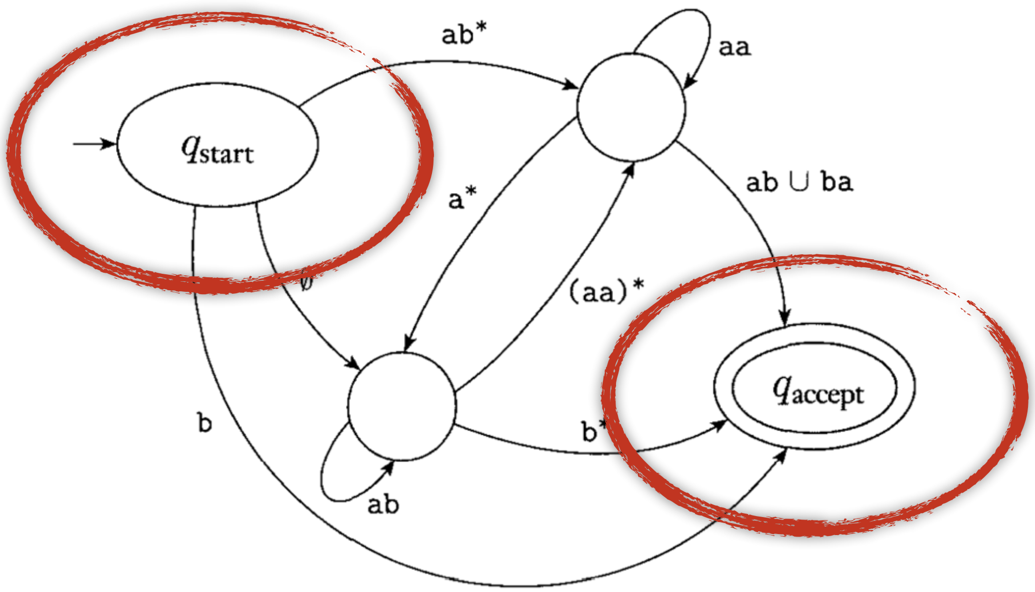
Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

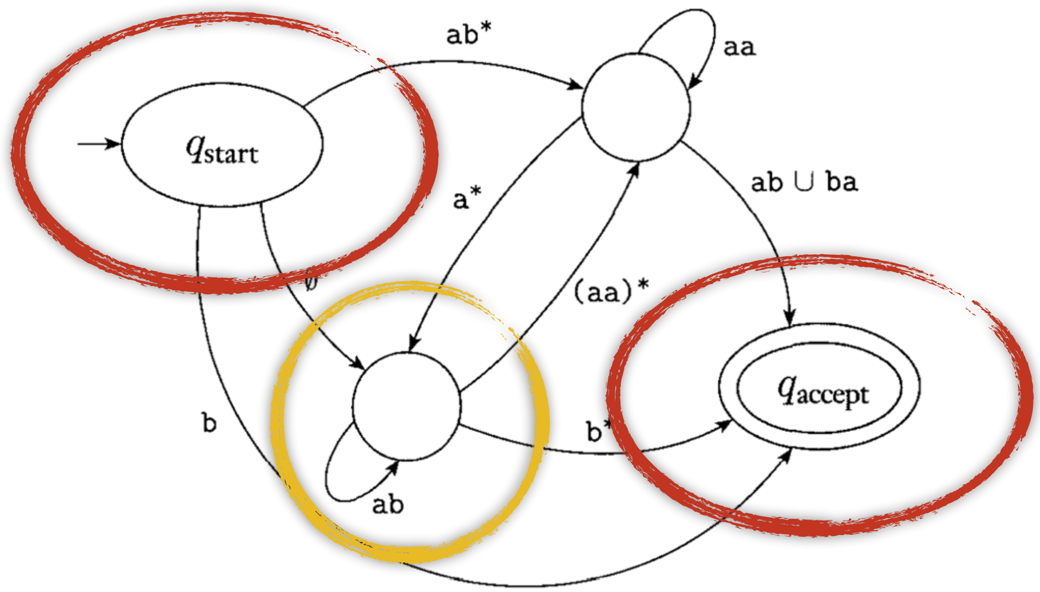
Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

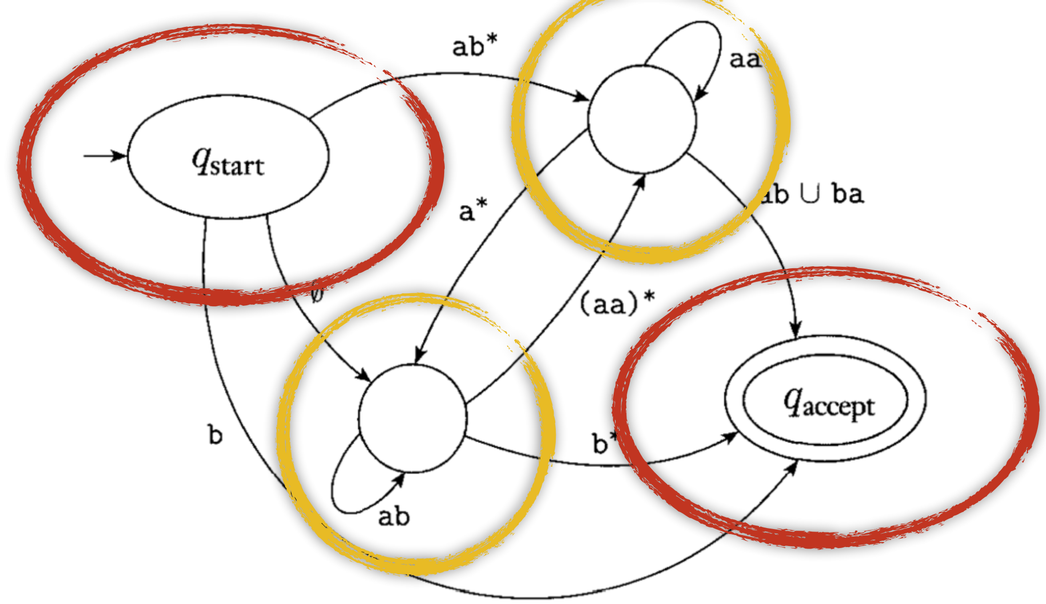
Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

Generalized NFA



For convenience we require that GNFAs always have a special form that meets the following conditions.

- The start state has transition arrows going to every other state but no arrows coming in from any other state.
- There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.
- Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.

Generalized NFA

DEFINITION 1.64

A ***generalized nondeterministic finite automaton*** is a 5-tuple, $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta\colon (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \longrightarrow \mathcal{R}$ is the transition function,
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

Definition of GNFA

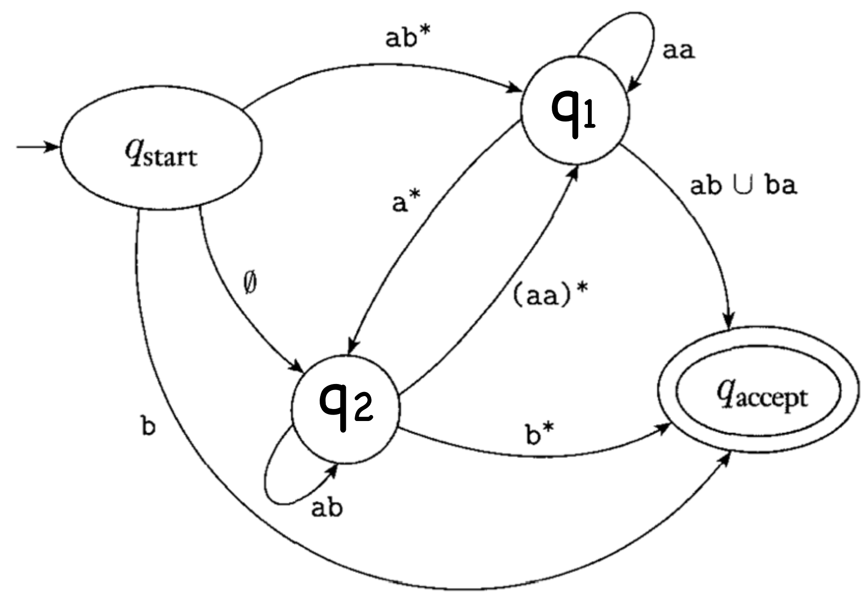


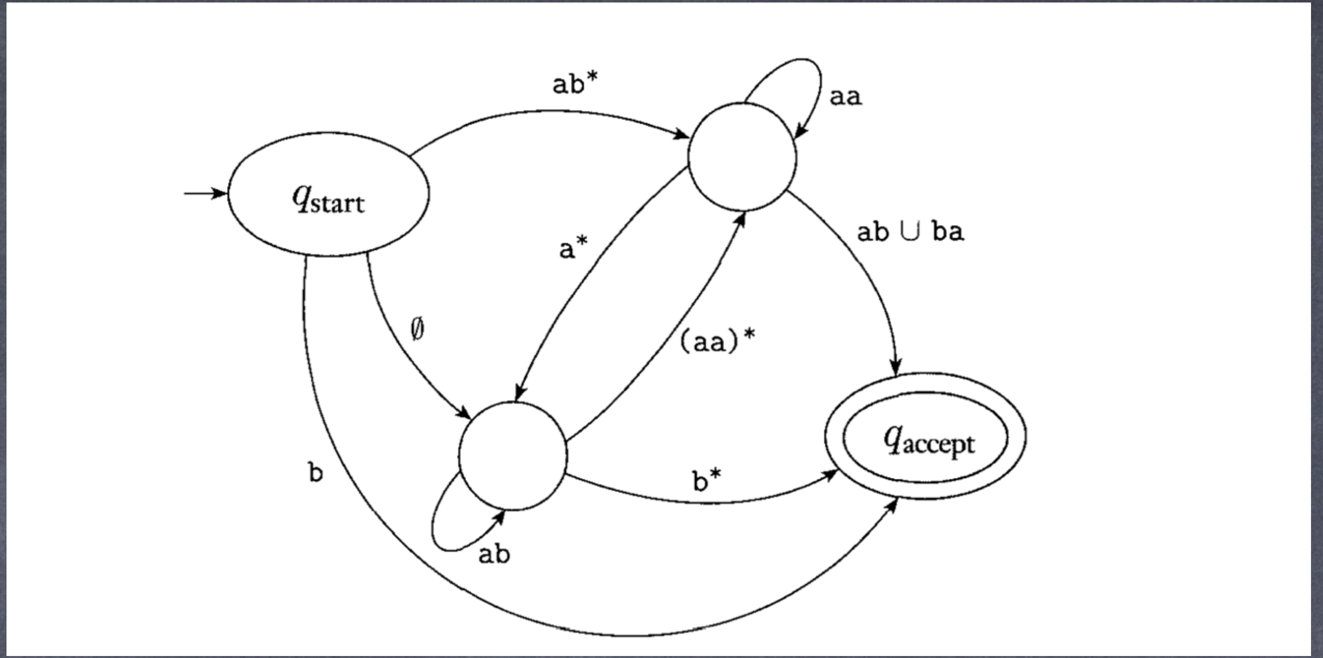
FIGURE 1.61
A generalized nondeterministic finite automaton

1. $Q=\{q_{start}, q_1, q_2, q_{accept}\}$
2. $\Sigma=\{a,b\}$
3. δ is given as

δ	q_1	q_2	q_{accept}
q_{start}	ab^*	$\emptyset$	b
q_1	aa	a^*	$ab \cup ba$
q_2	$(aa)^*$	ab	b^*

4. q_{start} is the start state
5. q_{accept} is the (unique) accept state

Definition of GNFA



- Let $G = (Q,\Sigma,\delta,q_{start},q_{accept})$ be a generalized nondeterministic finite state automaton and let $w=w_1w_2...w_n$ ($n \geq 0$) be a string where each sub-string $w_i \in \Sigma^*$.

- G **accepts** w if $\exists\ s_0,s_1,...,s_n$ s.t.
  1. $s_0 = q_{start}$
  2. $w_i \in L(\ \delta(s_{i-1},s_i)\ )$ for $i = 1 ... n$, and
  3. $s_n = q_{accept}$

Example of GNFA

abbbaaaababaaba

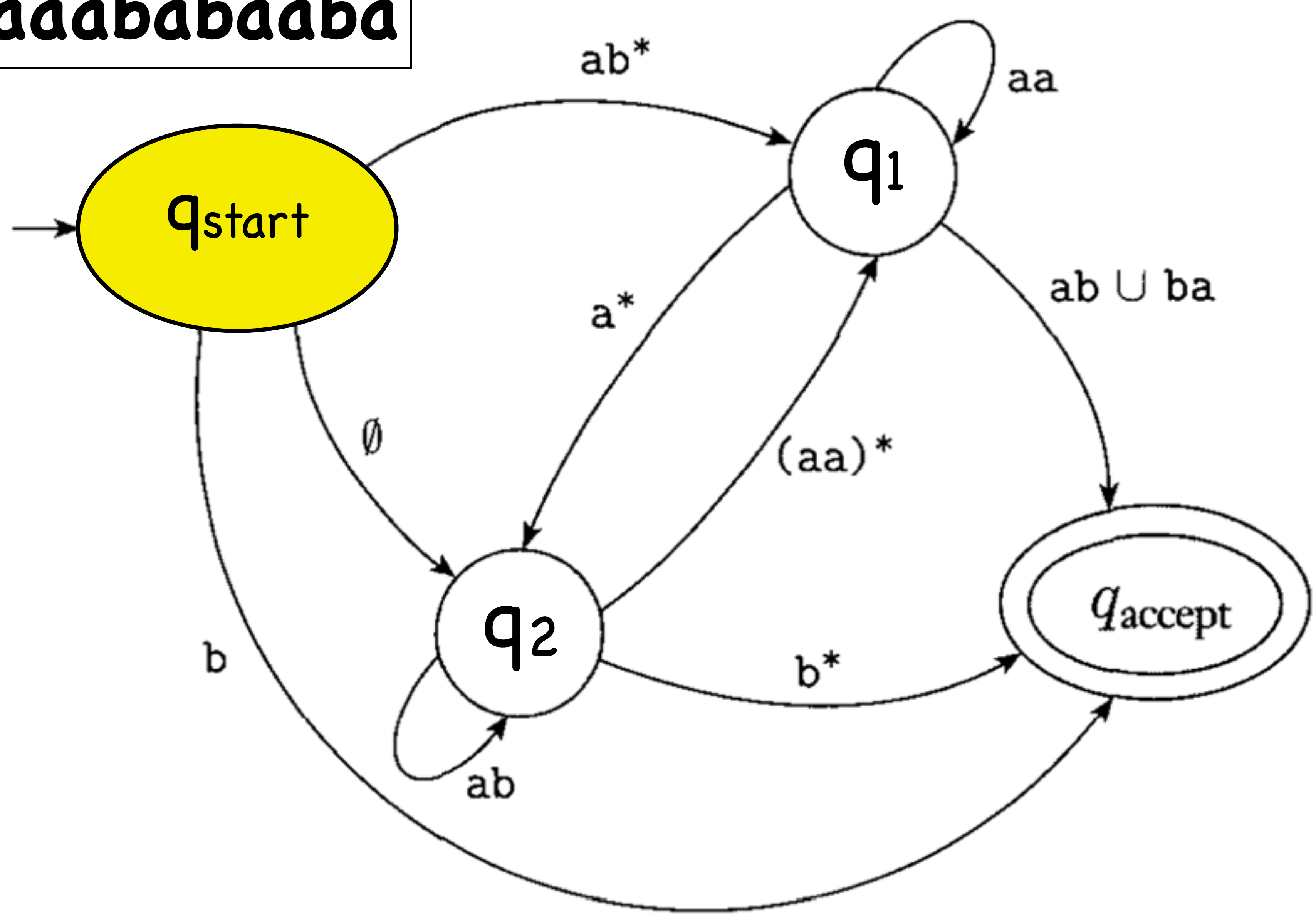


FIGURE 1.61
A generalized nondeterministic finite automaton

Example of GNFA

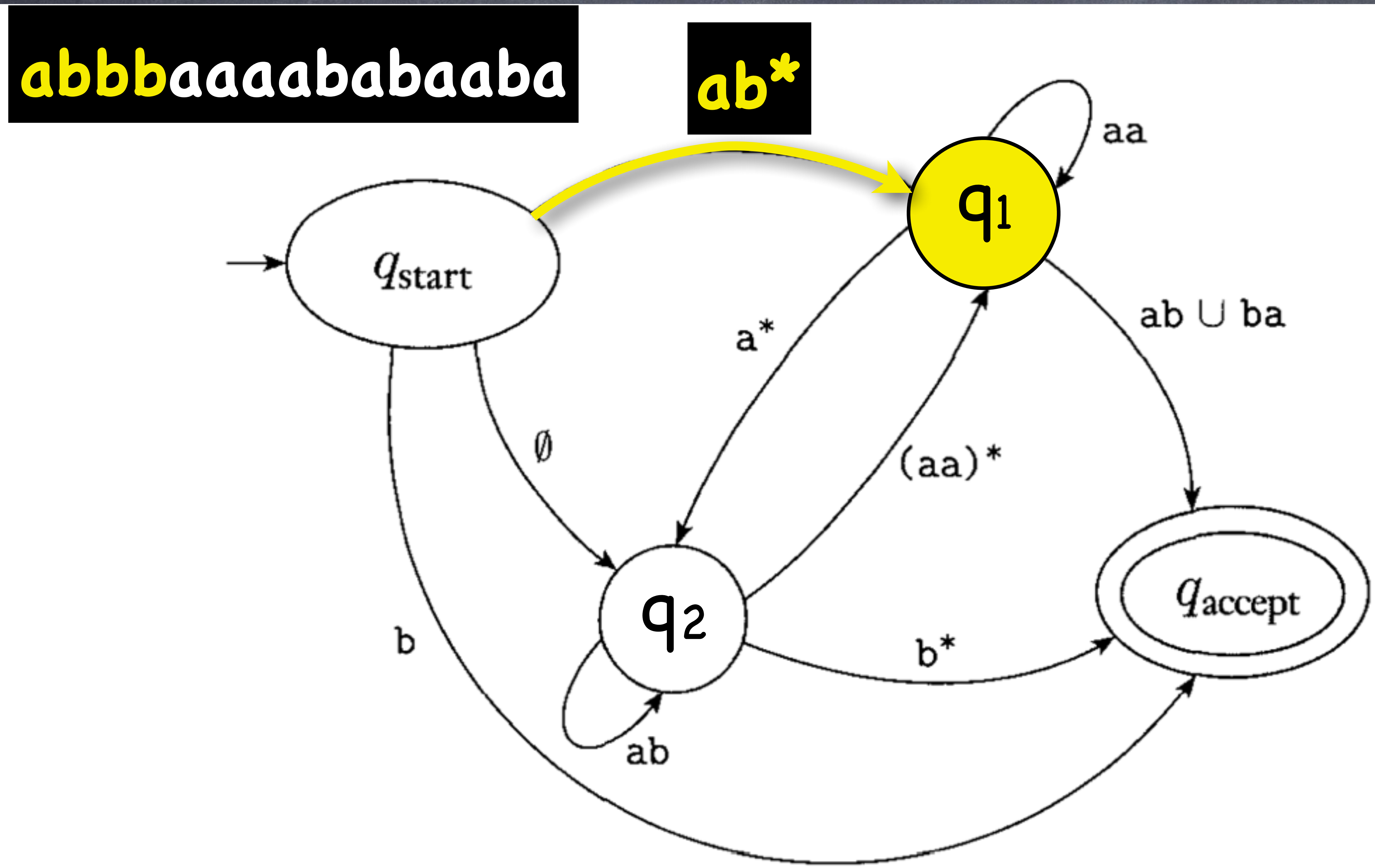


FIGURE 1.61
A generalized nondeterministic finite automaton

Example of GNFA

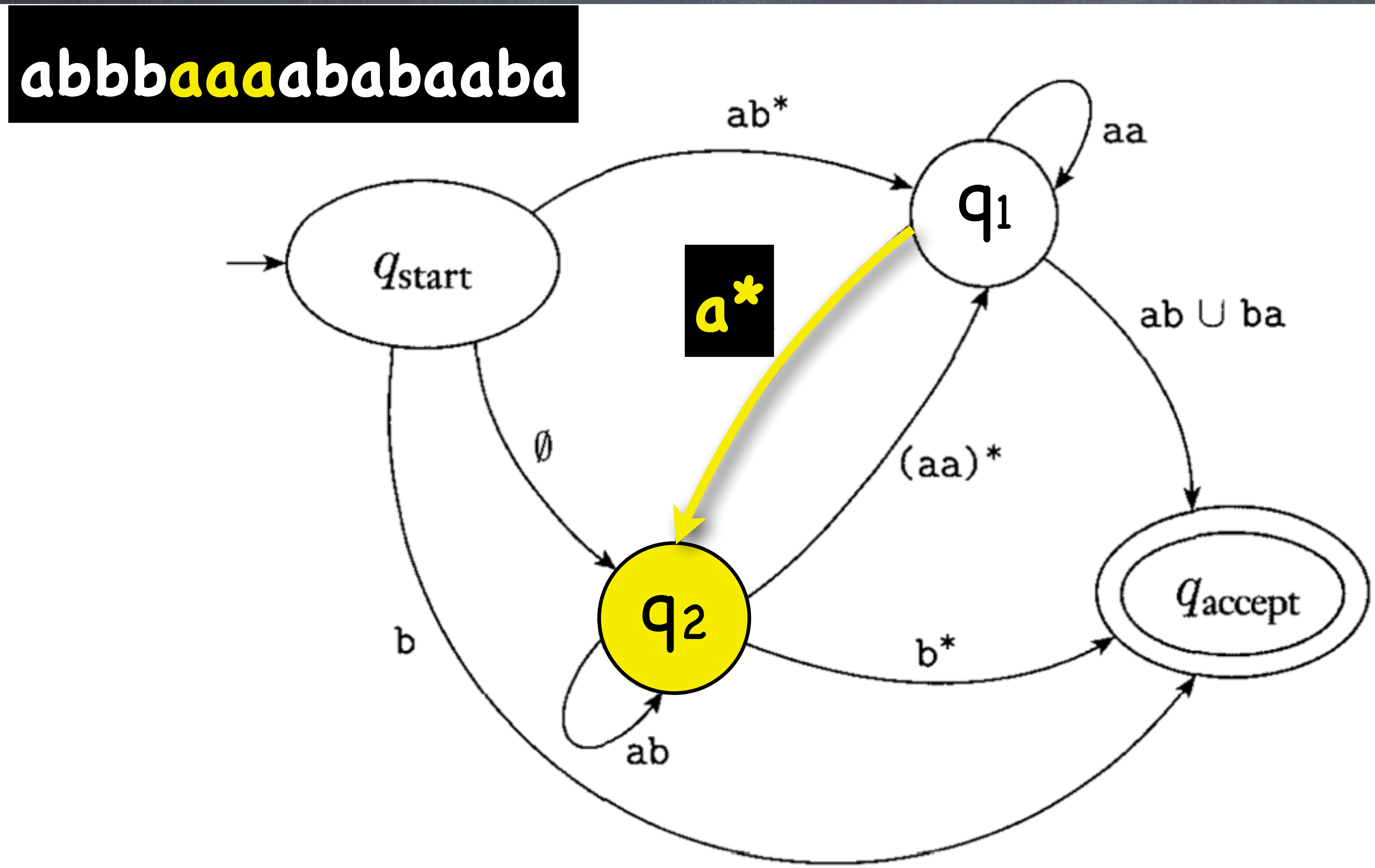


FIGURE 1.61
A generalized nondeterministic finite automaton

Example of GNFA

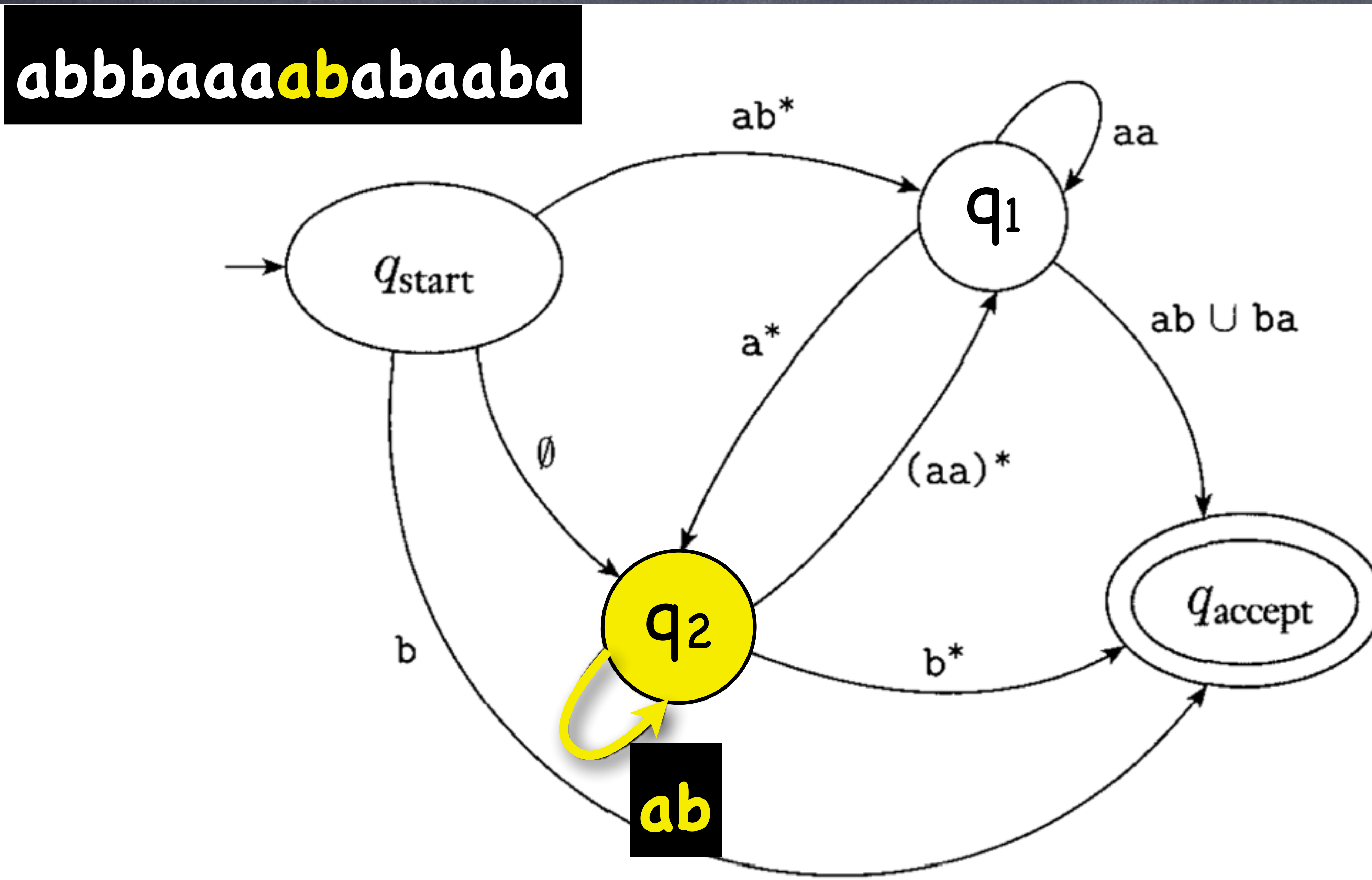


FIGURE 1.61
A generalized nondeterministic finite automaton

Example of GNFA

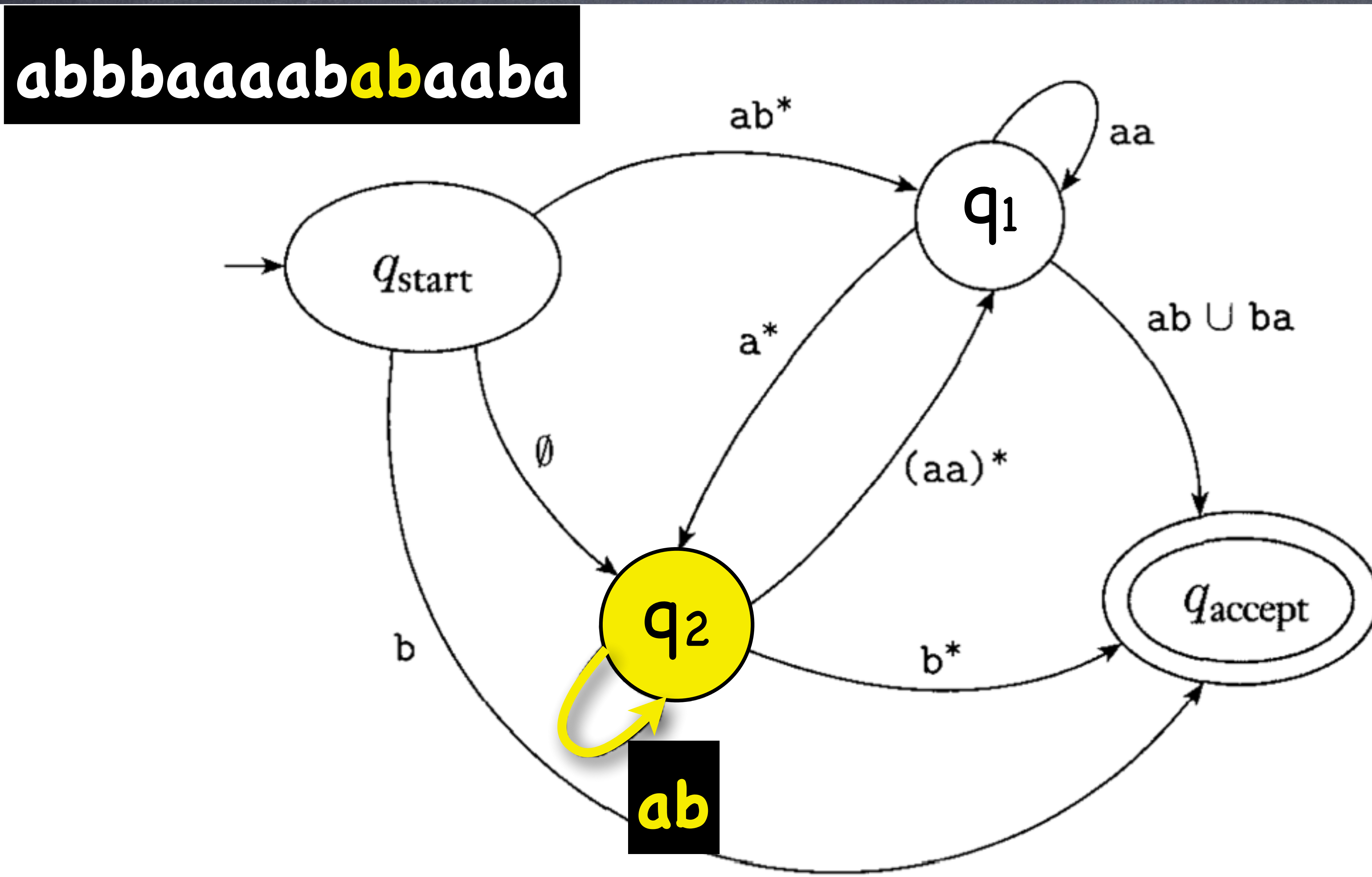


FIGURE 1.61
A generalized nondeterministic finite automaton

Example of GNFA

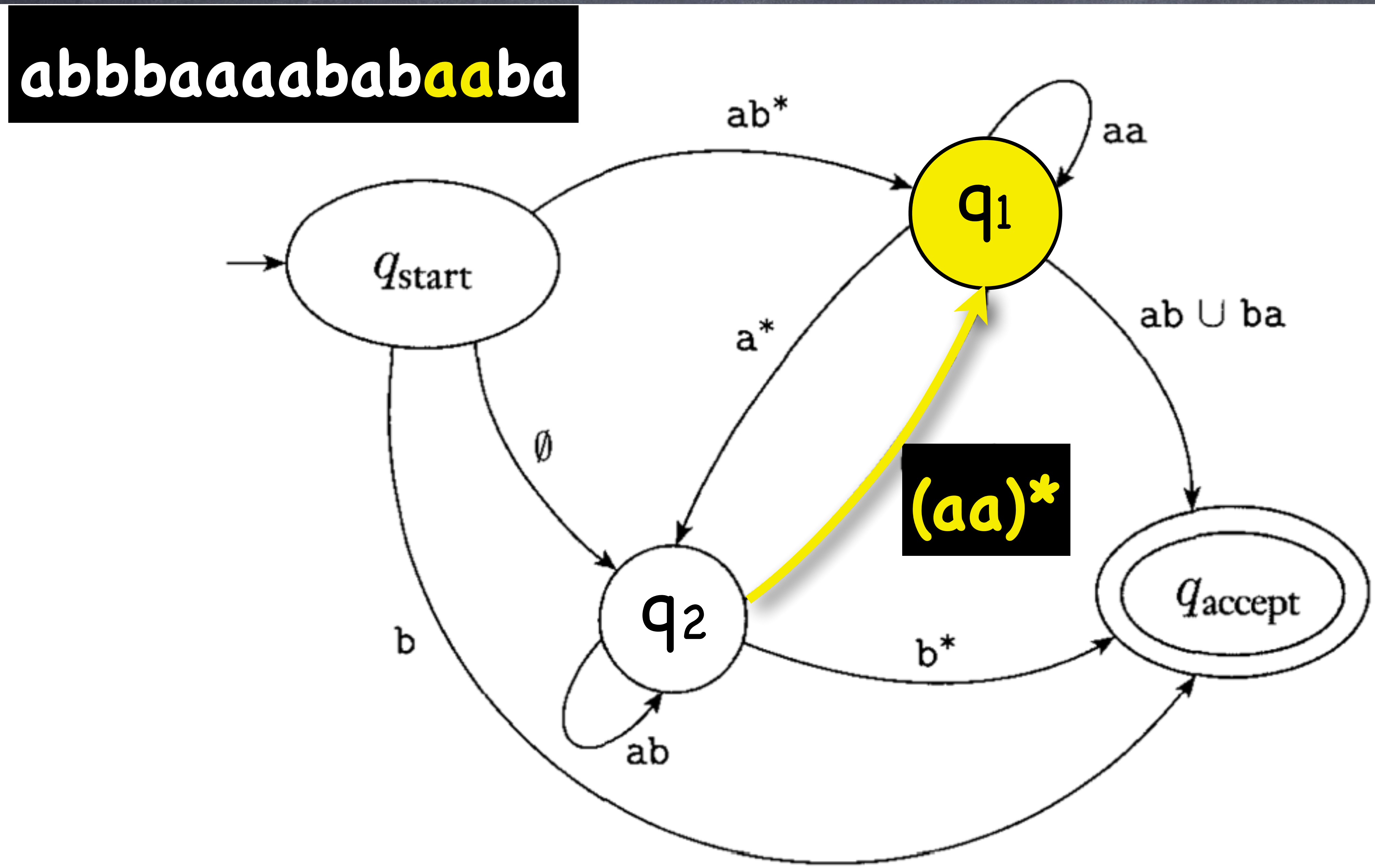


FIGURE 1.61
A generalized nondeterministic finite automaton

Example of GNFA

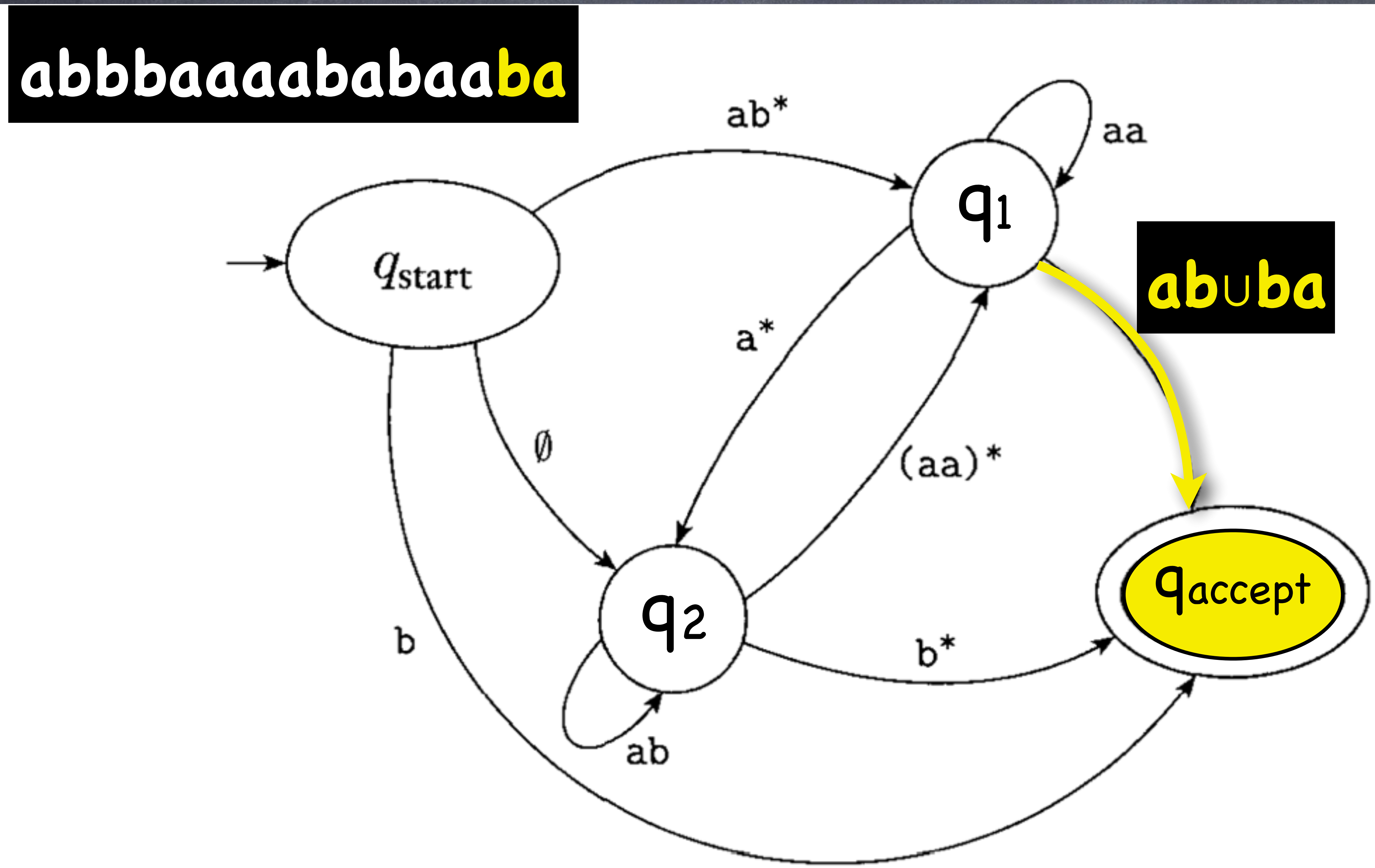


FIGURE 1.61
A generalized nondeterministic finite automaton

DFA → GNFA → Reg. Exp.

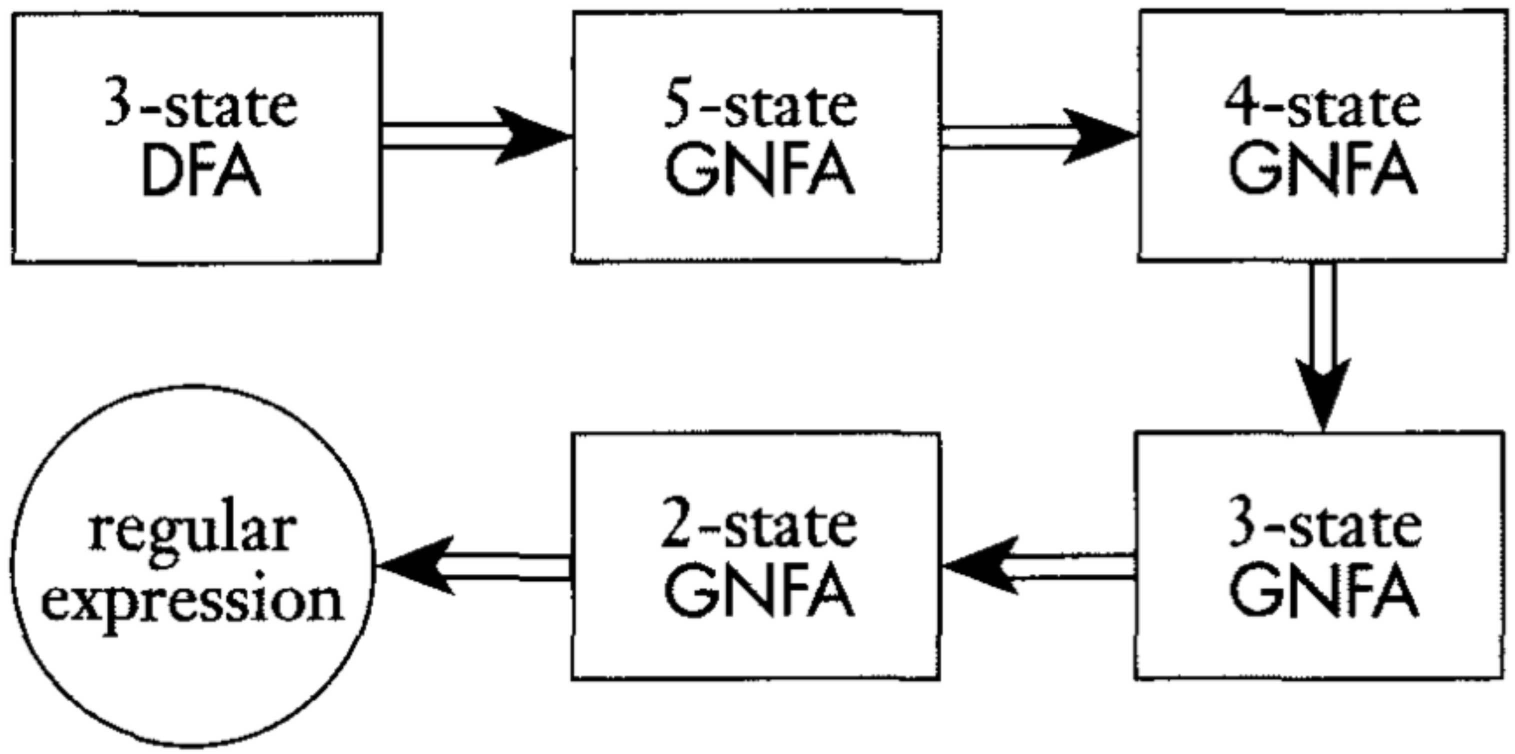
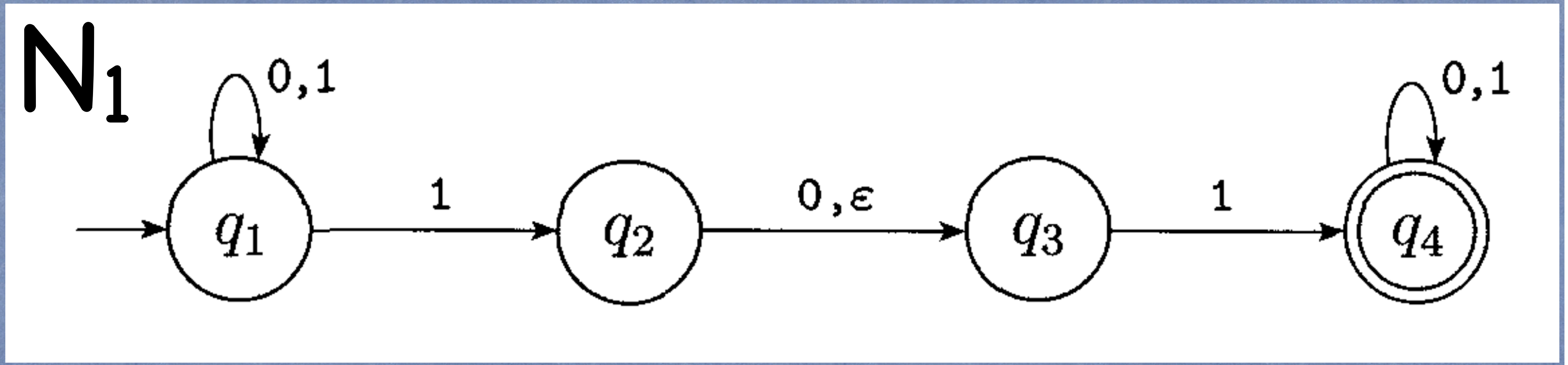
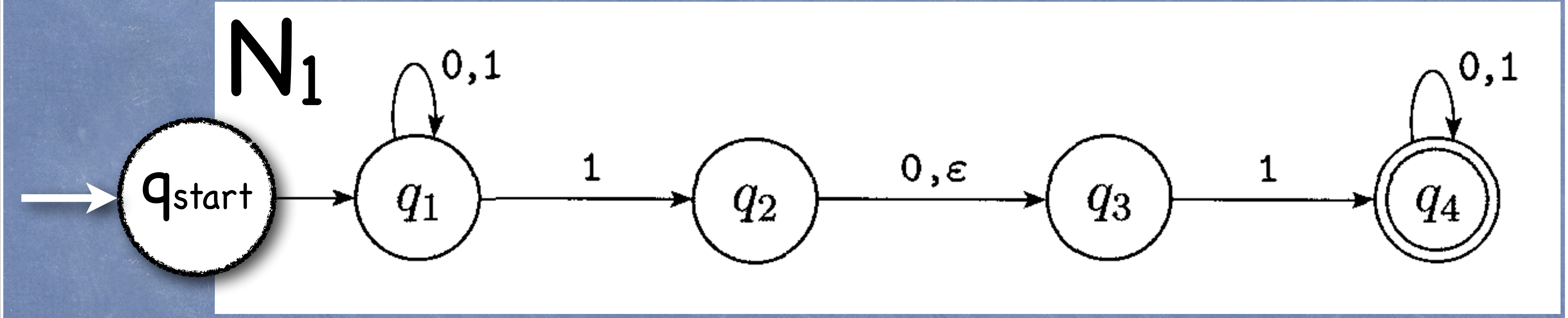


FIGURE 1.62
Typical stages in converting a DFA to a regular expression

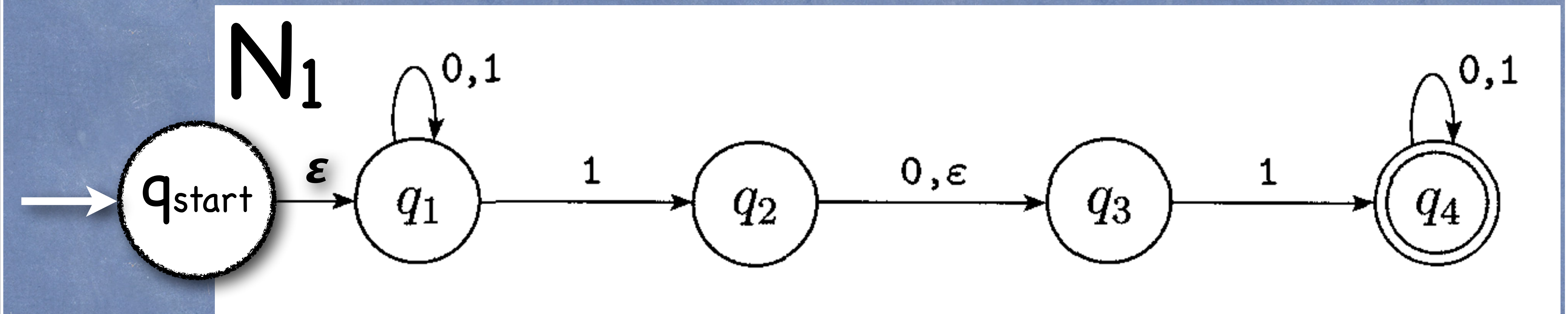
Example NFA→GNFA



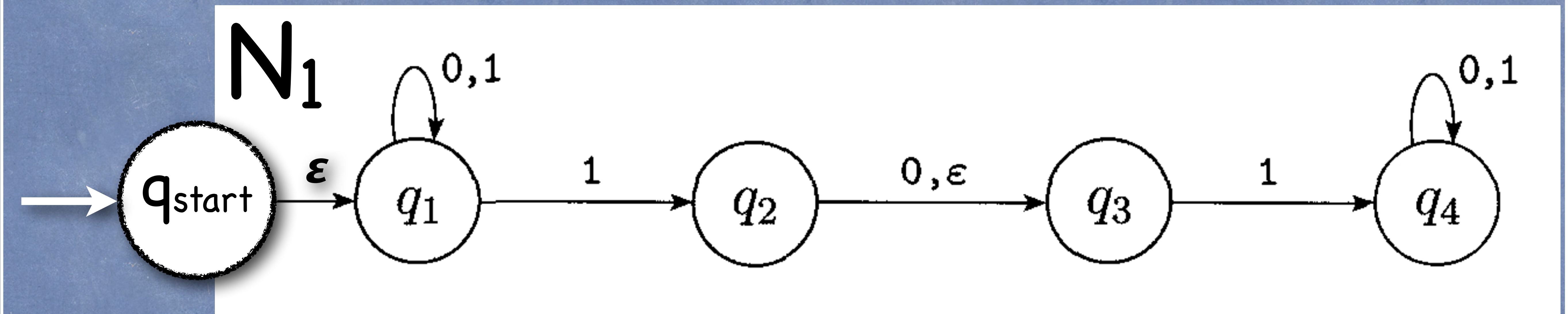
Example NFA→GNFA



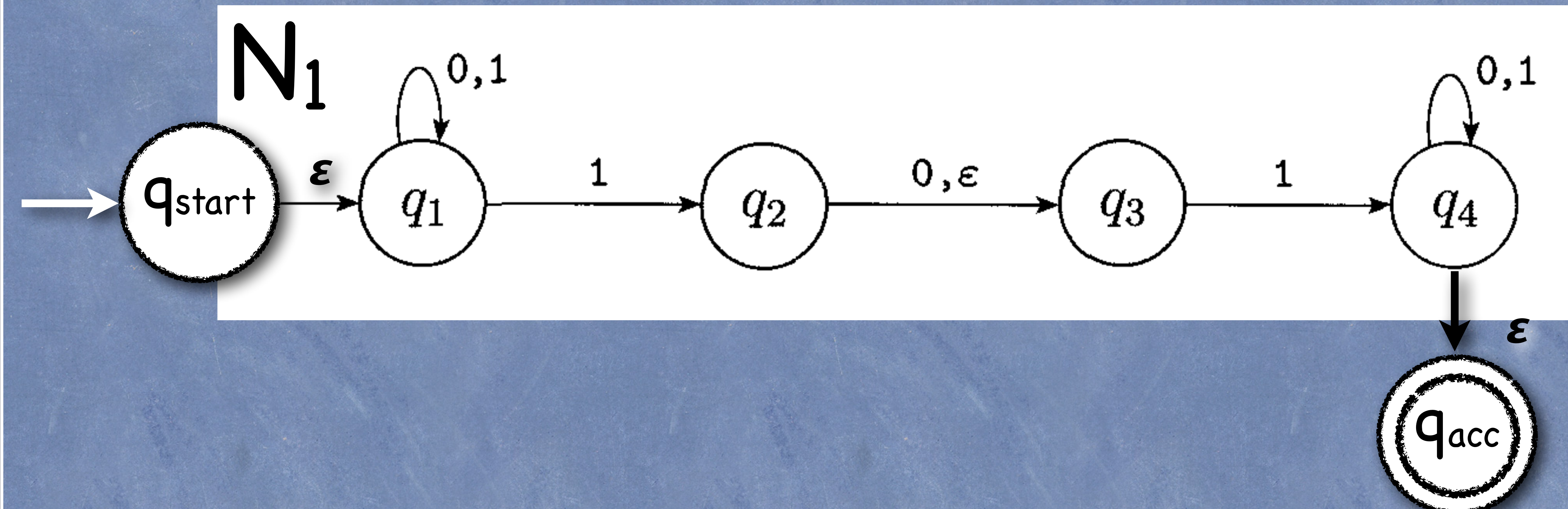
Example NFA→GNFA



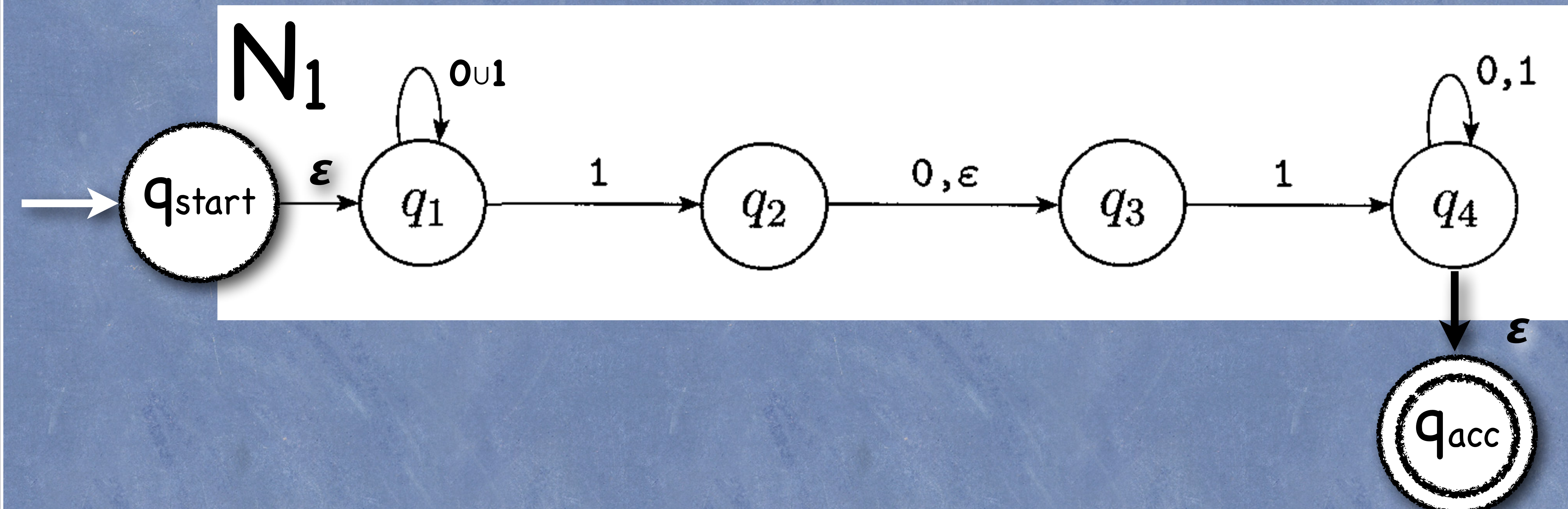
Example NFA→GNFA



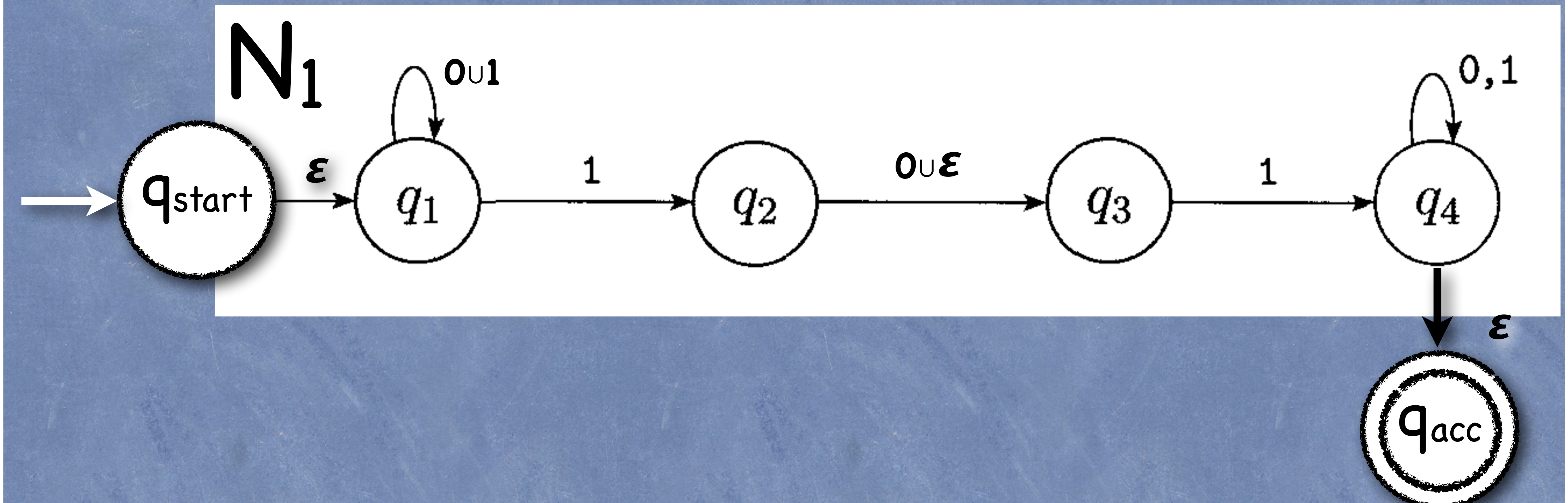
Example NFA→GNFA



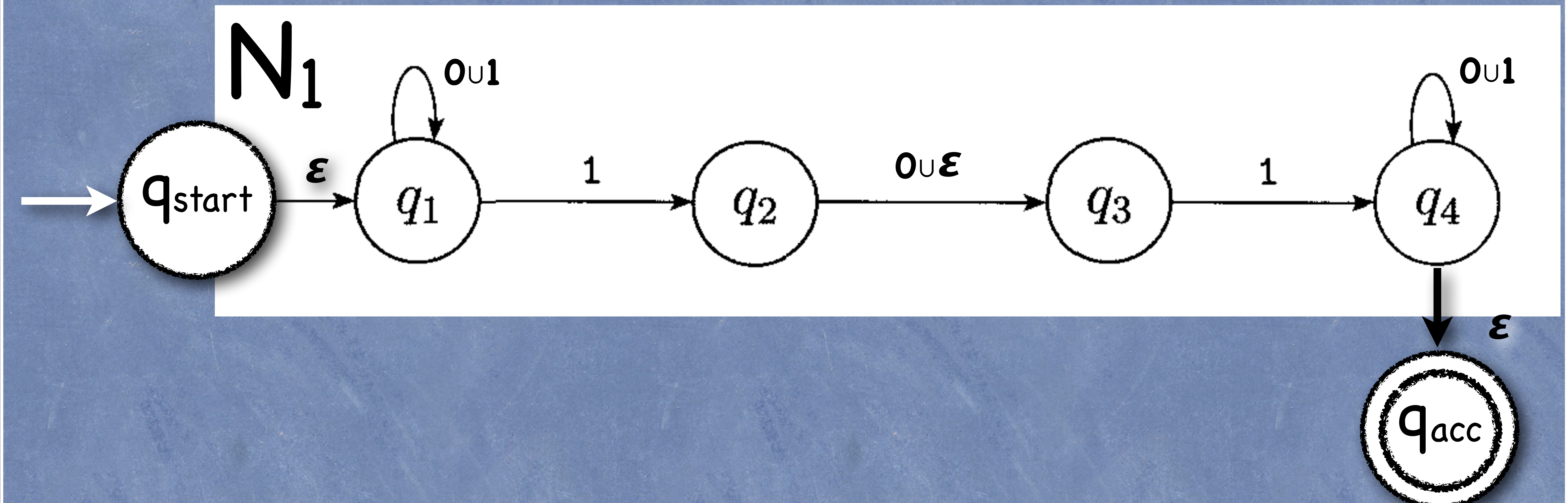
Example NFA→GNFA



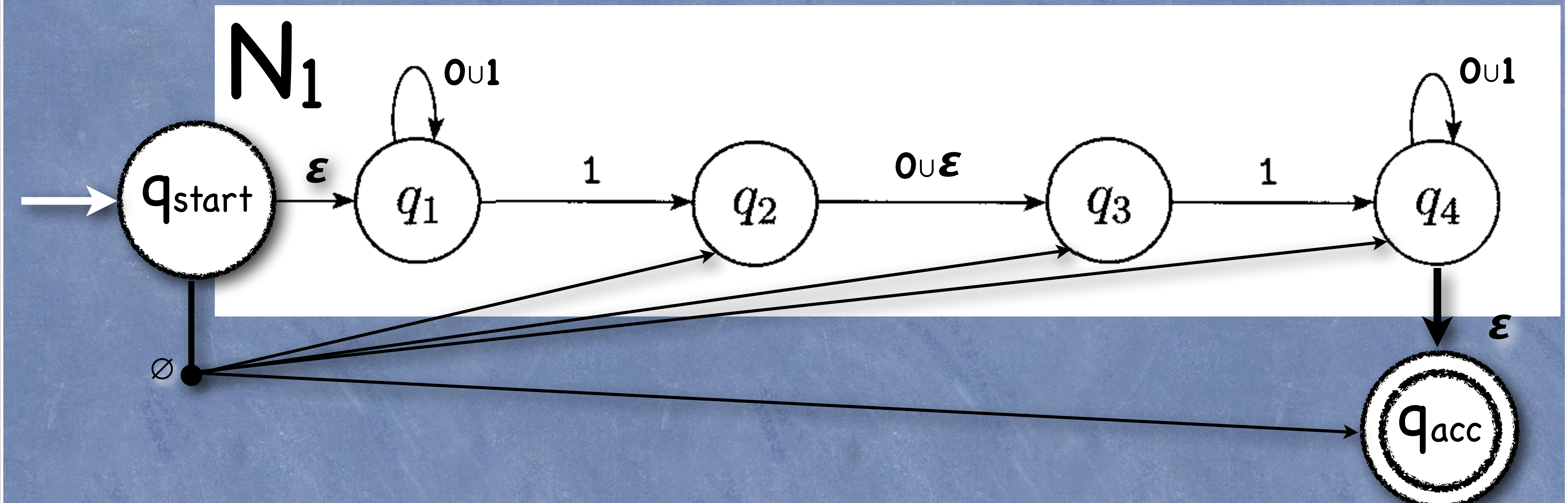
Example NFA→GNFA



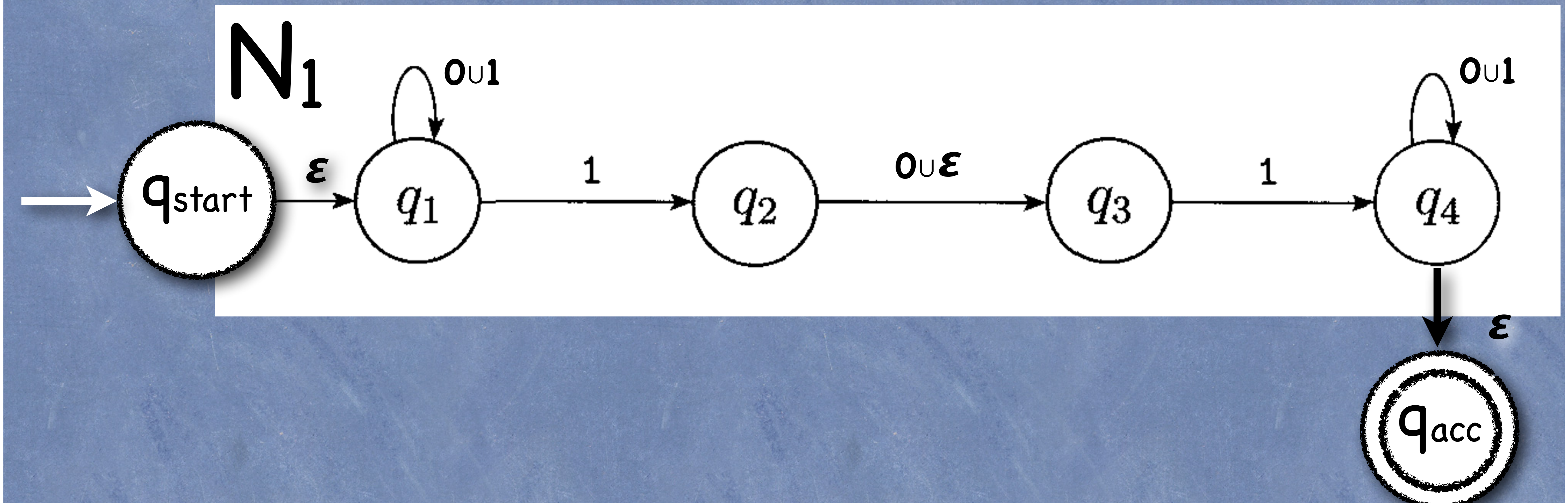
Example NFA→GNFA



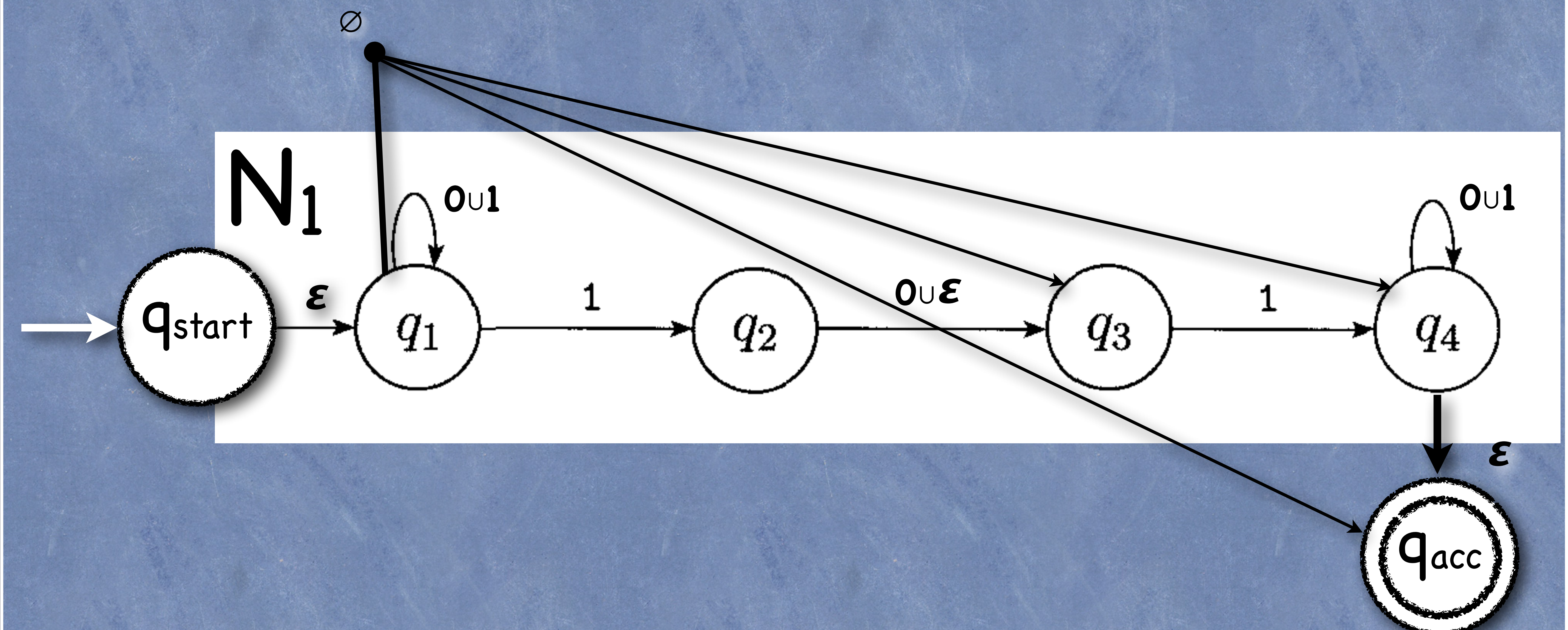
Example NFA→GNFA



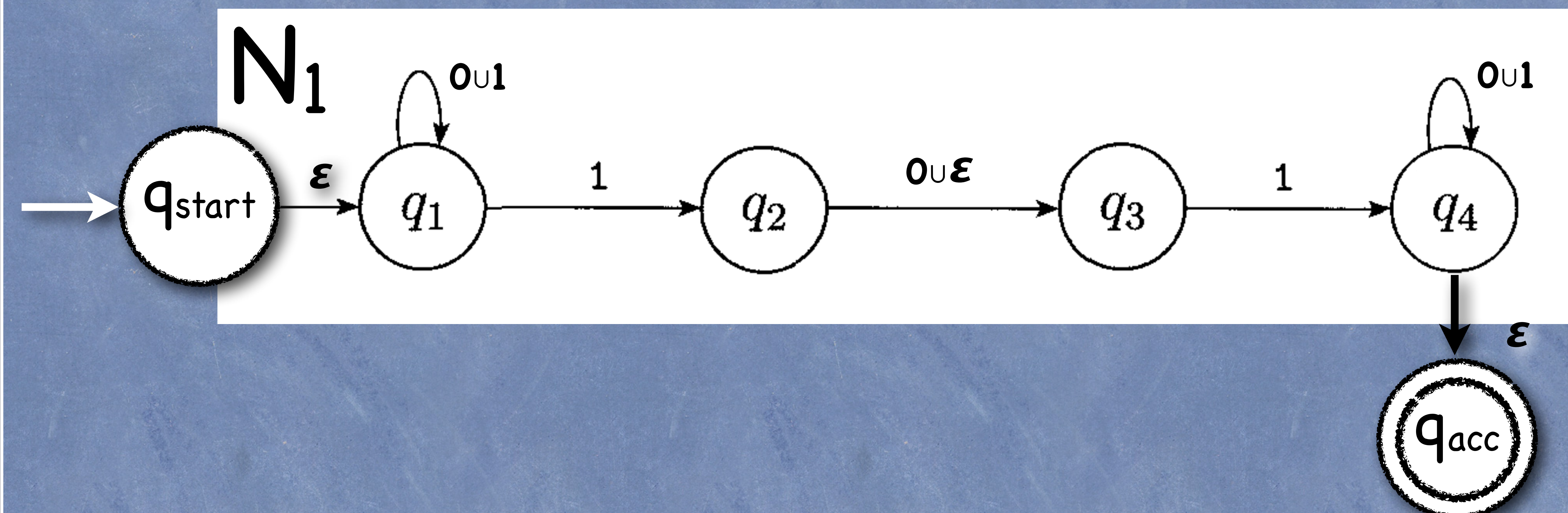
Example NFA→GNFA



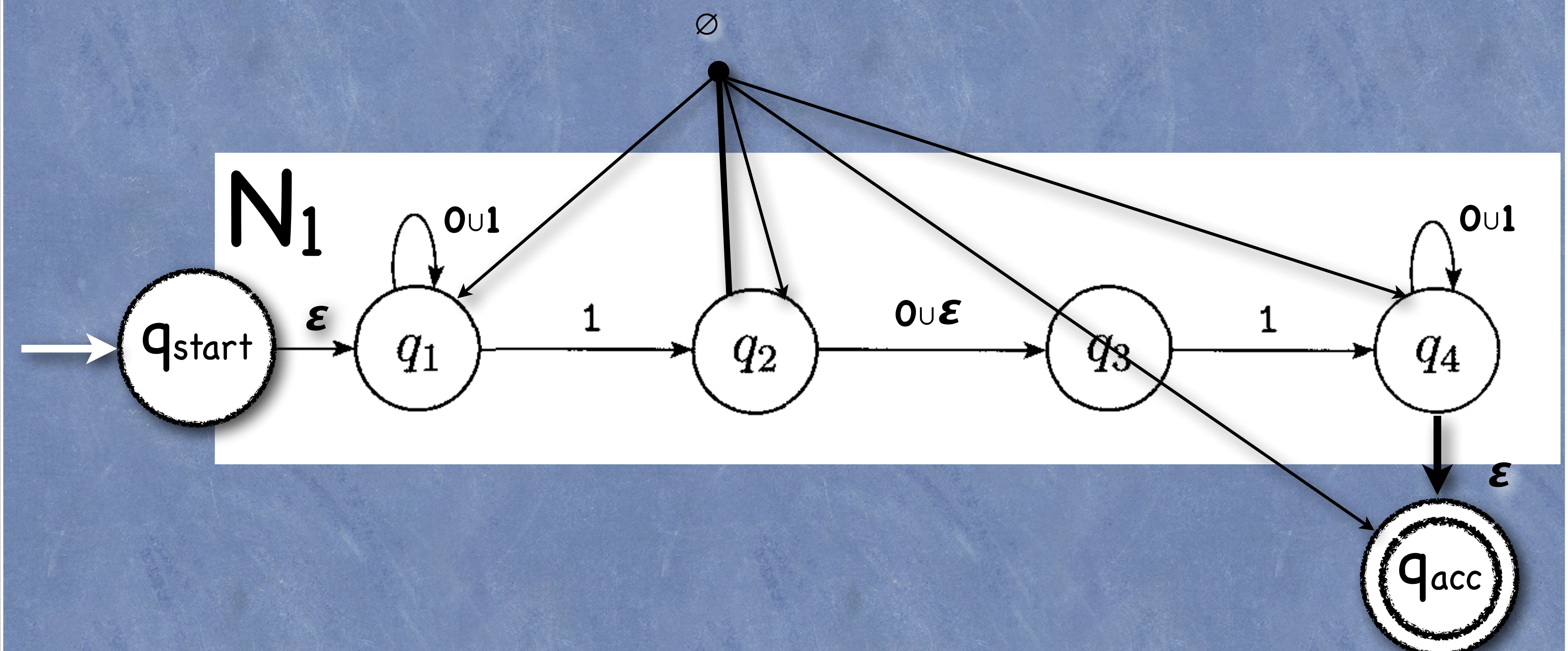
Example NFA→GNFA



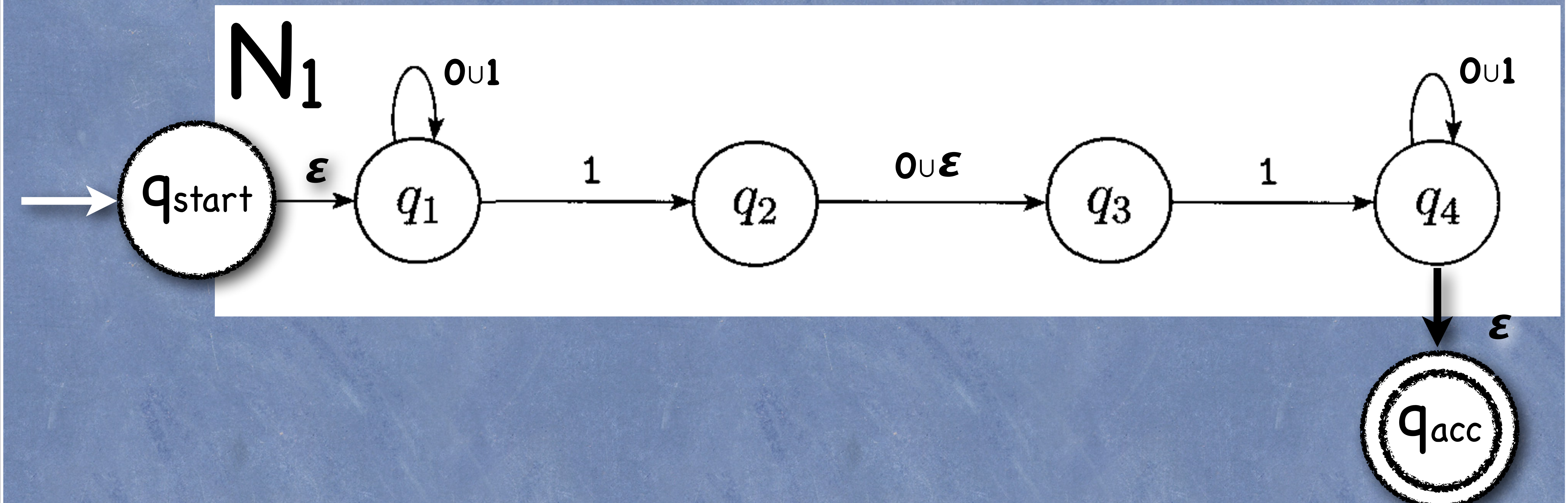
Example NFA→GNFA



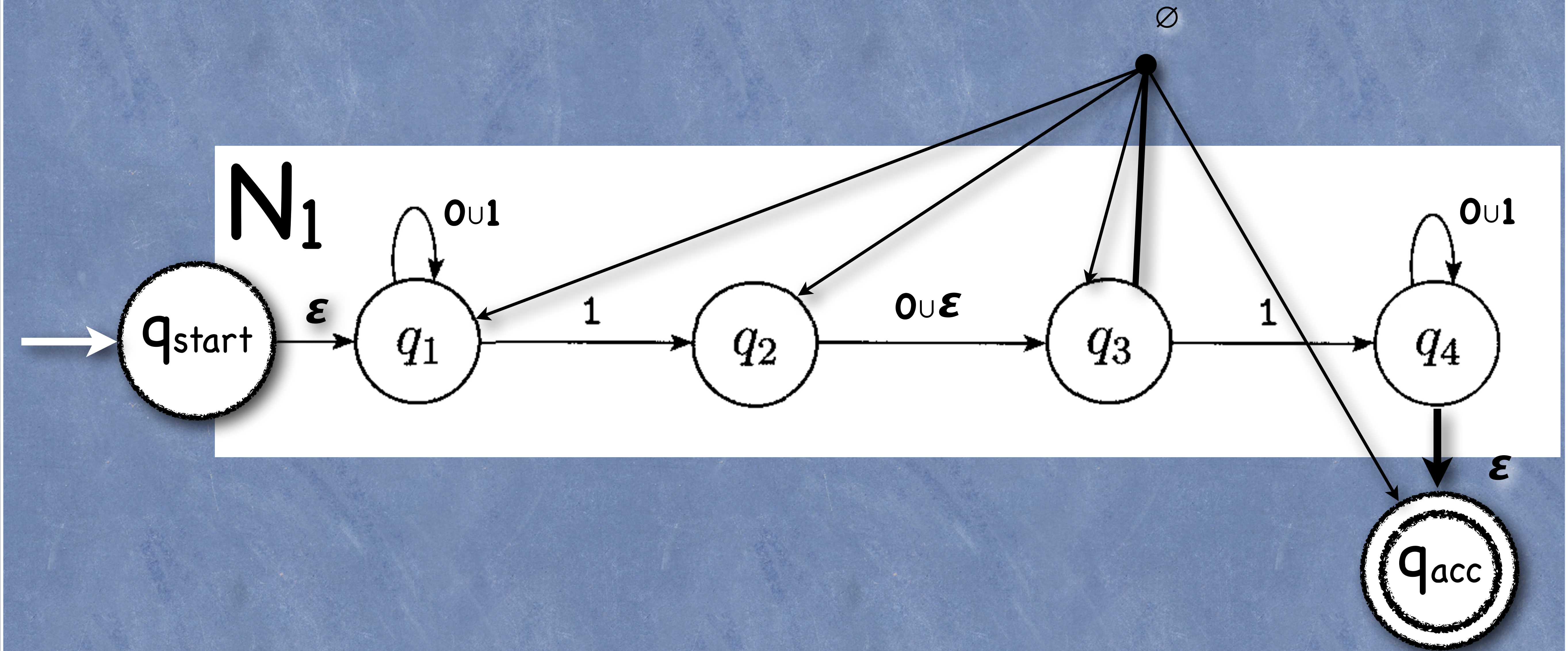
Example NFA→GNFA



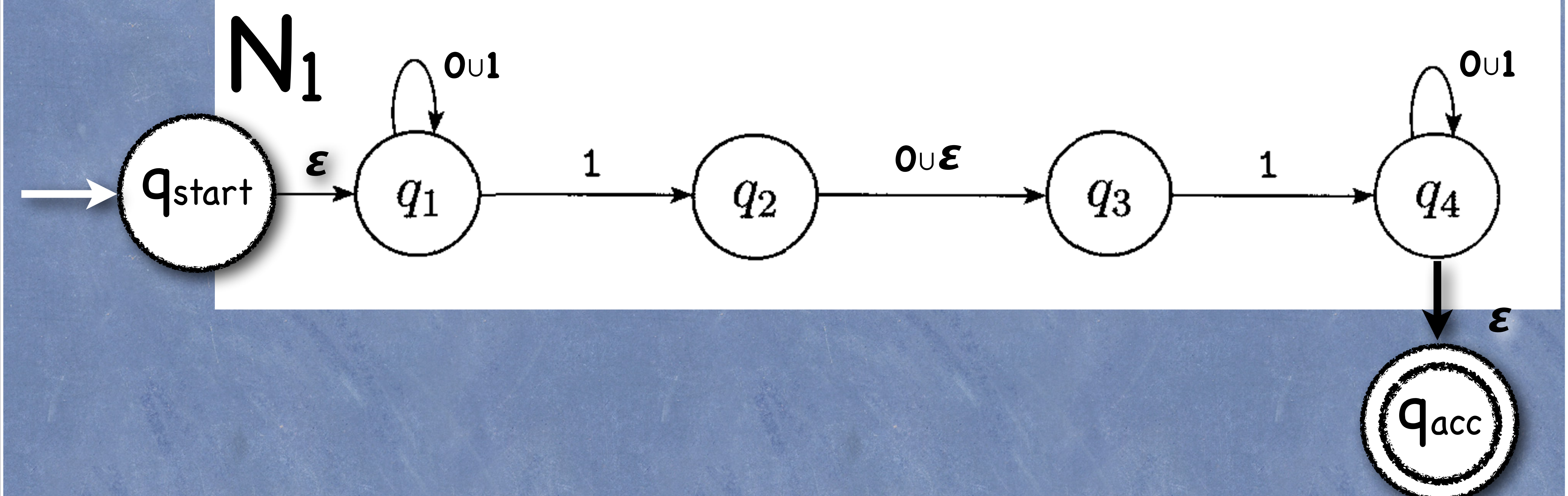
Example NFA→GNFA



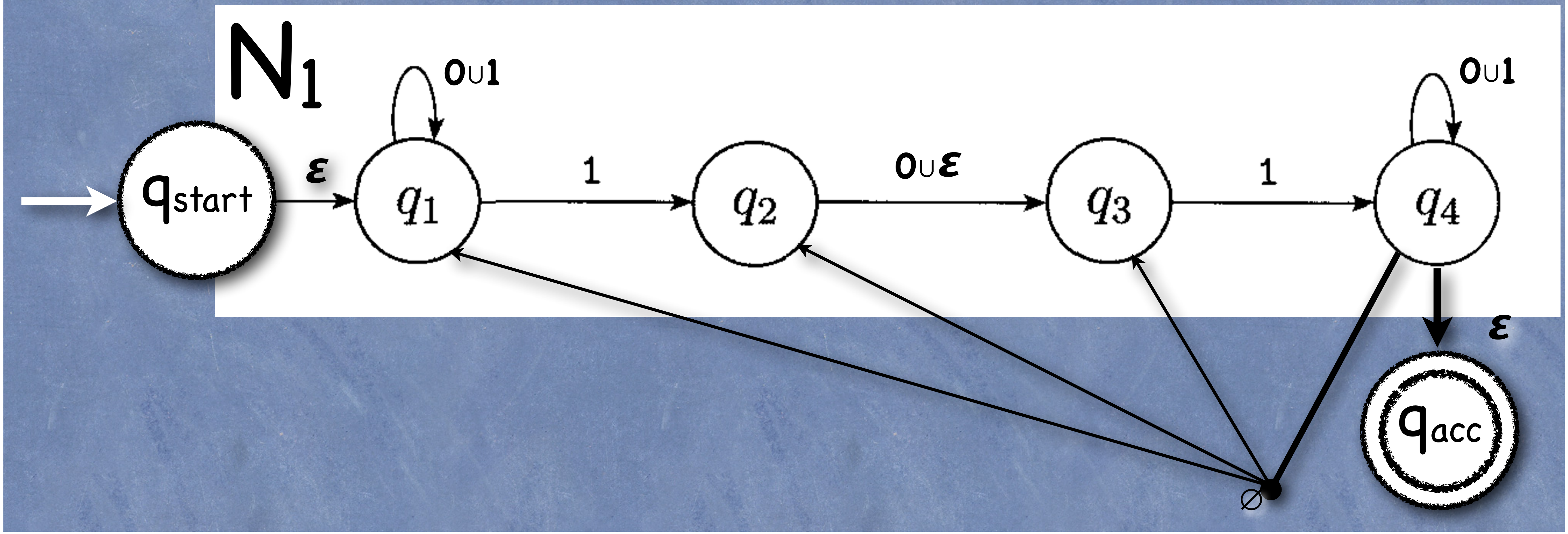
Example NFA→GNFA



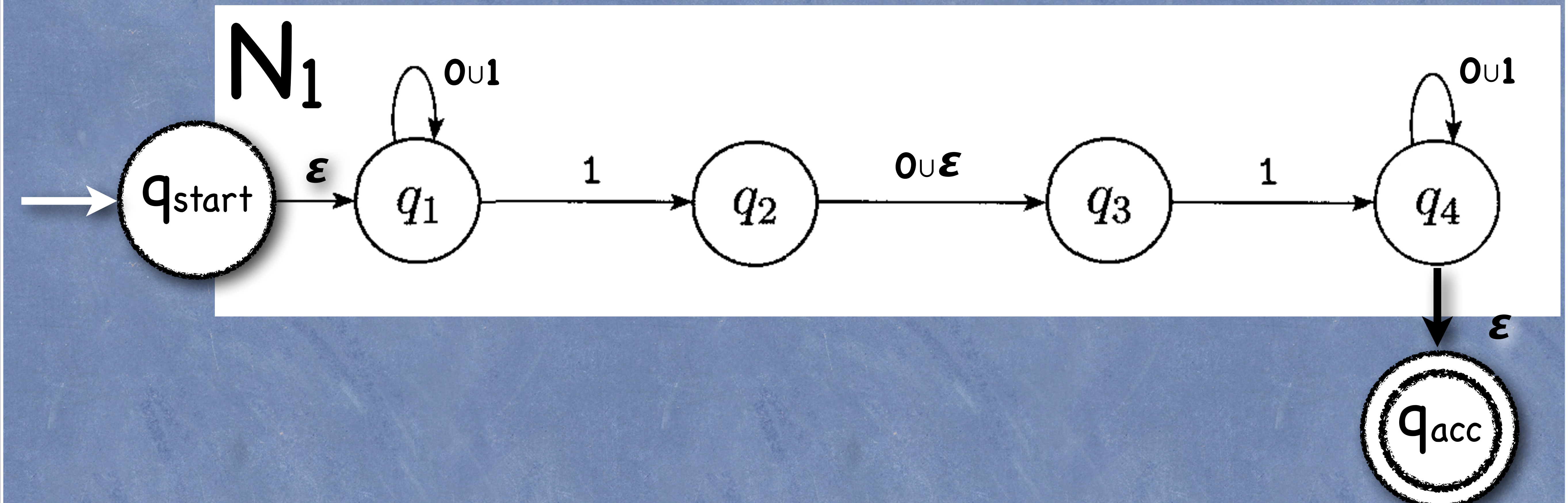
Example NFA→GNFA



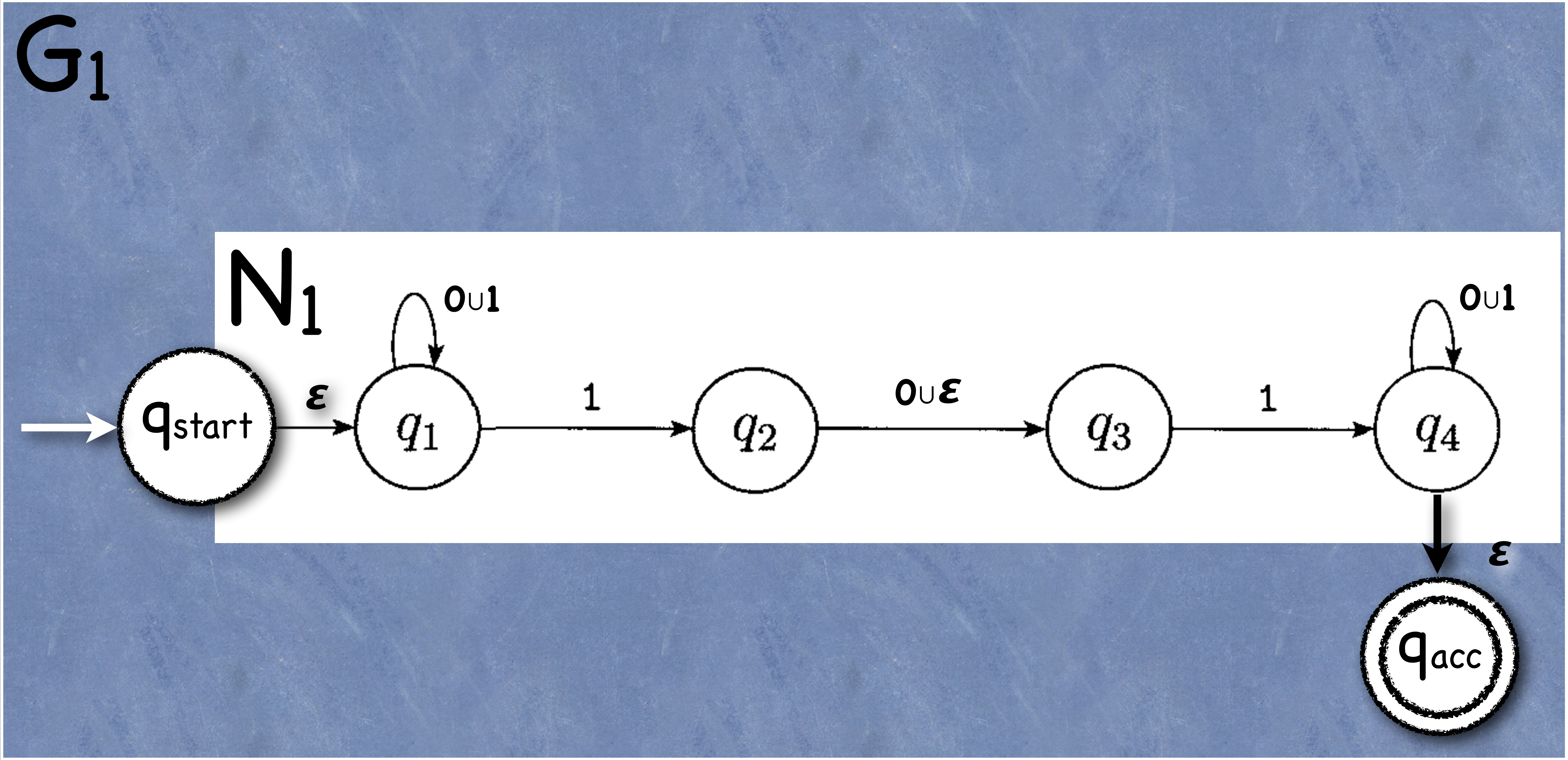
Example NFA→GNFA



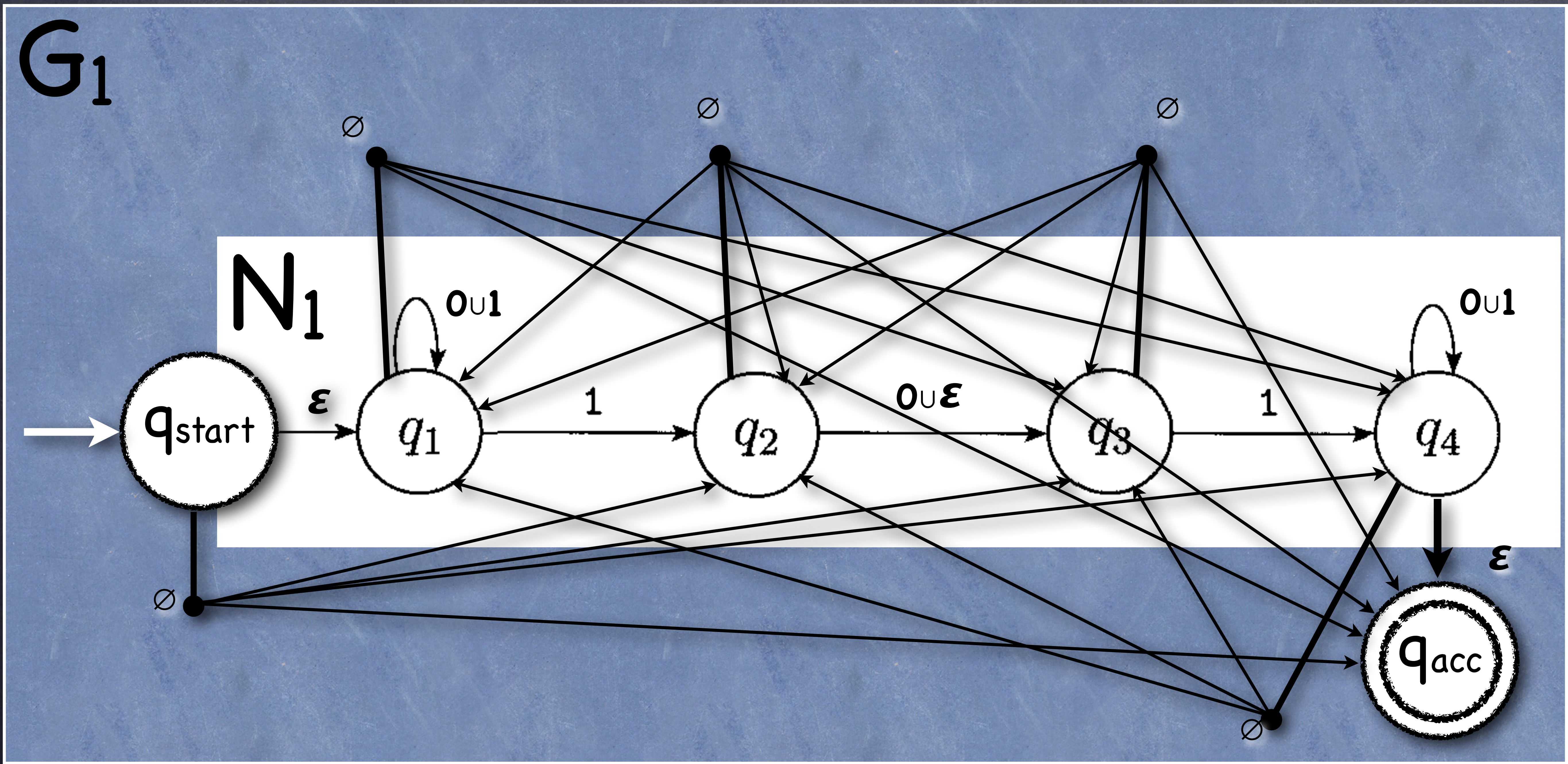
Example NFA→GNFA



Example NFA→GNFA



Example NFA→GNFA



DFA → GNFA → Reg. Exp.

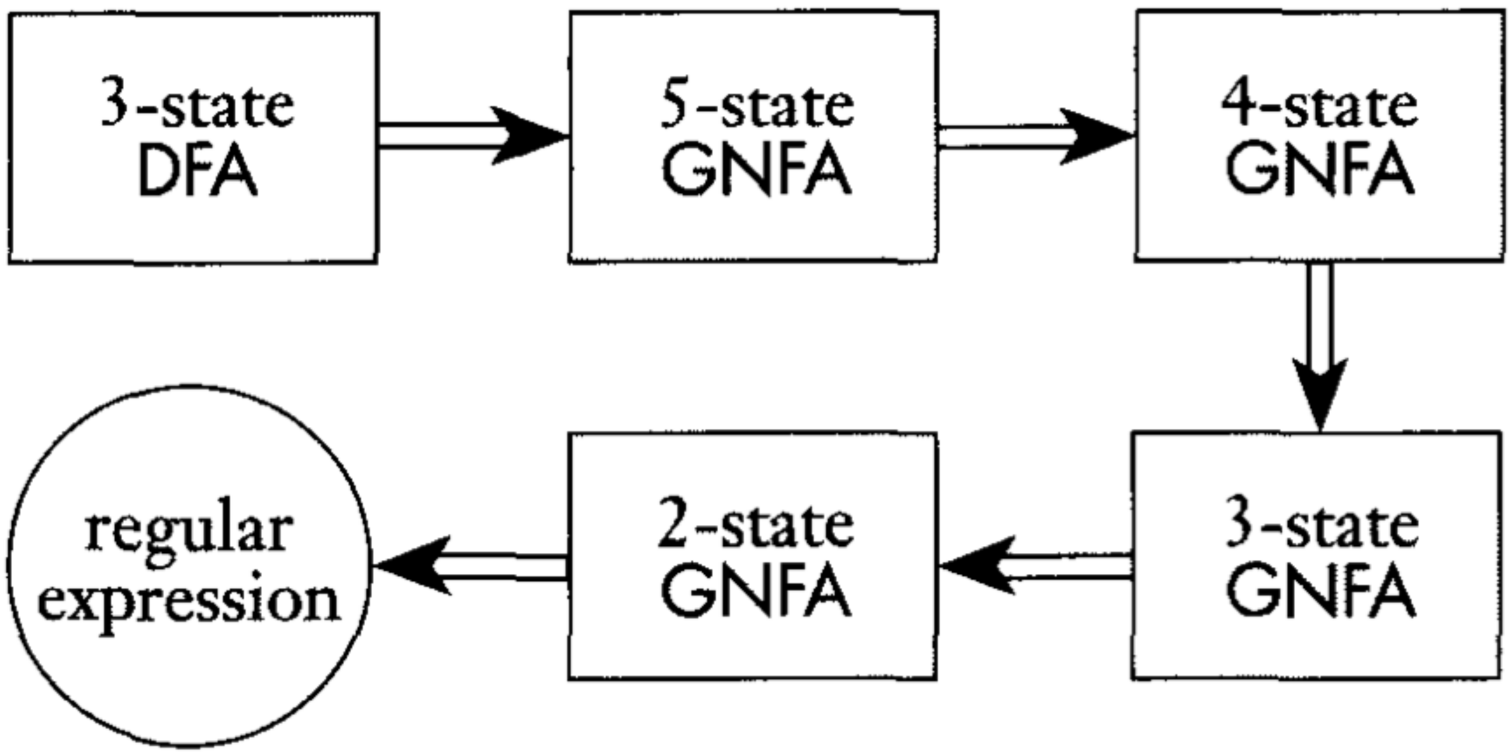


FIGURE 1.62
Typical stages in converting a DFA to a regular expression

Ripping a state

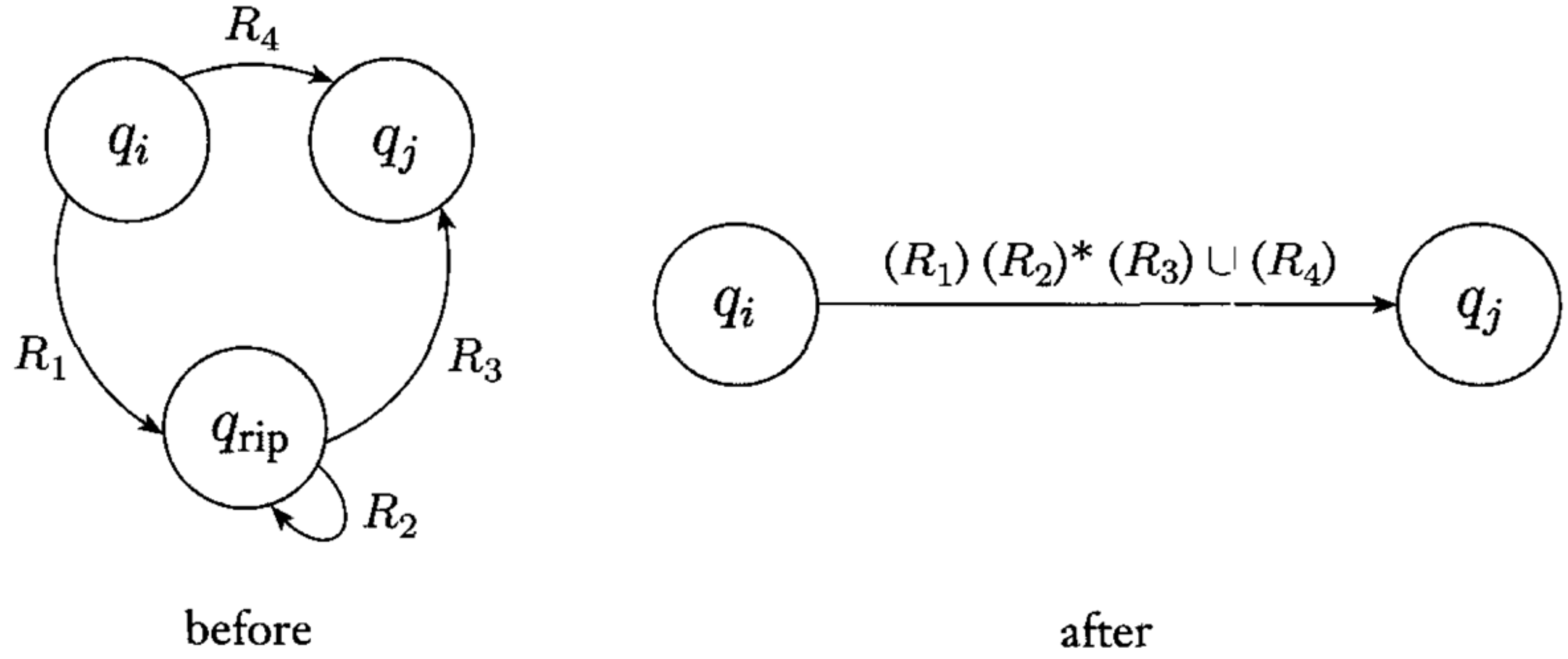
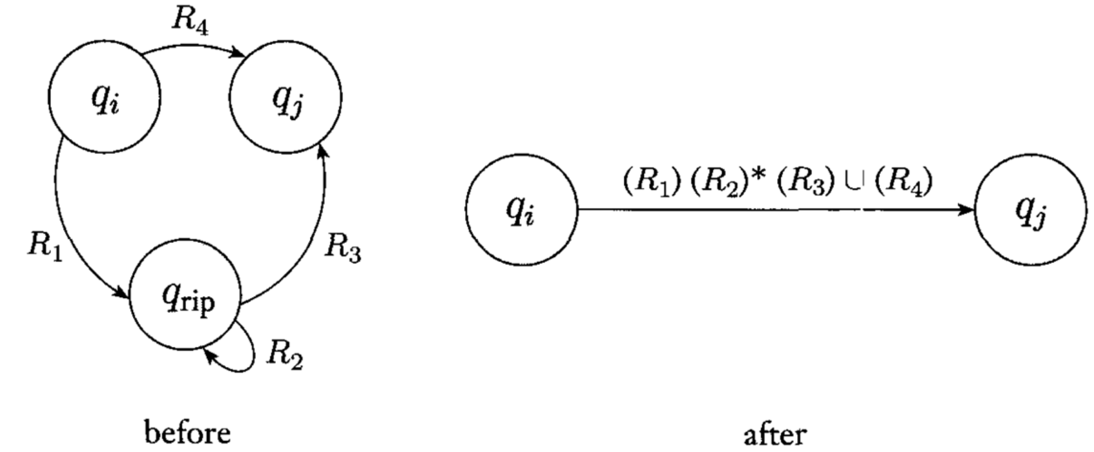


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

GNFA→ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{rip} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where

$$Q' = Q - \{q_{rip}\},$$

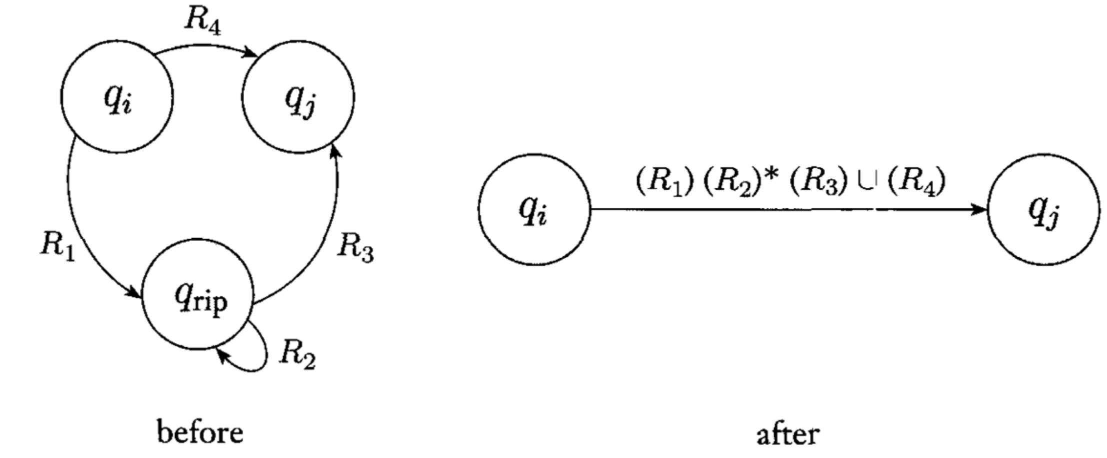
and for any $q_i \in Q' - \{q_{accept}\}$ and any $q_j \in Q' - \{q_{start}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

GNFA→ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{rip} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where

$$Q' = Q - \{q_{rip}\},$$

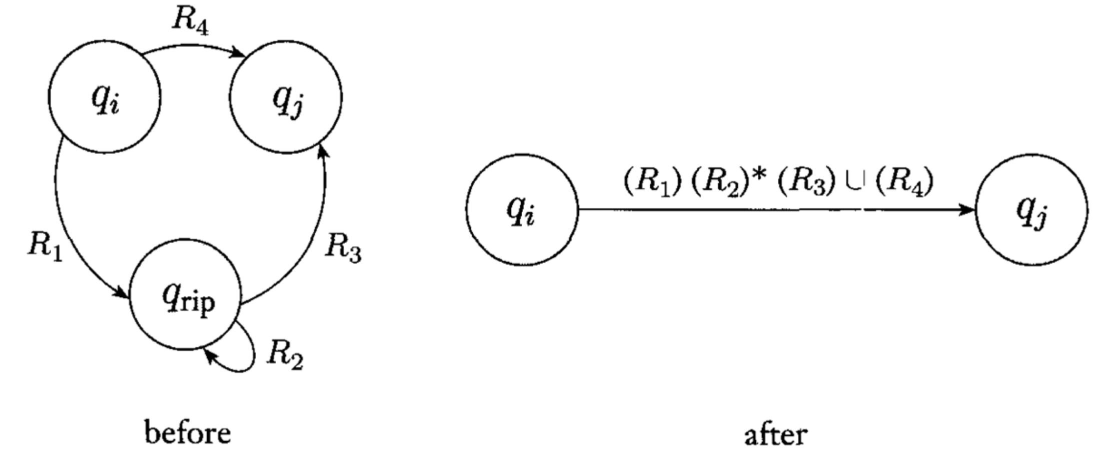
and for any $q_i \in Q' - \{q_{accept}\}$ and any $q_j \in Q' - \{q_{start}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

GNFA→ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{rip} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where

$$Q' = Q - \{q_{rip}\},$$

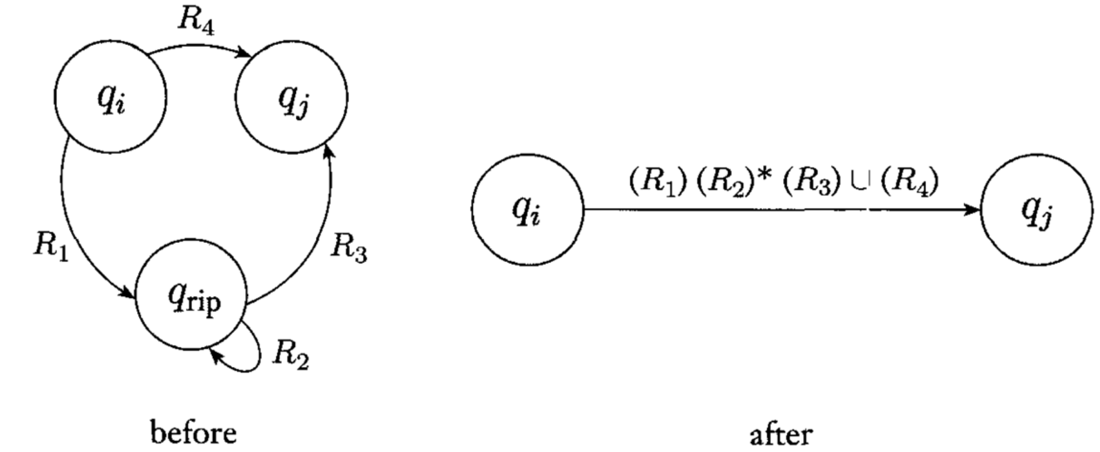
and for any $q_i \in Q' - \{q_{accept}\}$ and any $q_j \in Q' - \{q_{start}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

GNFA→ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where

$$Q' = Q - \{q_{\text{rip}}\},$$

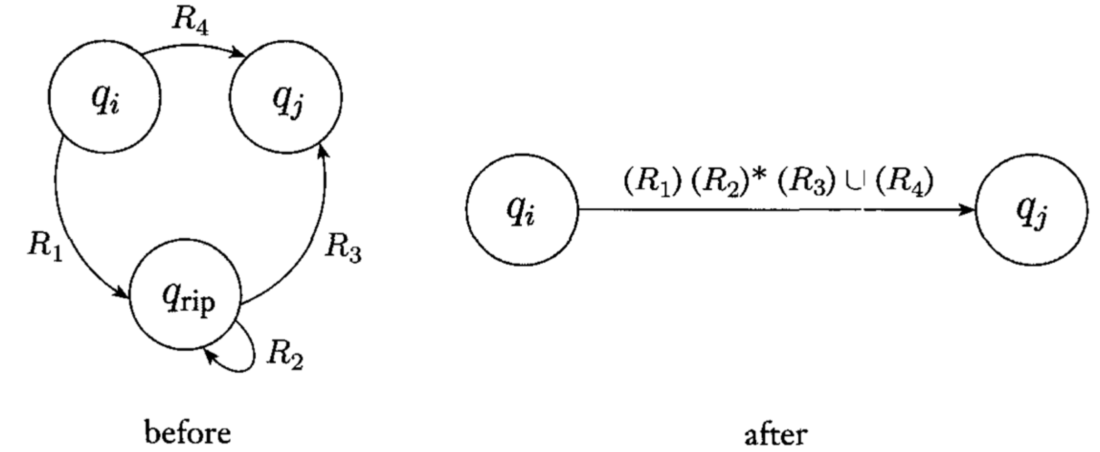
and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

GNFA→ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{rip} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where

$$Q' = Q - \{q_{rip}\},$$

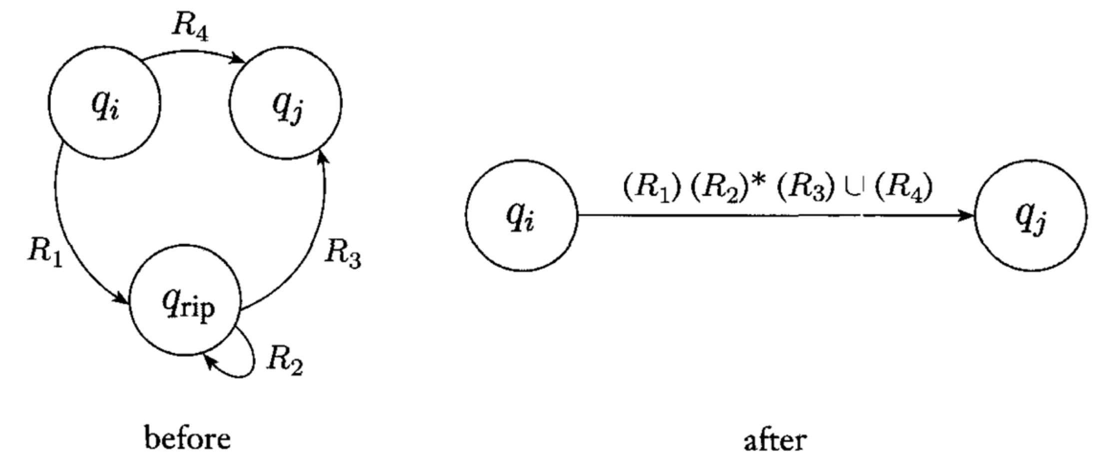
and for any $q_i \in Q' - \{q_{accept}\}$ and any $q_j \in Q' - \{q_{start}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{rip})$, $R_2 = \delta(q_{rip}, q_{rip})$, $R_3 = \delta(q_{rip}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

GNFA→ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where

$$Q' = Q - \{q_{\text{rip}}\},$$

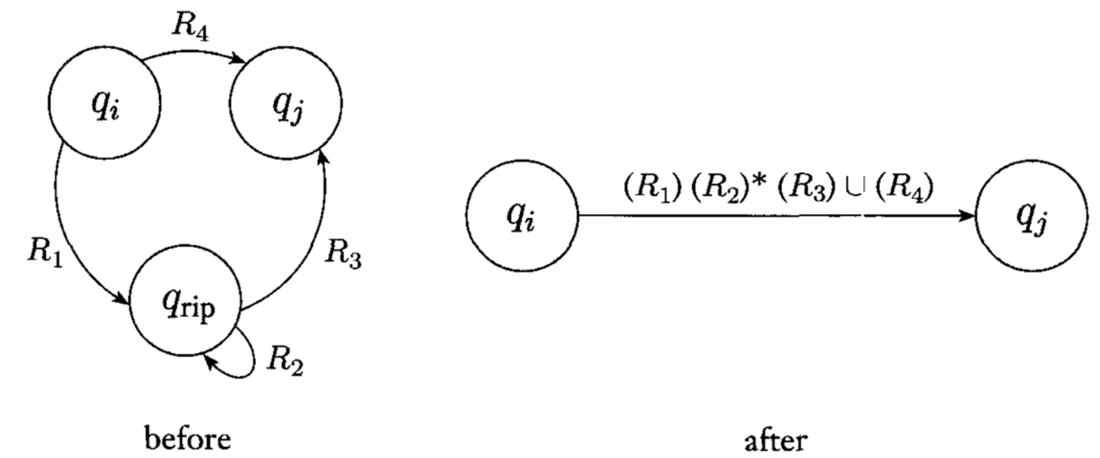
and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

GNFA→ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where

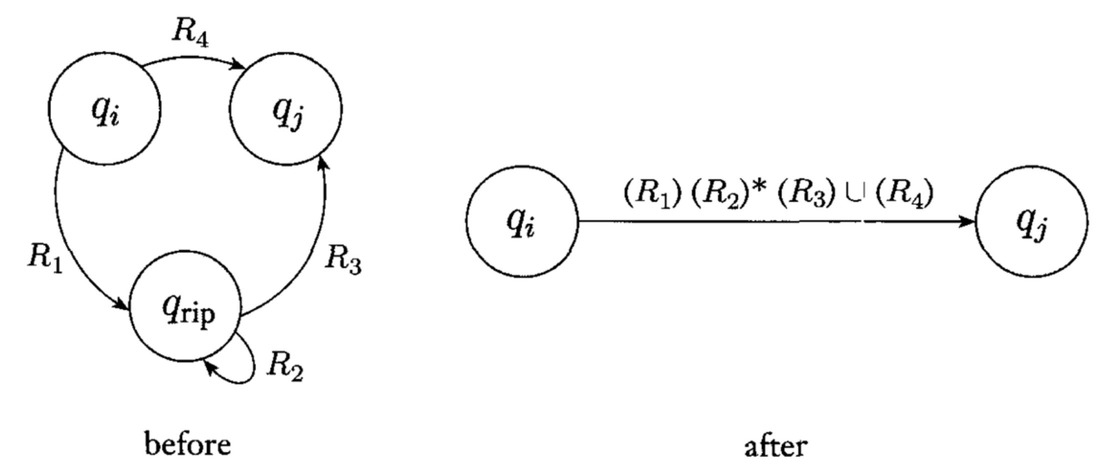
$$Q' = Q - \{q_{\text{rip}}\},$$

and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.
4. Compute CONVERT(G') and return this value.

GNFA⟶ Reg. Exp.



CONVERT(G):

1. Let k be the number of states of G.
2. If $k = 2$, then G must consist of a start state, an accept state, and a single arrow connecting them and labeled with a regular expression R. Return the expression R.
3. If $k > 2$, we select any state $q_{\text{rip}} \in Q$ different from q_{start} and q_{accept} and let G' be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$, where

$$Q' = Q - \{q_{\text{rip}}\},$$

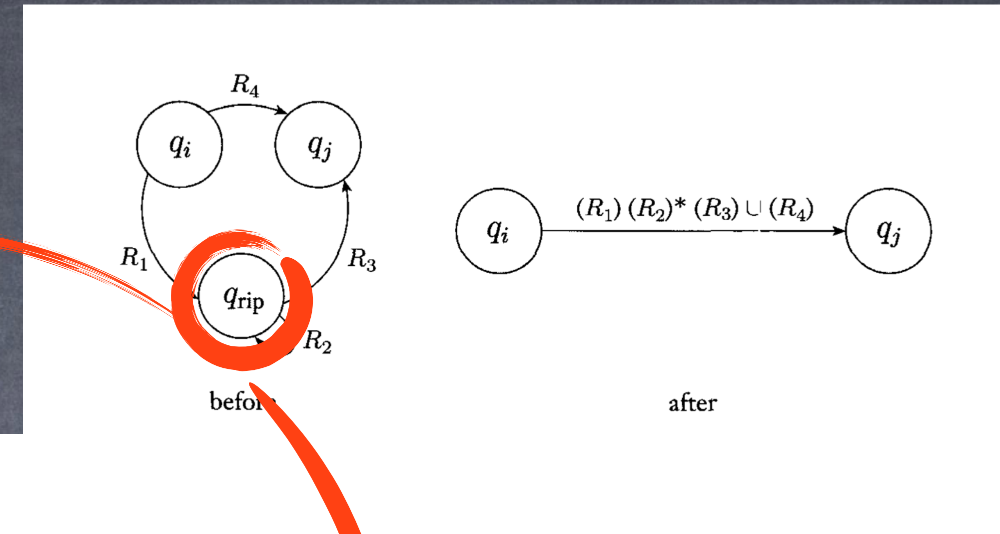
and for any $q_i \in Q' - \{q_{\text{accept}}\}$ and any $q_j \in Q' - \{q_{\text{start}}\}$ let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$, and $R_4 = \delta(q_i, q_j)$.

4. Compute CONVERT(G') and return this value.

Ripping of a GNFA



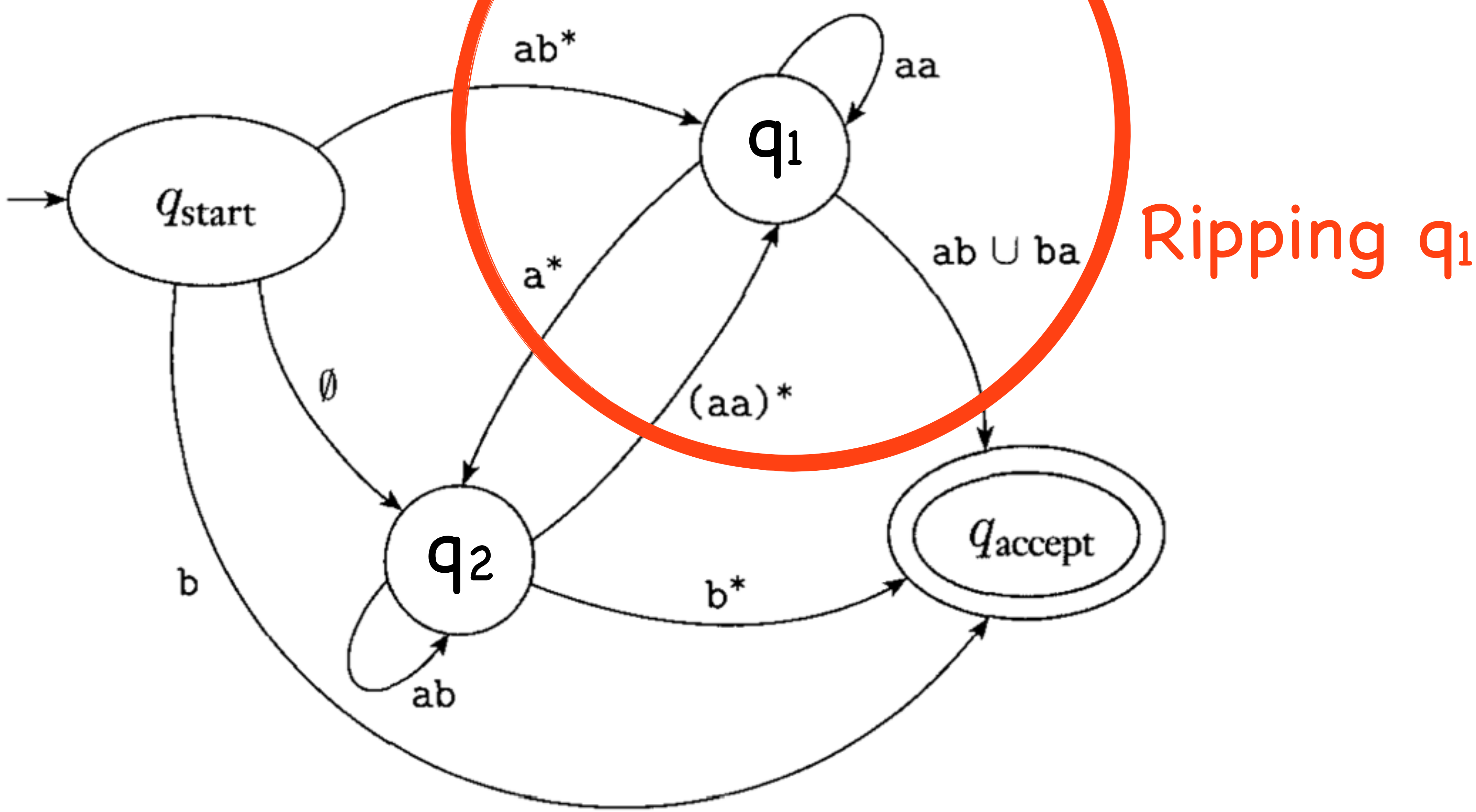
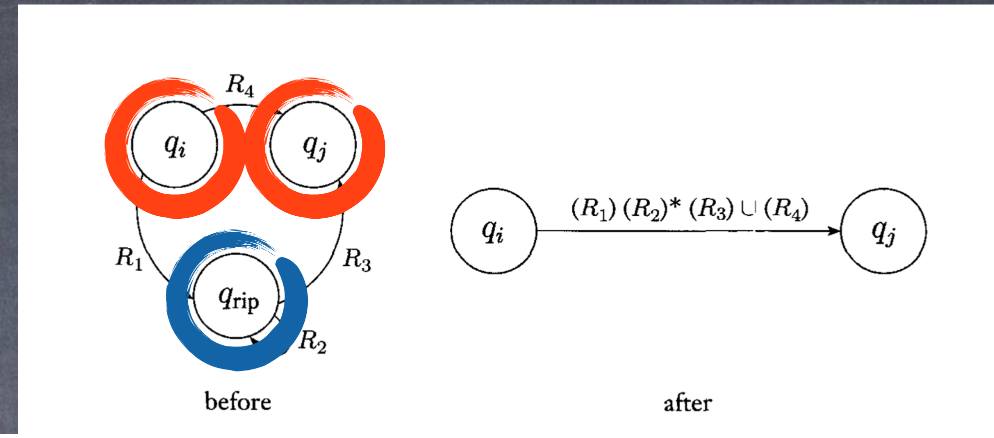
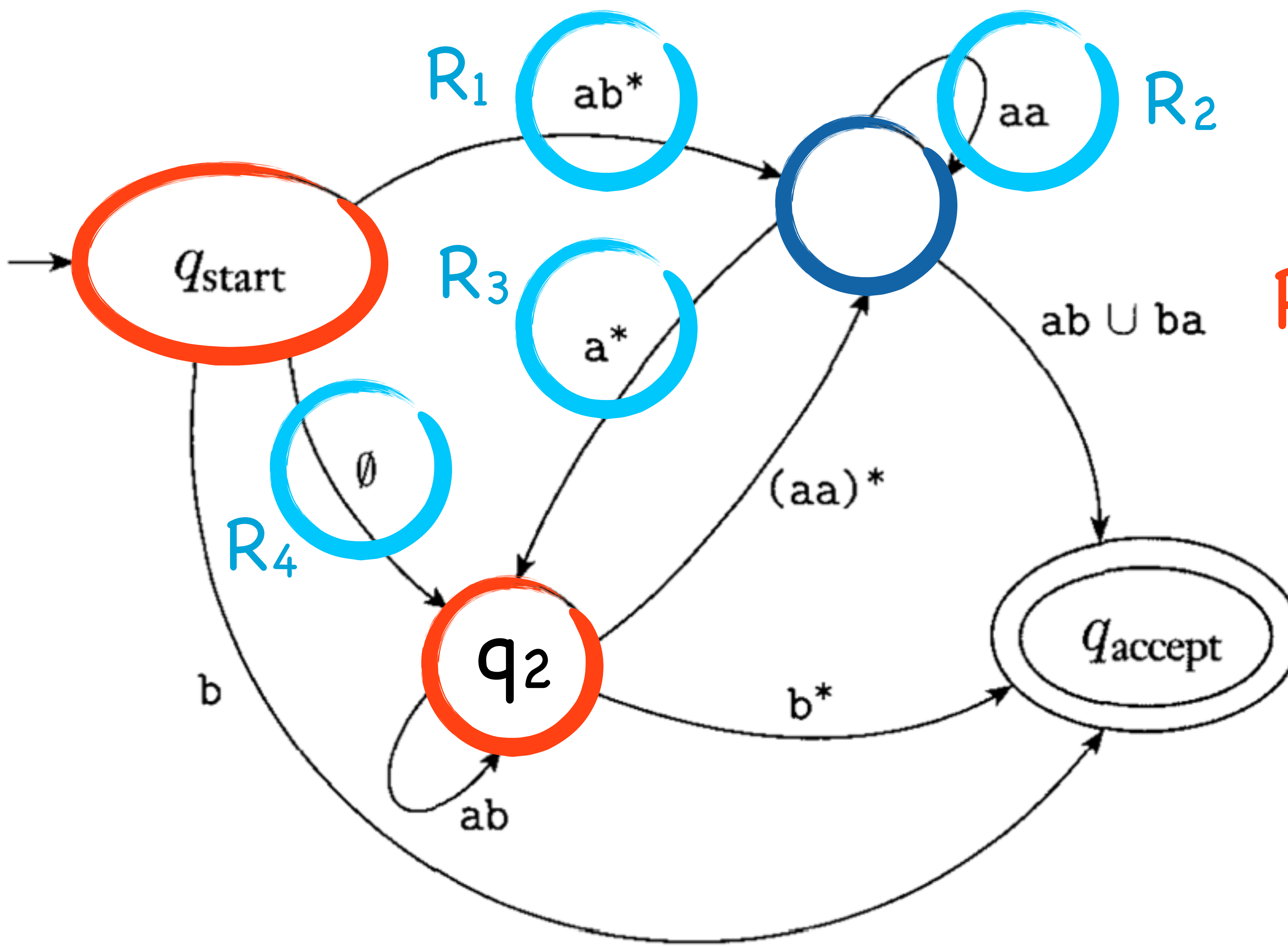


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA

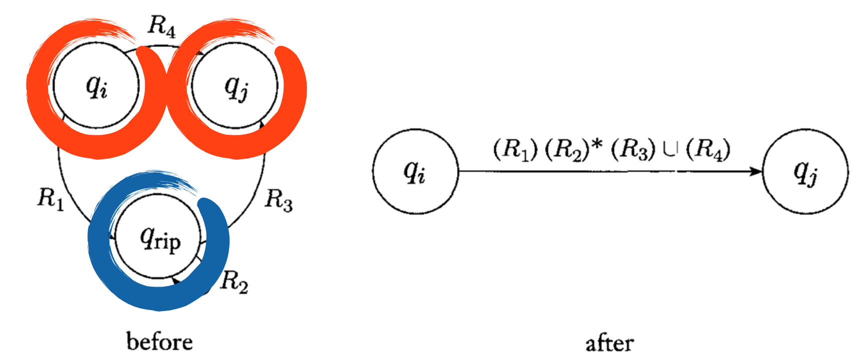


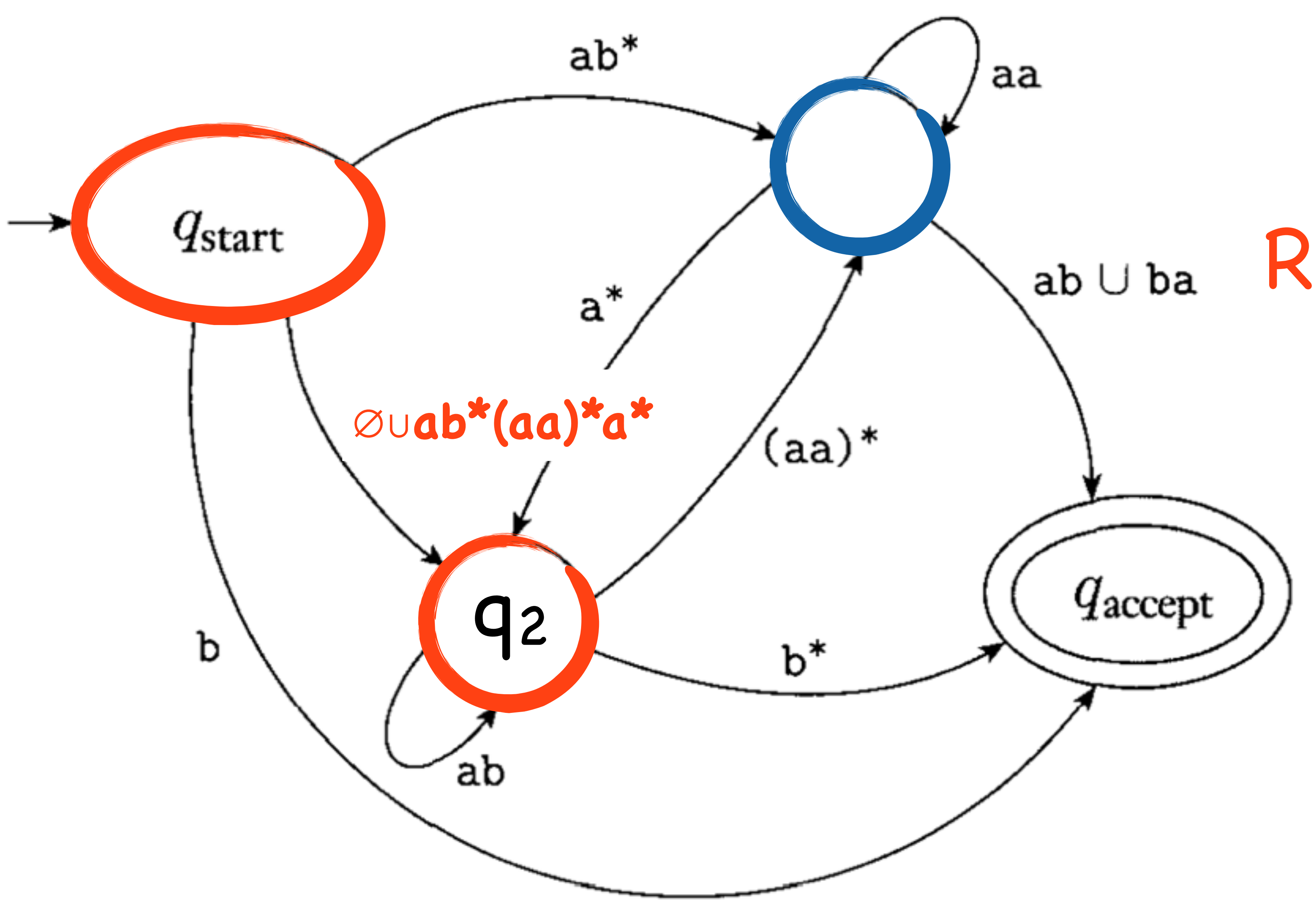


Ripping q1

FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA

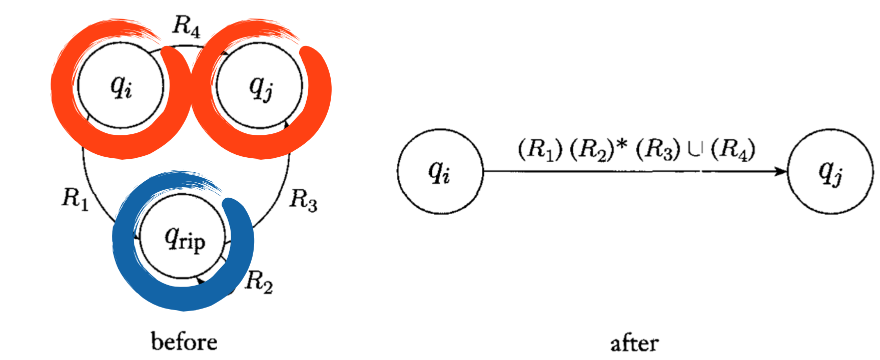




Ripping q_1

FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



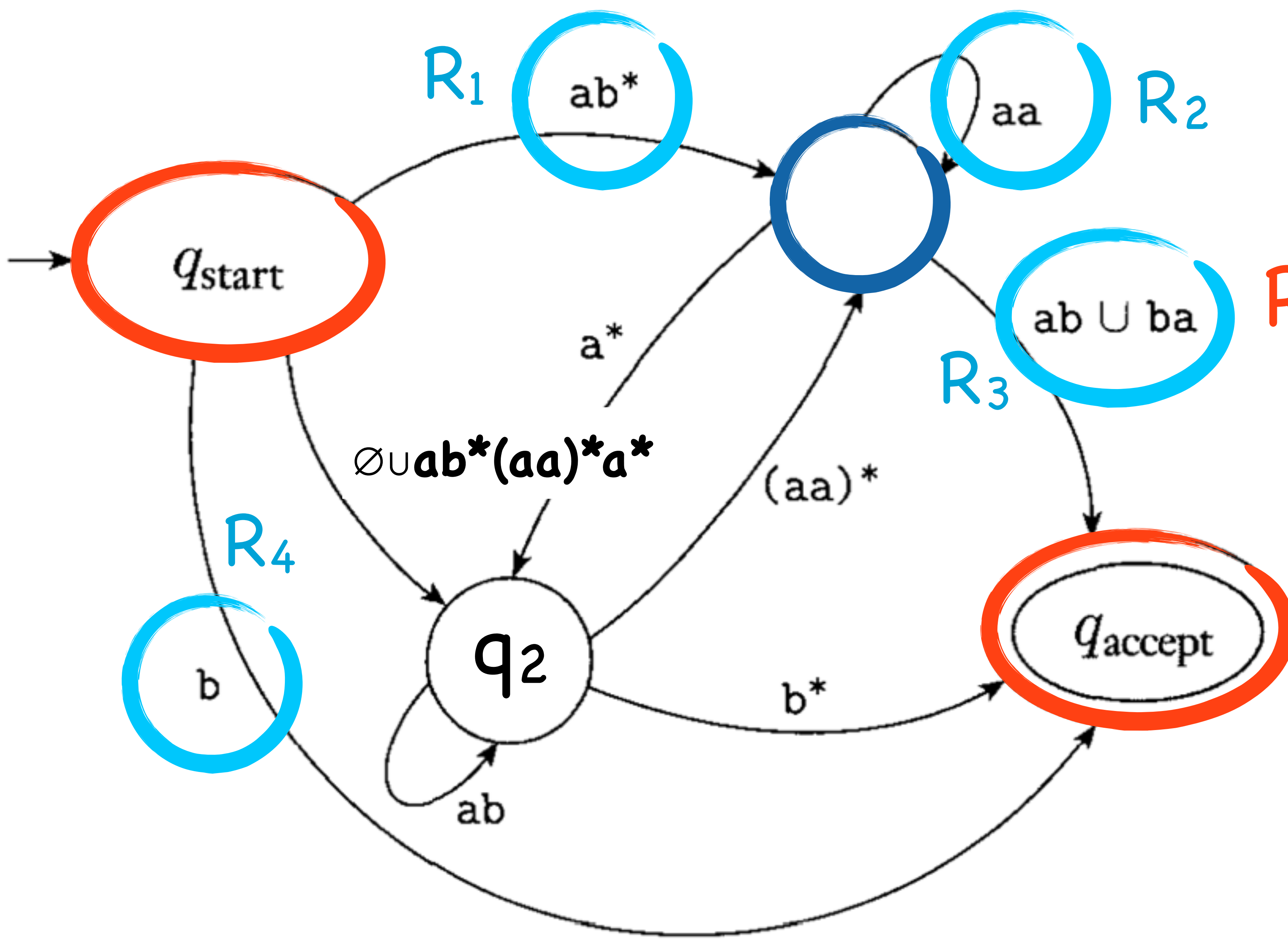
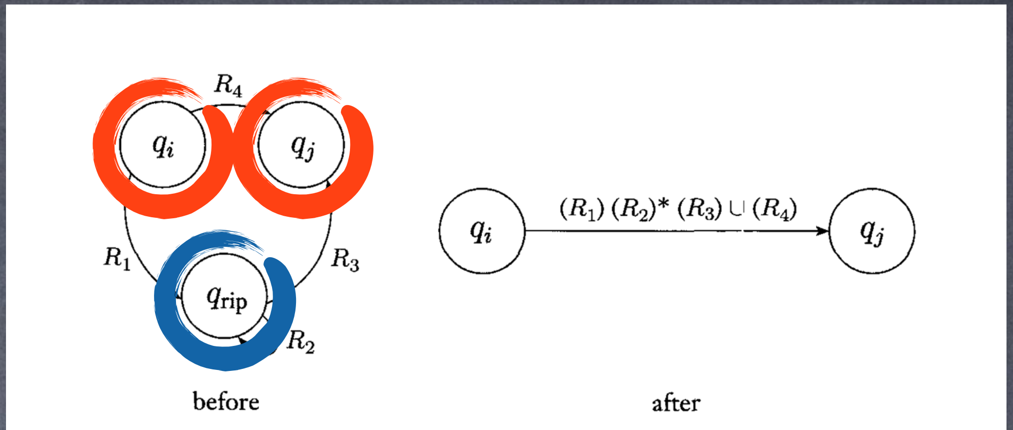
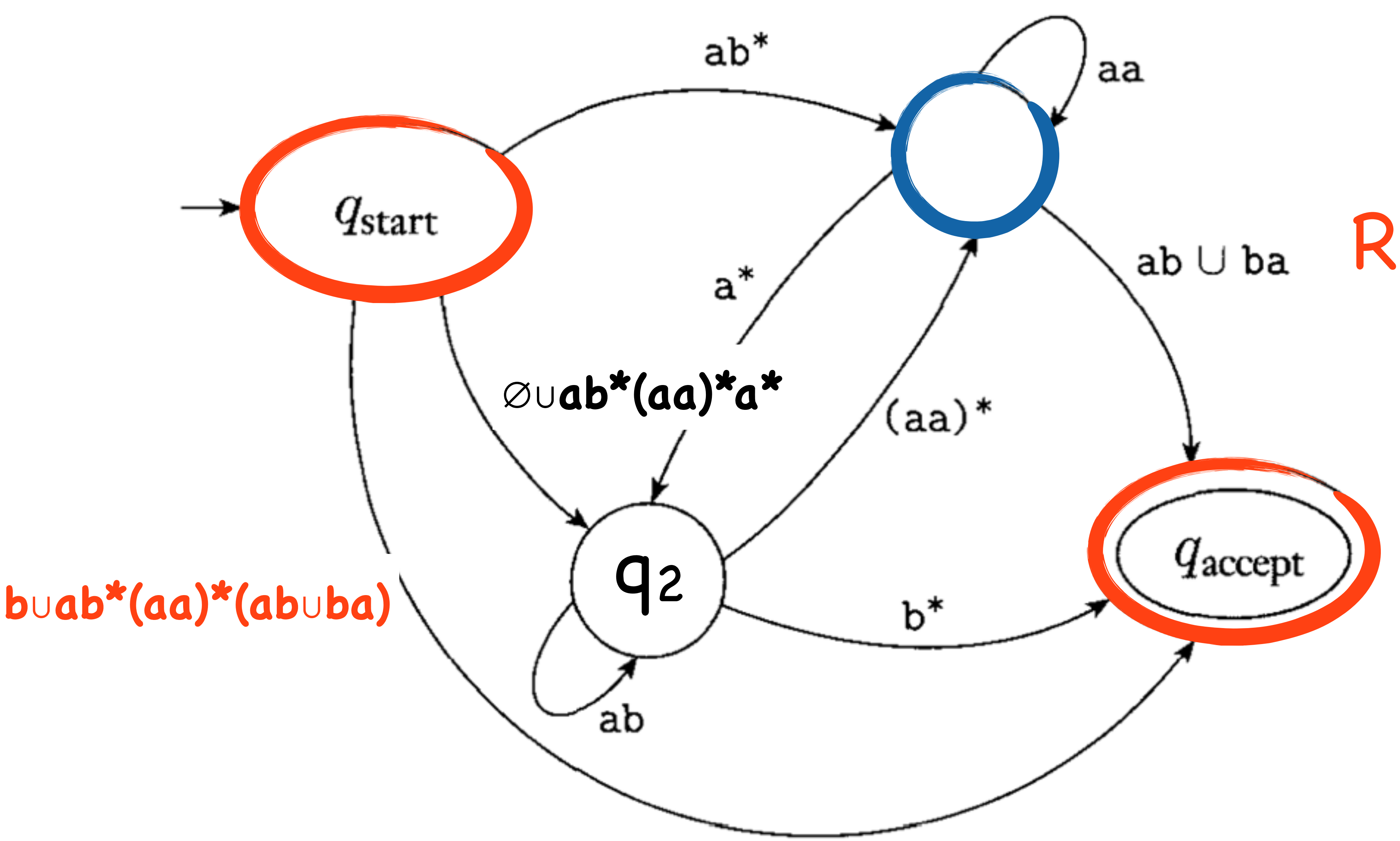


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA

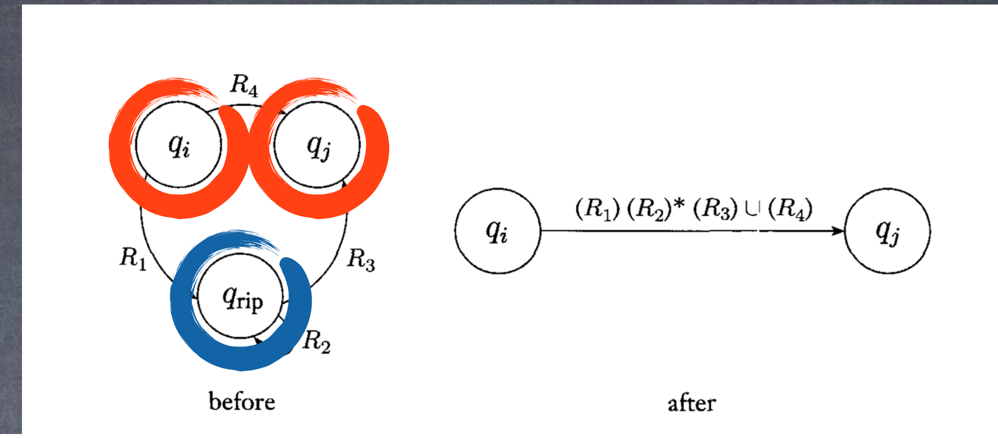




Ripping q_1

FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



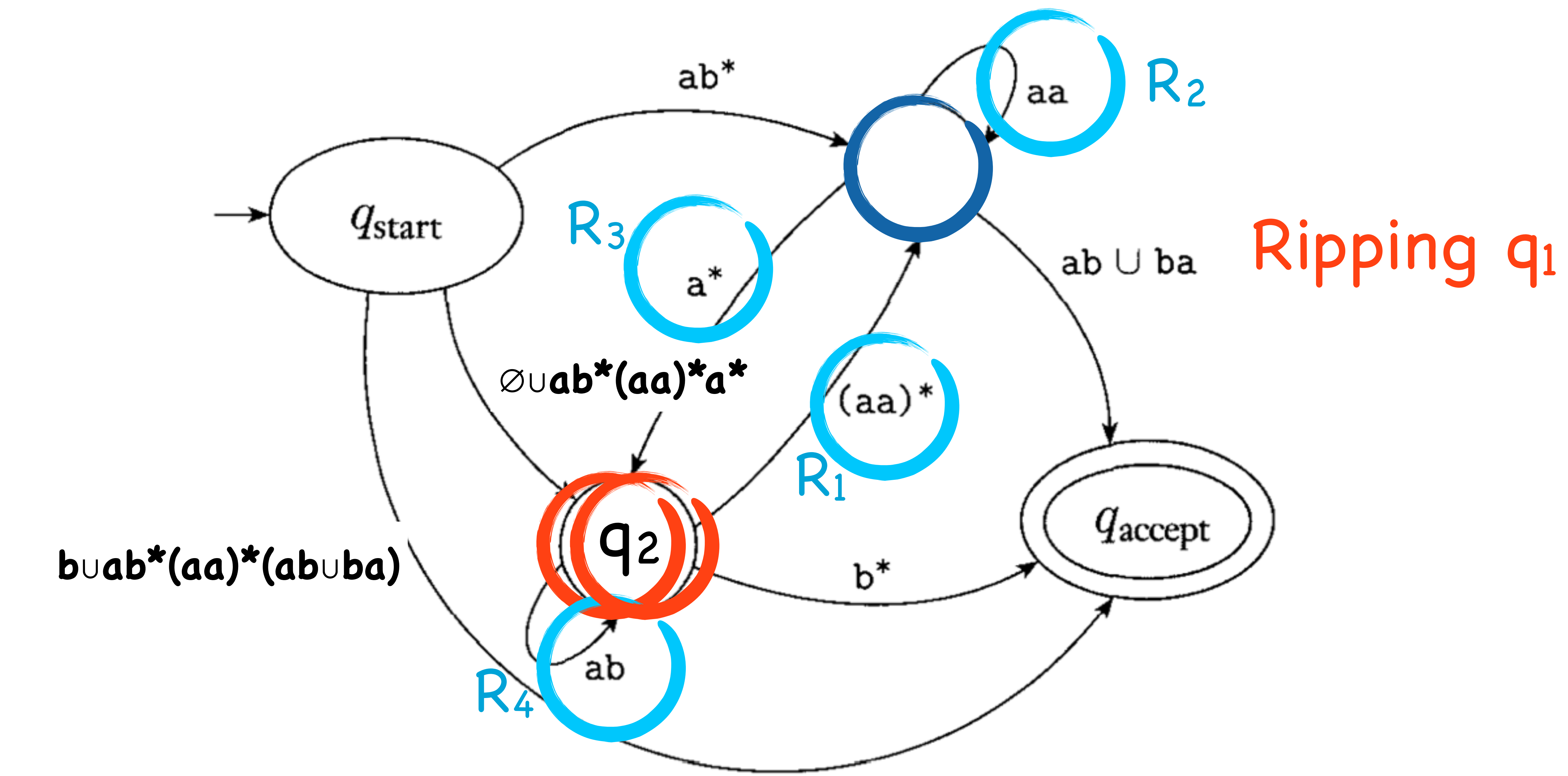
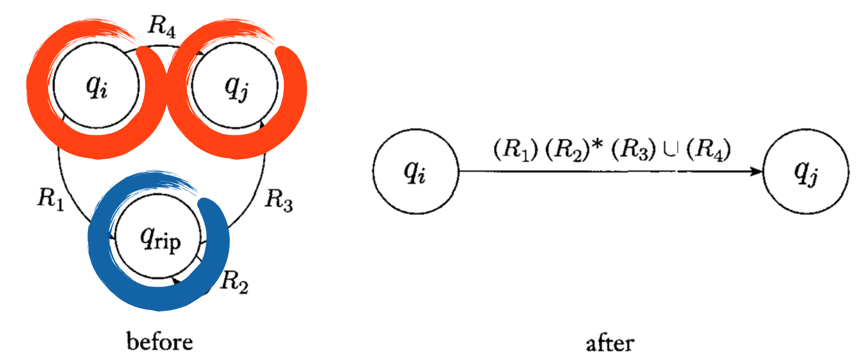
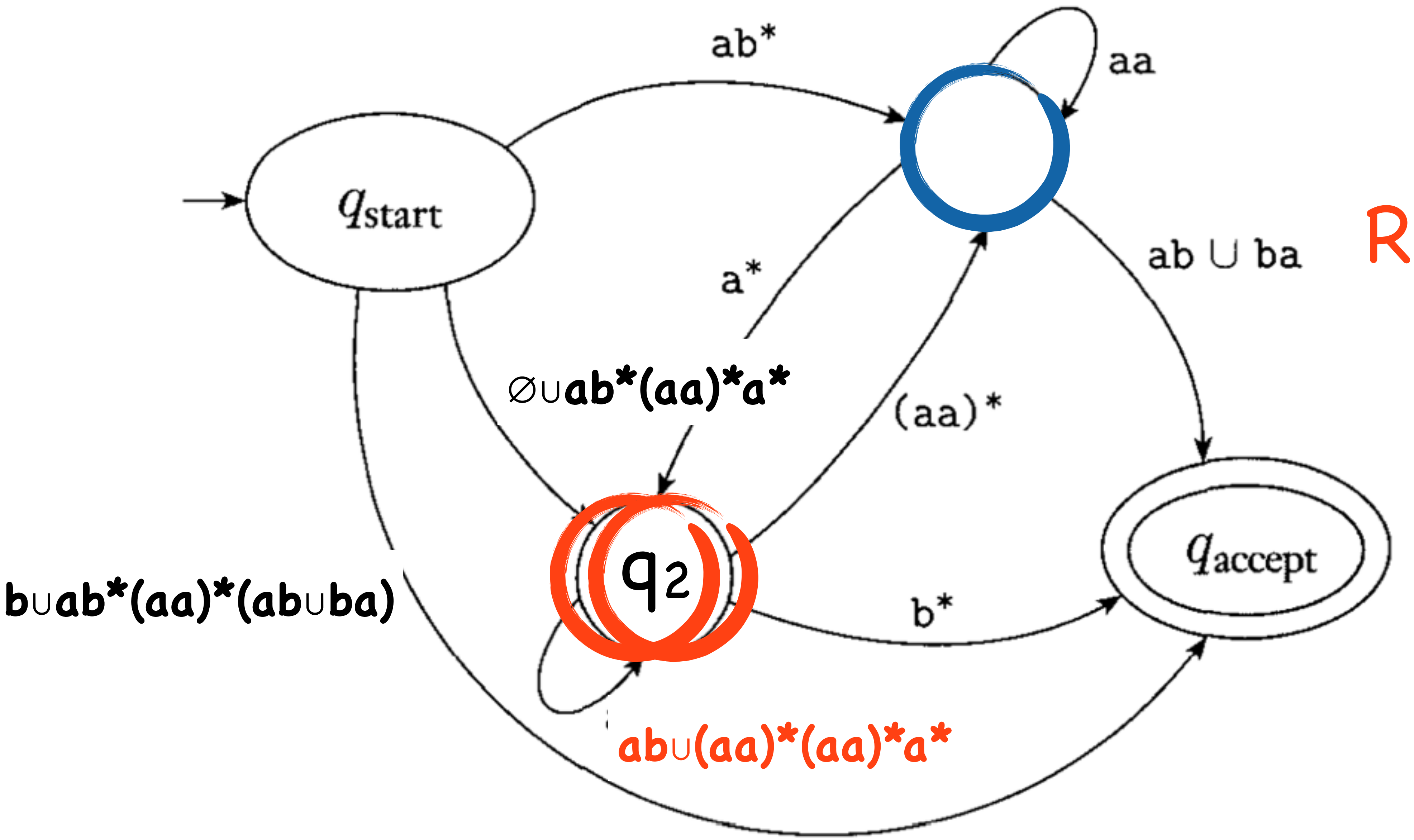


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA

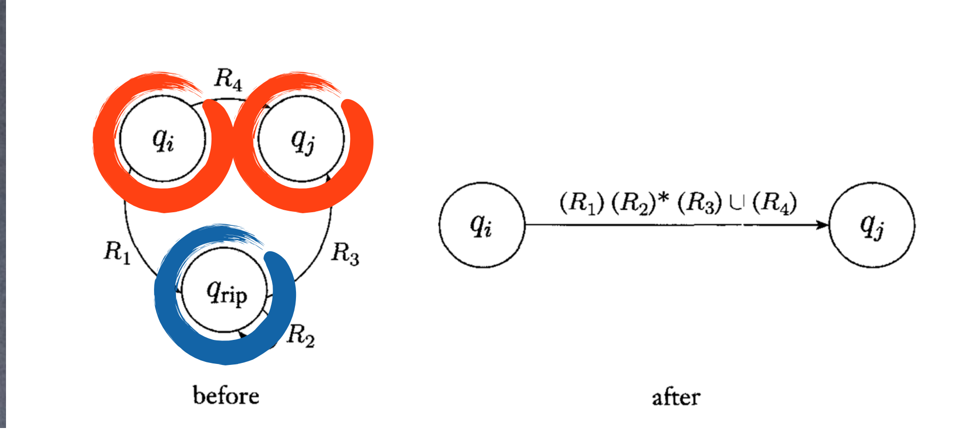




Ripping q_1

FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



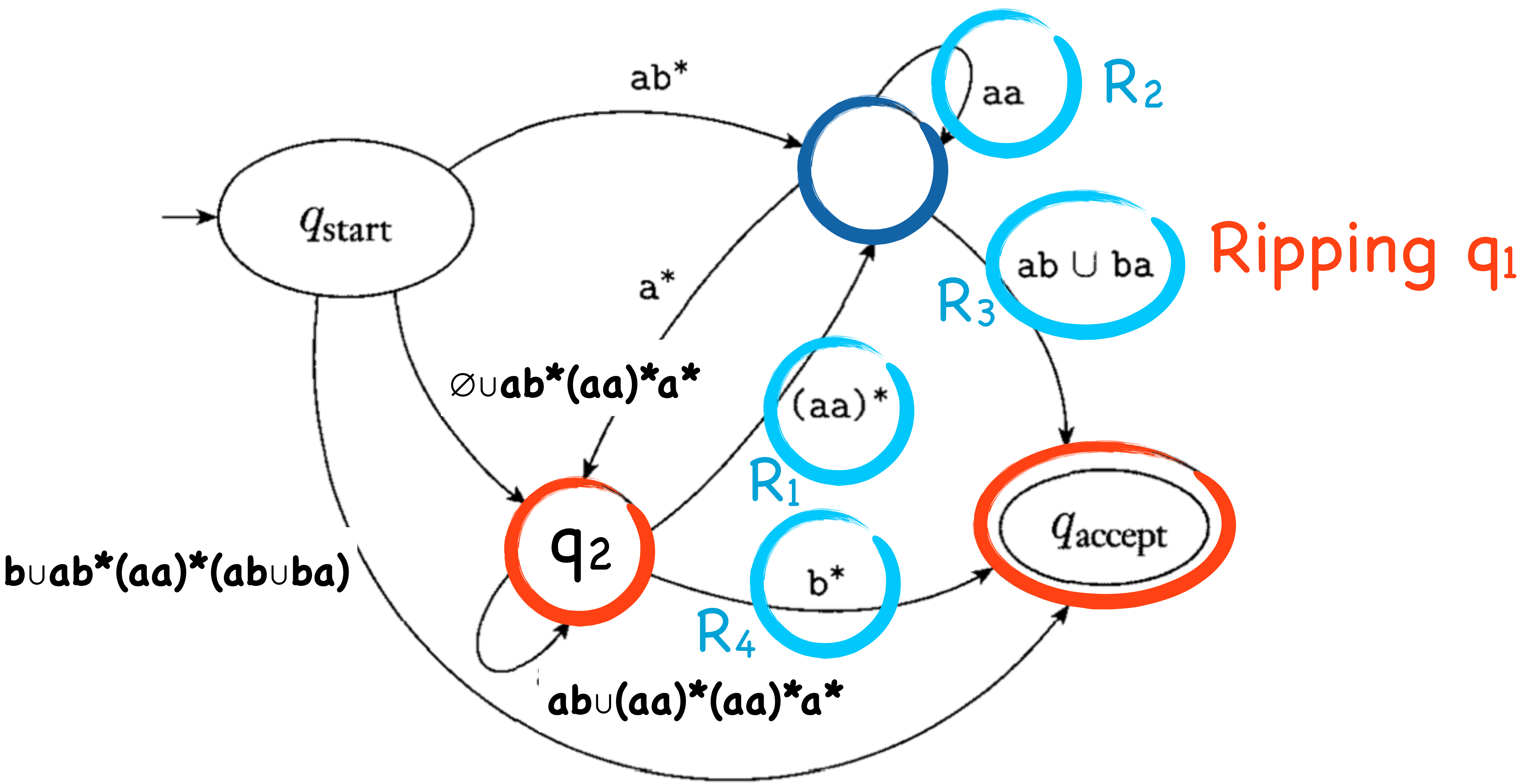
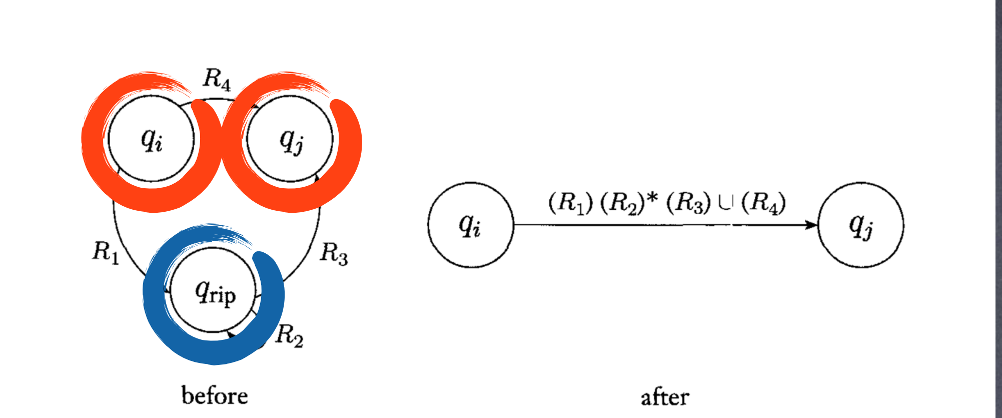


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



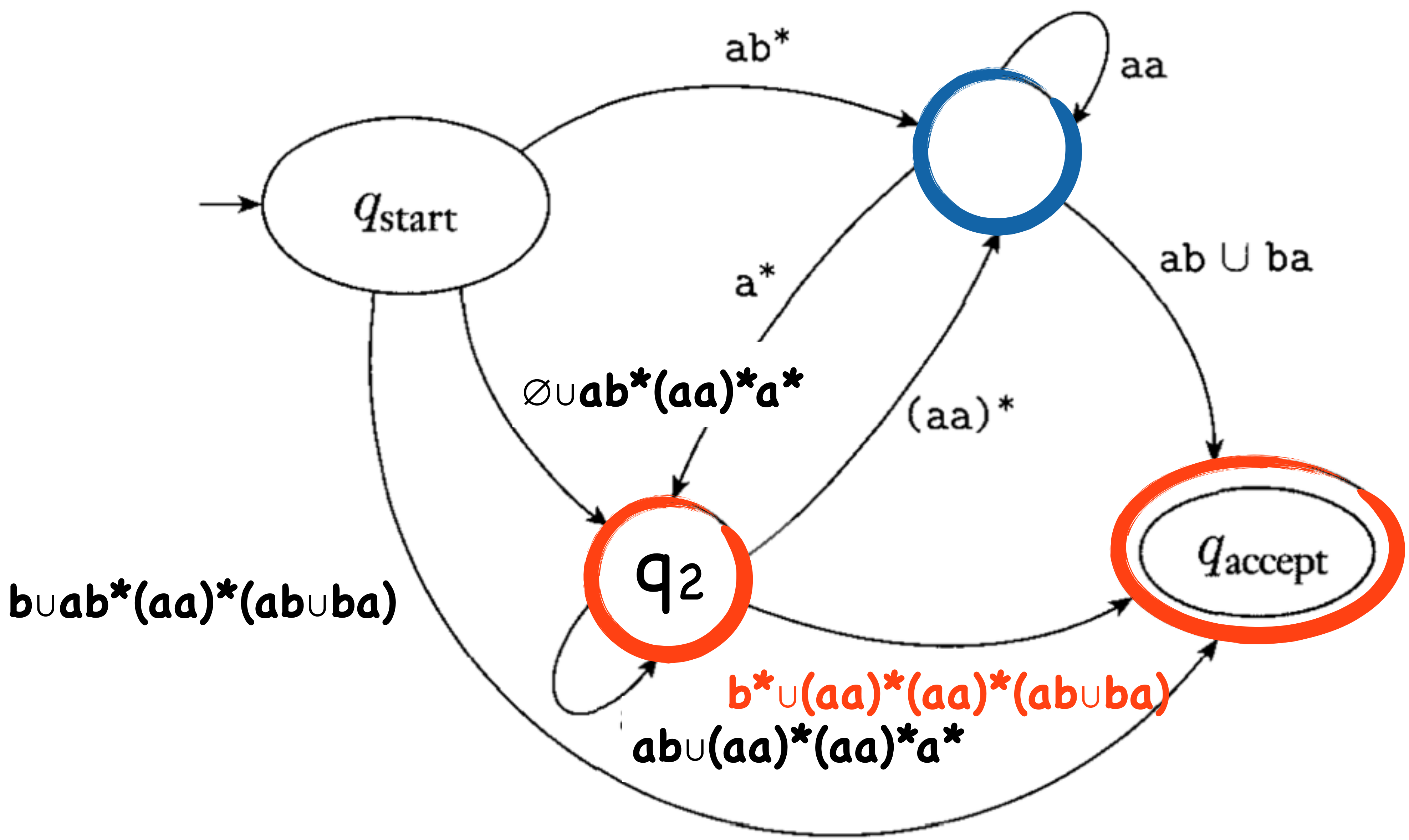
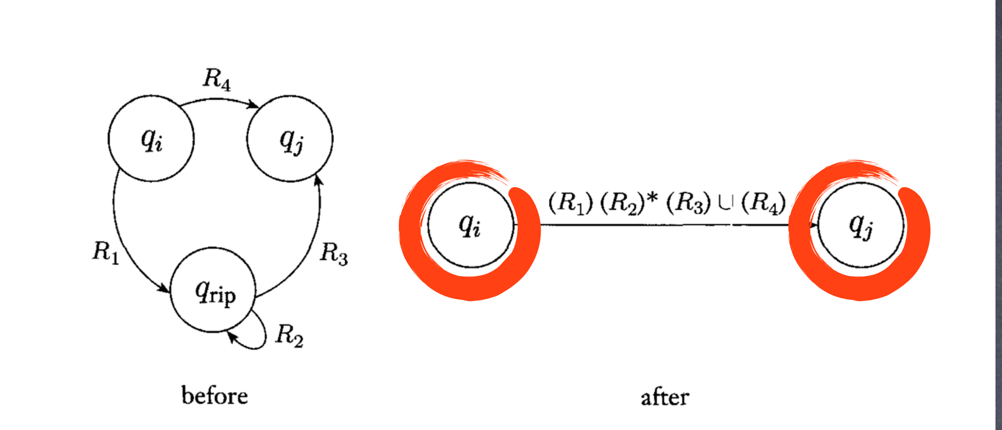


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



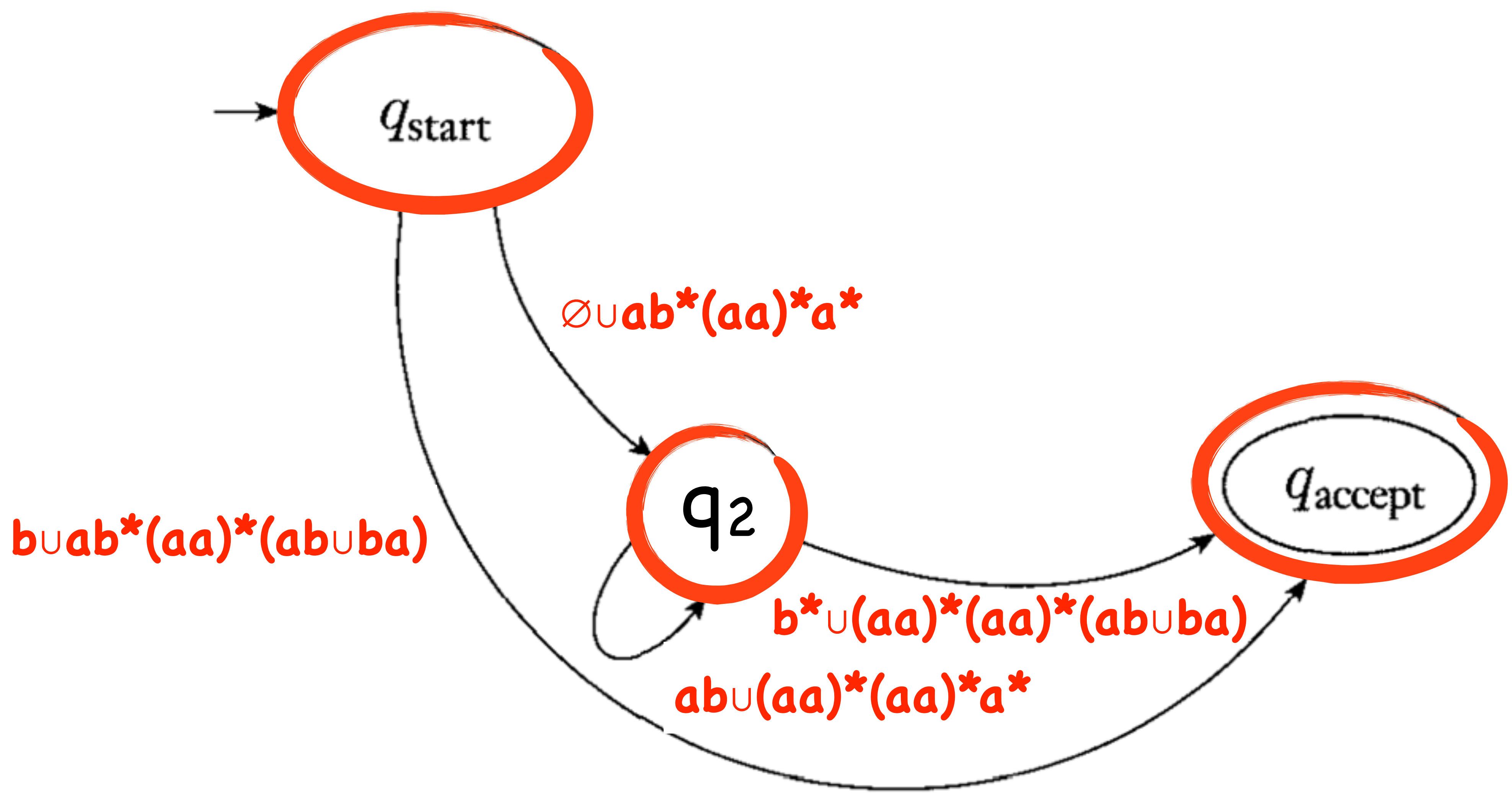
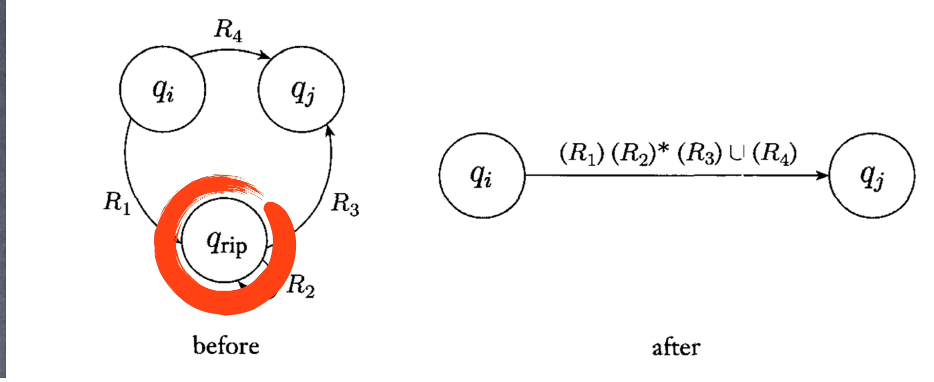


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



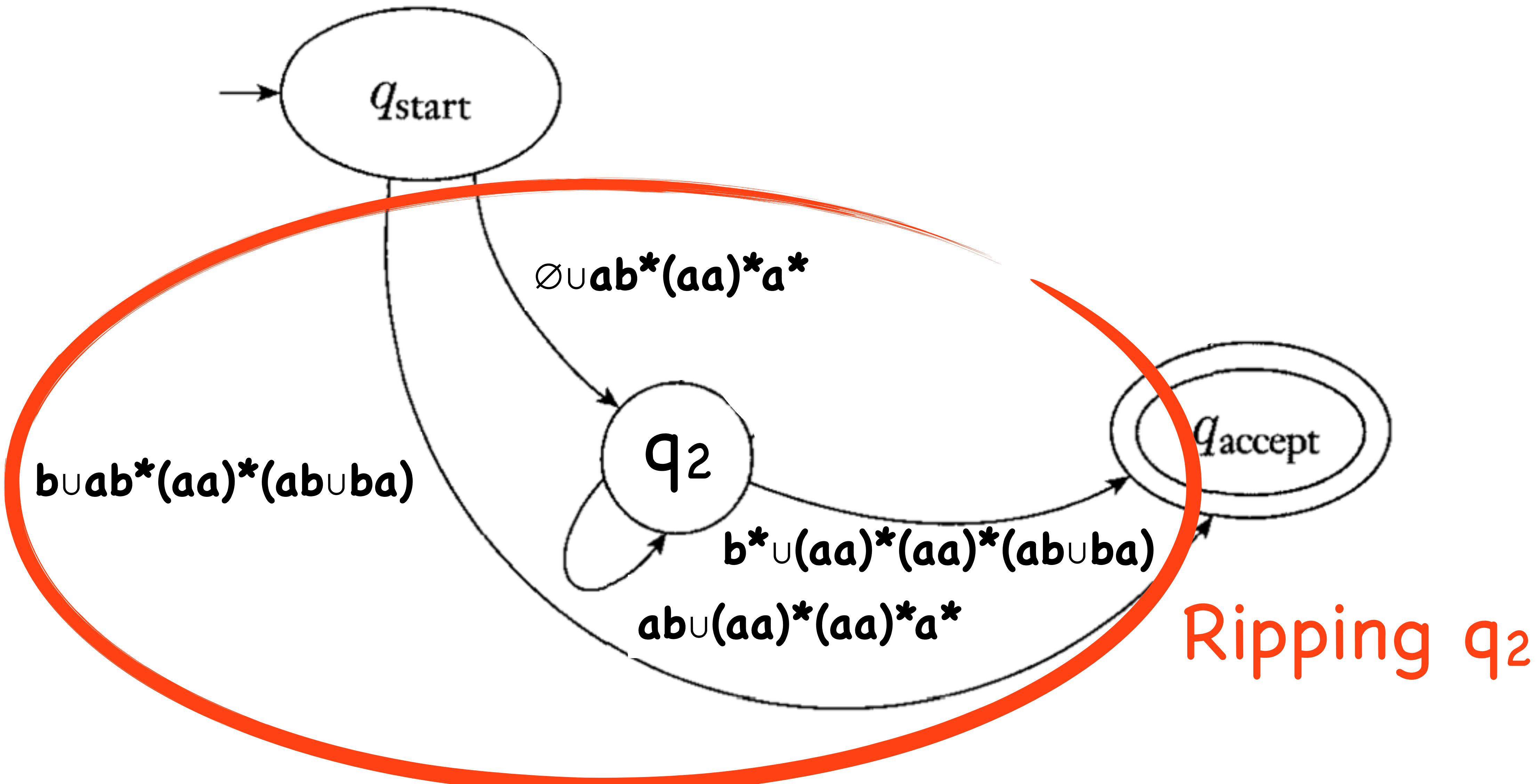
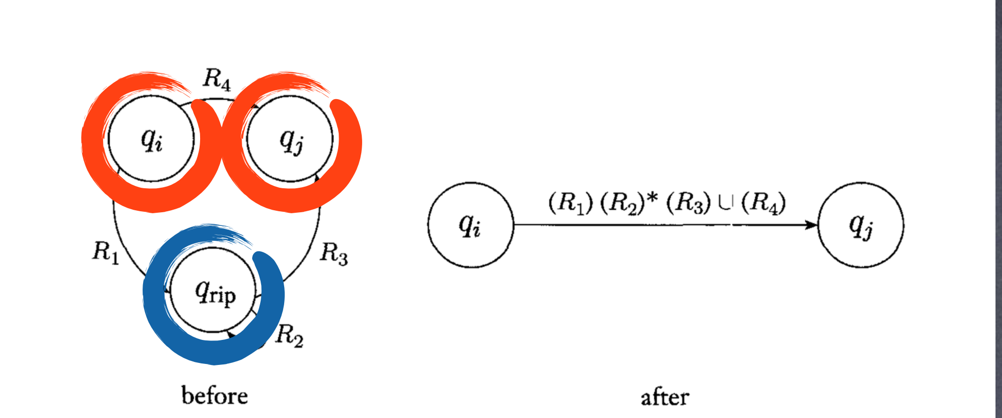


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



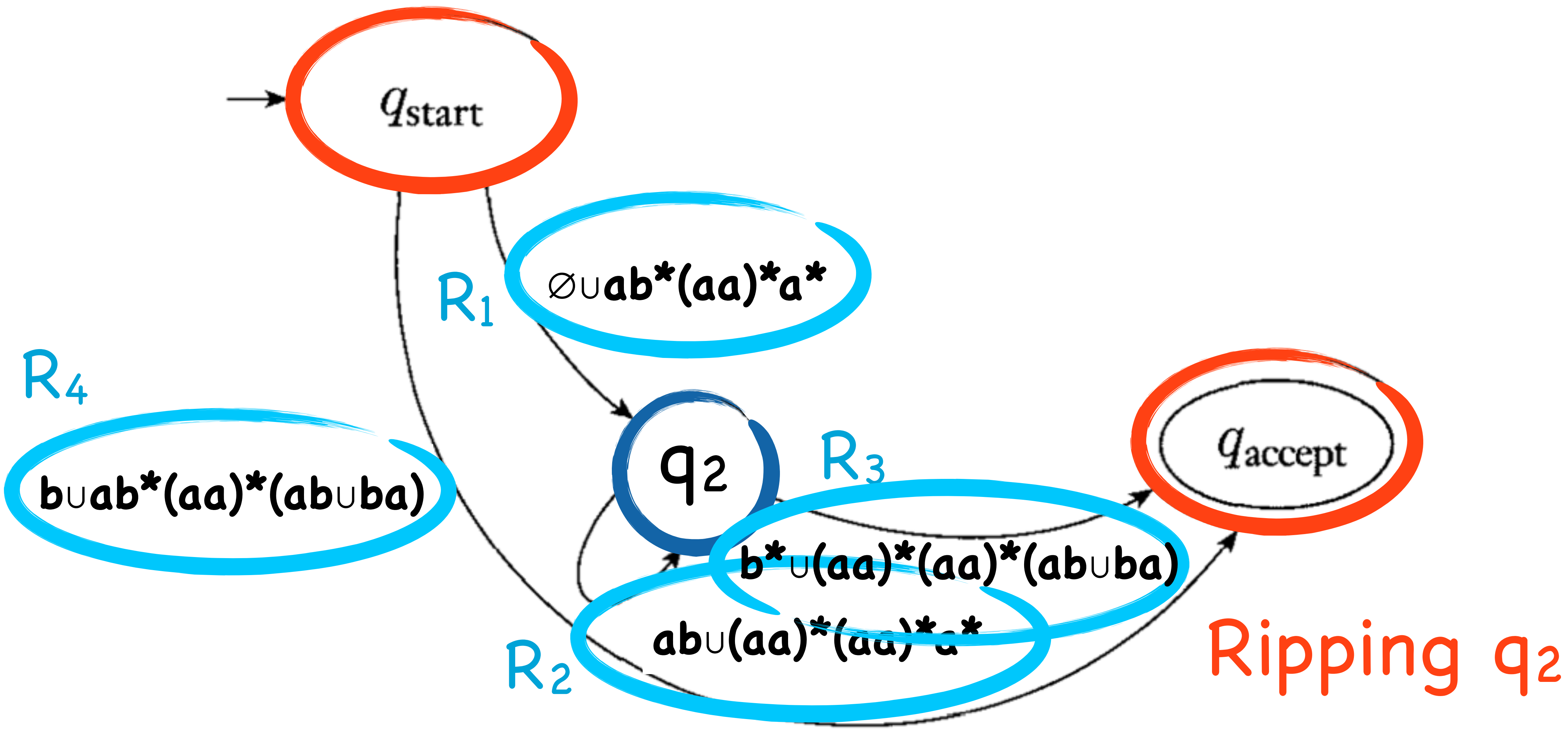
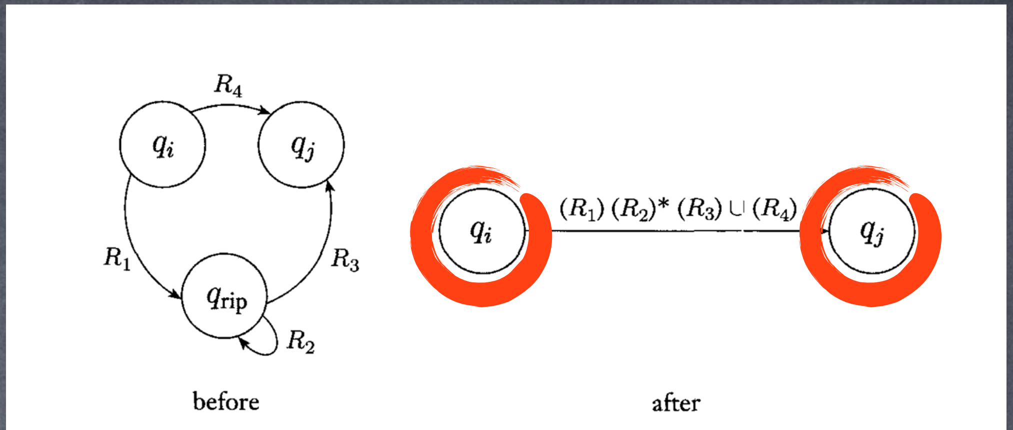


FIGURE 1.61
A generalized nondeterministic finite automaton

Ripping of a GNFA



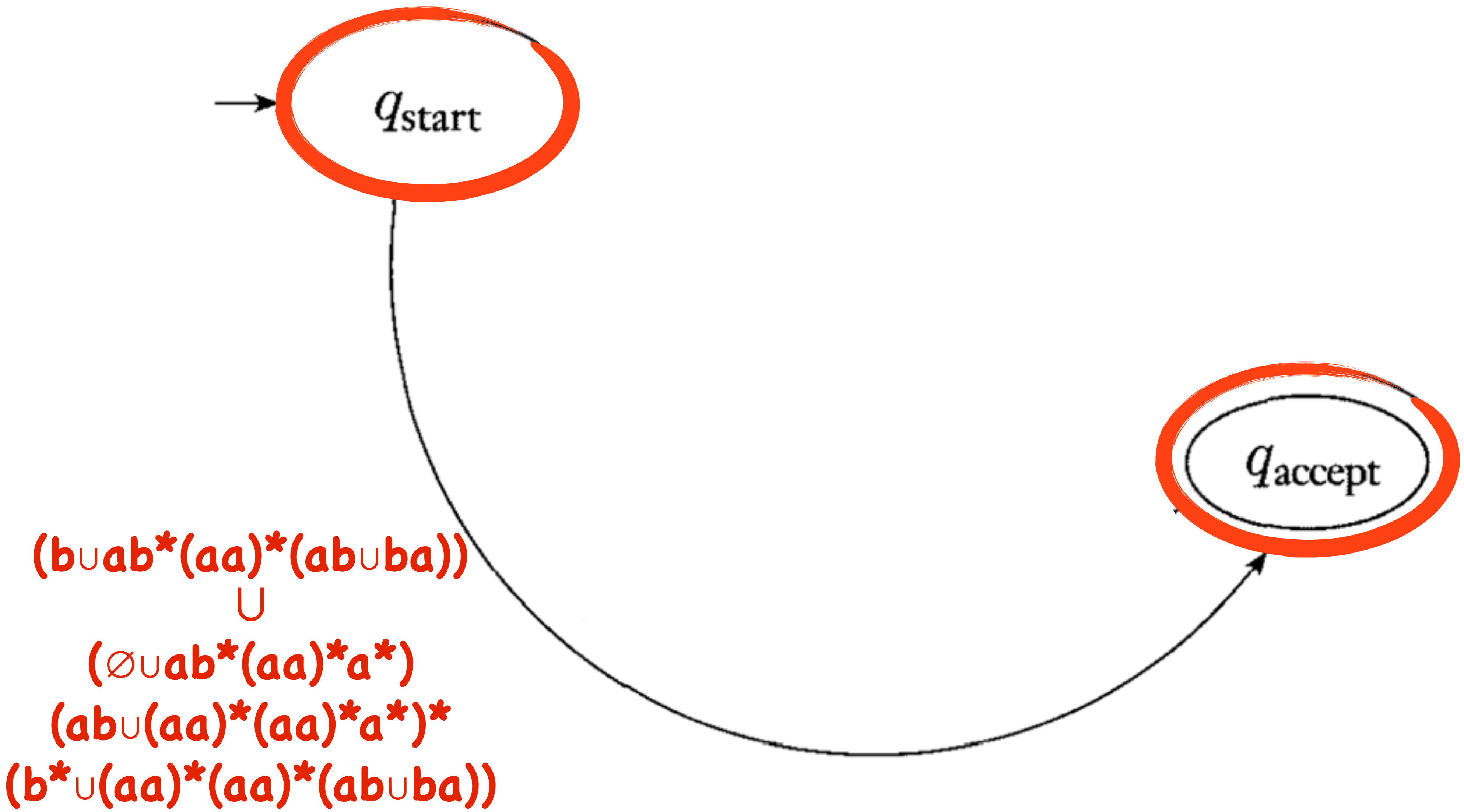


FIGURE 1.61
A generalized nondeterministic finite automaton

GNFA → Reg. Expression

CLAIM 1.65

For any GNFA G, CONVERT(G) is equivalent to G.

We prove this claim by induction on k, the number of states of the GNFA.

"equivalent" means L(CONVERT(G)) = L(G)

GNFA → Reg. Expression

- Induction basis
- Let G be a GNFA with exactly k=2 states. Because of the special form of our GNFA, the two states are the start and accept states. The regular expression on the transition from q_{start} to q_{accept} generates the language accepted by this GNFA.

GNFA → Reg. Exp.

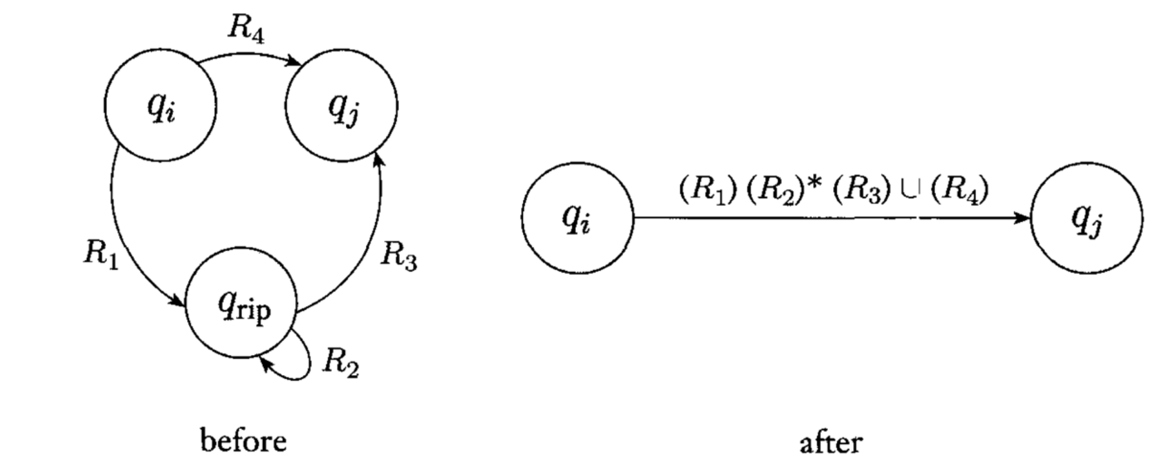


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

- Induction step
- Let G be a GNFA with exactly k>2 states. We assume for induction hypothesis that all GNFA G′ of k-1 states accept the laguage defined by the regular expression obtained via CONVERT, i.e. L(G′)=L(CONVERT(G′)).
- Since k>2 then there exists at least one state q_{rip} which is neither q_{start} nor q_{accept}.

GNFA → Reg. Exp.

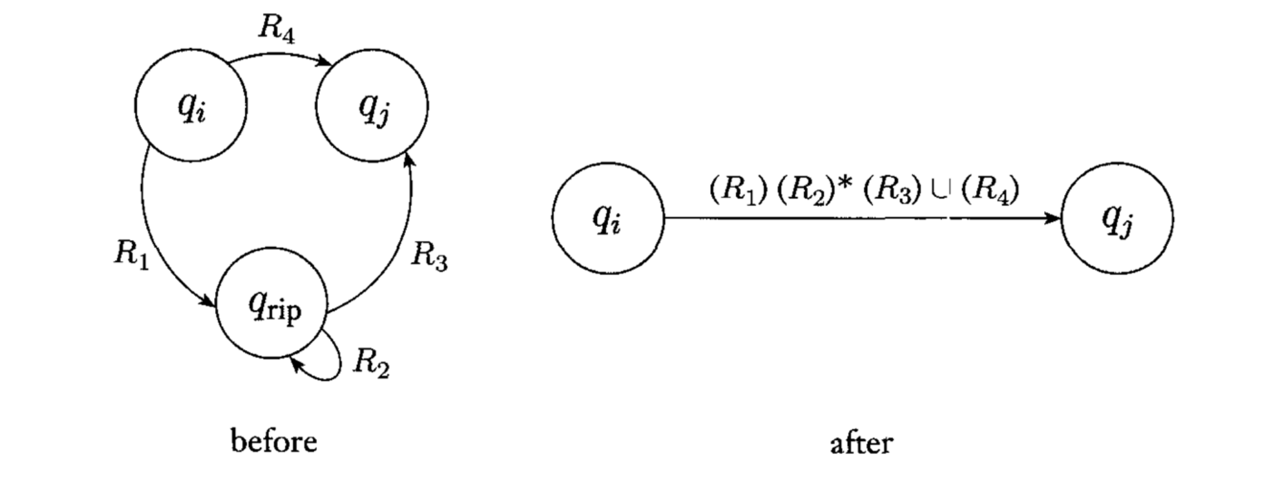


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

- Induction step
- Let G' be, as in CONVERT, the GNFA obtained after ripping q_{rip} from G.
- Let w be a string accepted by G, w∈L(G). Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w.

GNFA → Reg. Exp.

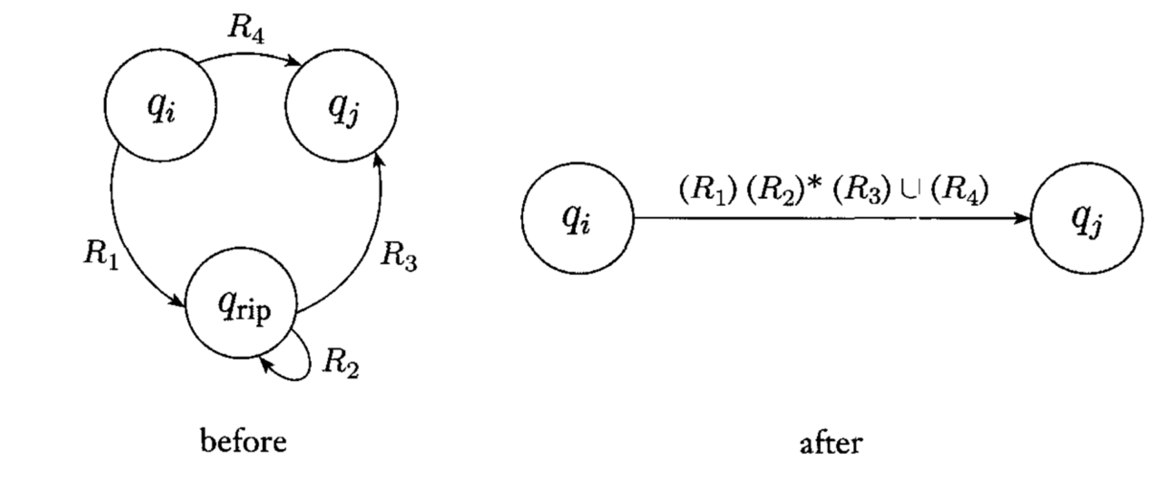
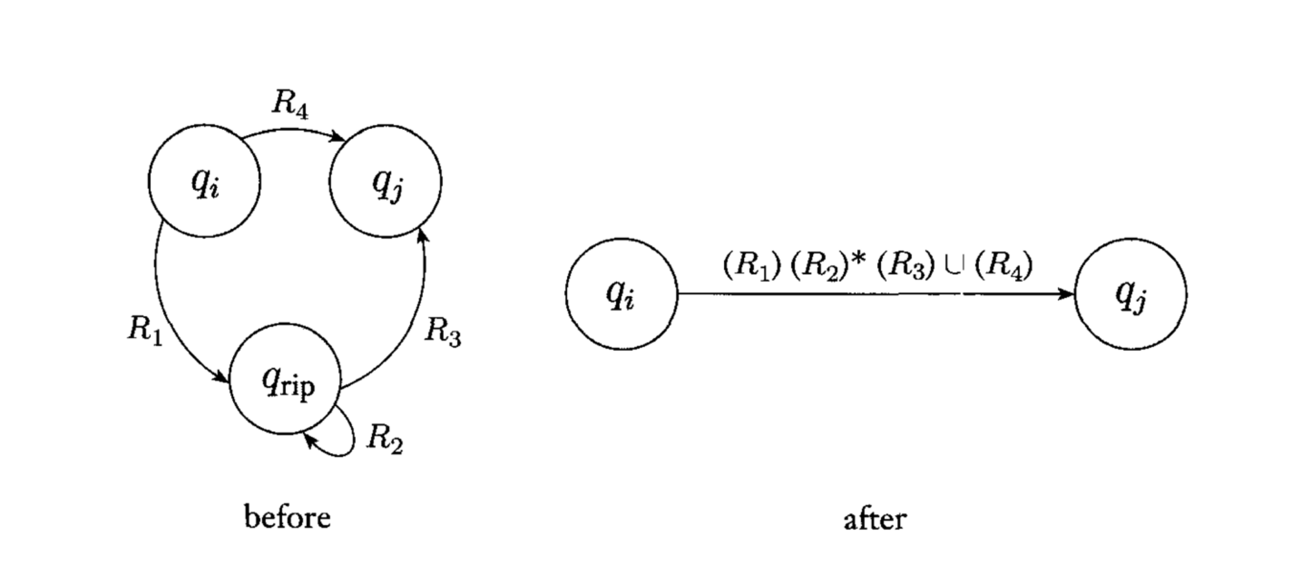


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

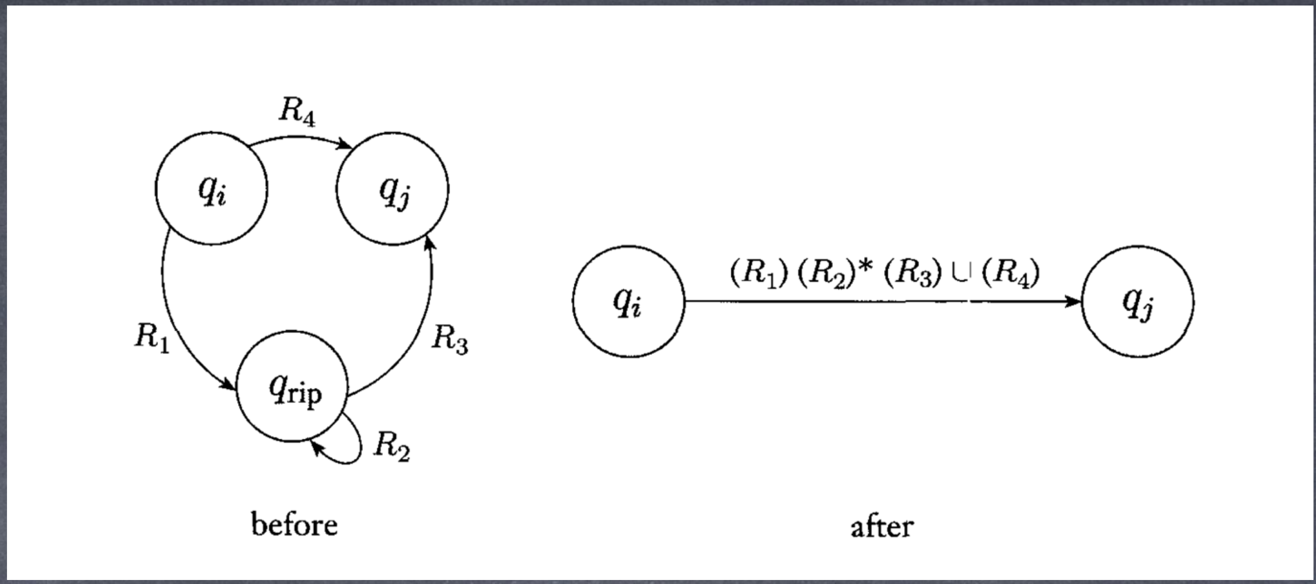
- If q_{rip} is not a state of the sequence, then the very same exact sequence will accept w in G′ because its transitions R_4 contain all those R_4 in G (except for q_{rip}) in a union with new possibilities related to ripping q_{rip}.

GNFA → Reg. Exp.



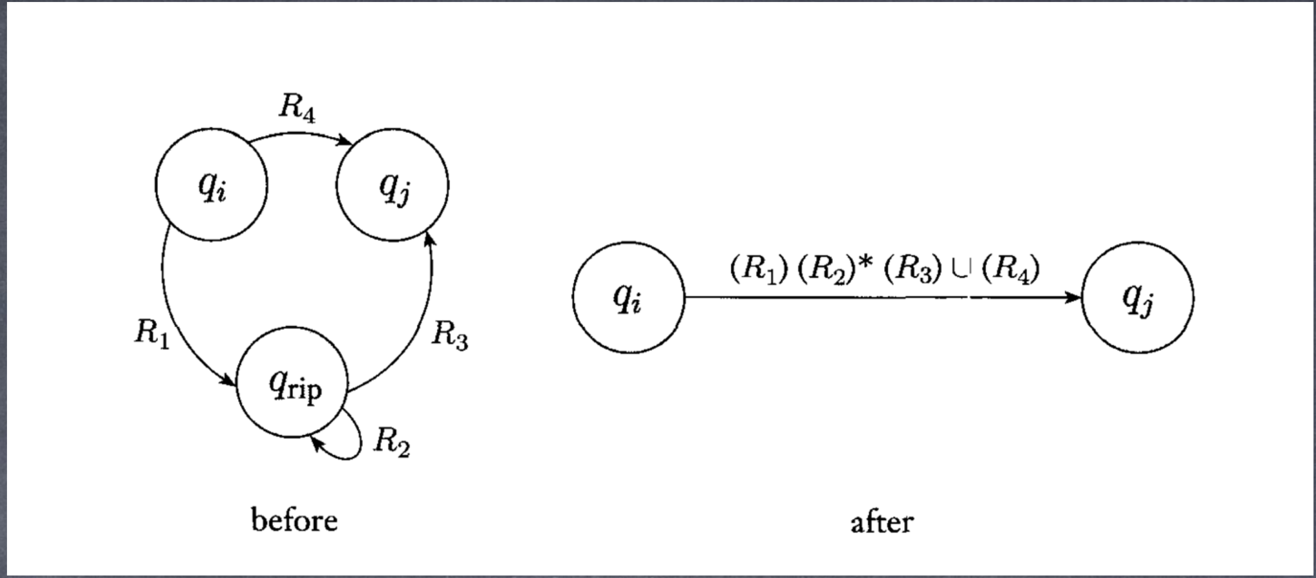
- If q_{rip} is a state of the sequence, then the same sequence (but with all q_{rip} removed) will accept w in G′. That's because any three elements in a row q_i,q_{rip},q_j ($q_i \neq q_{rip} \neq q_j$) in G's accepting sequence, will be processed identically through states q_i,q_j in G′. Remember that the transitions for q_i,q_j in G′ contain all those $R_1(R_2)^*R_3$ from G involving q_{rip} in a union with older possibilities (R_4). (we can deal with $q_i,q_{rip},\ldots,q_{rip},q_j$ similarly.)

GNFA → Reg. Exp.



- This proved "if w∈L(G) then w∈L(G′)". We should also prove "if w∈L(G′) then w∈L(G)".
- Let w be a string accepted by G′, i.e. w∈L(G′). Consider an accepting sequence $q_{start}, q_1, q_2, \ldots, q_{accept}$ for string w. Consider any two consecutive states q_i, q_{i+1}. The same portion of w is processed in G in either part of the union, $R_1(R_2)^*R_3$ or R_4, along the transition between q_i and q_{i+1}.

GNFA → Reg. Exp.



- If the portion of w is generated by R_4 in G′ then it is also generated by R_4 in G. If the portion of w is generated by $R_1(R_2)^*R_3$ in G′ then there exists an m such that it is generated by $R_1(R_2)^mR_3$ and it is also generated in G by R_1, going through q_{rip} m times via R_2 and finally R_3. Thus q_i,q_{i+1} is replaced by $q_i,q_{rip},\ldots,q_{rip},q_{i+1}$.
- We conclude that if $w \in L(G')$ then $w \in L(G)$.

GNFA → Reg. Exp.

- Combining both statements we get L(G′)=L(G).
- By induction hypothesis L(G′)=L(CONVERT(G′)) because G′ contains k-1 states. By construction, CONVERT(G)=CONVERT(G′). Therefore
  L(G)=L(CONVERT(G))=L(CONVERT(G′))=L(G′).

QED

COMP-330
Theory of Computation

Fall 2019 -- Prof. Claude Crépeau

Lec. 7 : Regular Expressions & GNFA